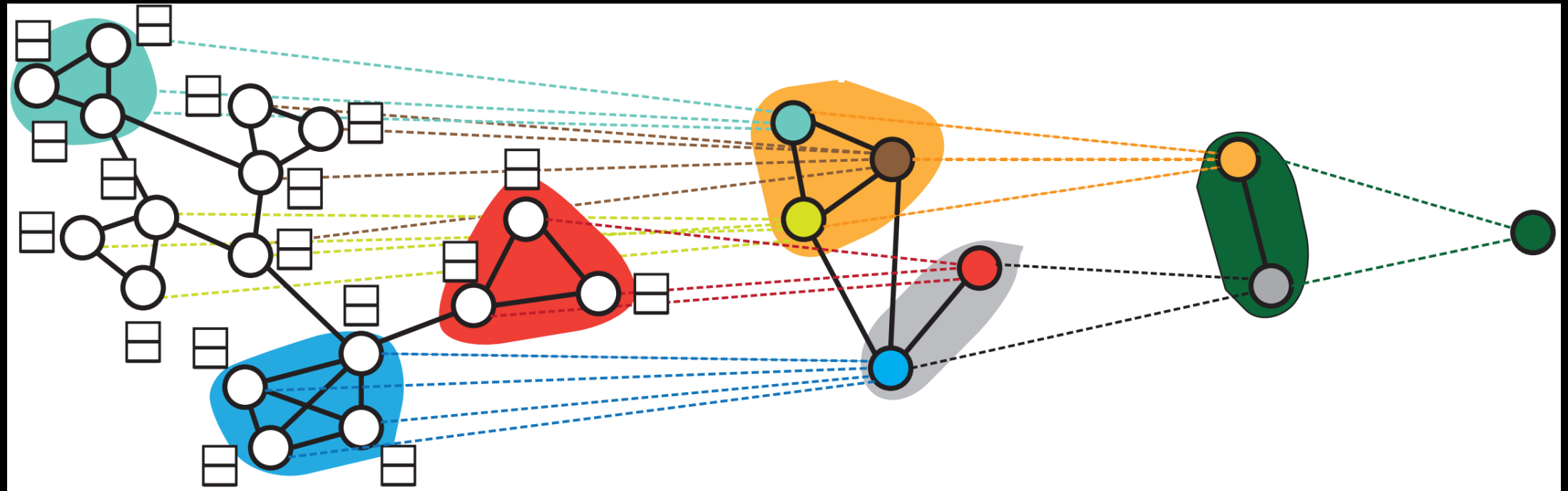


https://arxiv.org/pdf/1812.04202.pdf

Graph Convolutional Networks

Mehrnaz Amjadi

References

Main reference

- Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." arXiv preprint arXiv:1901.00596 (2019).

Further reading:

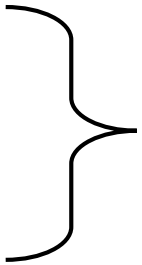
Zhang, Ziwei, Peng Cui, and Wenwu Zhu. "Deep learning on graphs: A survey." arXiv preprint arXiv:1812.04202 (2018).

Zhou, Jie, et al. "Graph Neural Networks: A Review of Methods and Applications." arXiv preprint arXiv:1812.08434 (2018).

Deep Learning Success in AI and ML

- Expressive power to extract complex patterns underlying data

- **ML task**
  - object detection
  - machine translation
  - speech recognition



~~handcrafted feature engineering~~

CNN, LSTM, Autoencoders

- **Success of deep learning**
  - ✓ rapidly developing computational resources (e.g. GPU)
  - ✓ availability of large training data
  - ✓ effective to extract latent representation from Euclidean data (image, text, and video)

Graphs

- **Non-Euclidean**
- **Predominant in the real world**
- **Representing objects and relationships**
  - Social networks (classification in citation network)
  - E-commerce networks (recommendation systems)
  - Biology networks (molecular graph for medicine)
  - Traffic networks
- **Known to have complicated structures**

How to utilize deep learning for graph data analysis?

Challenges

- **Irregular domain**: no clear grid structure

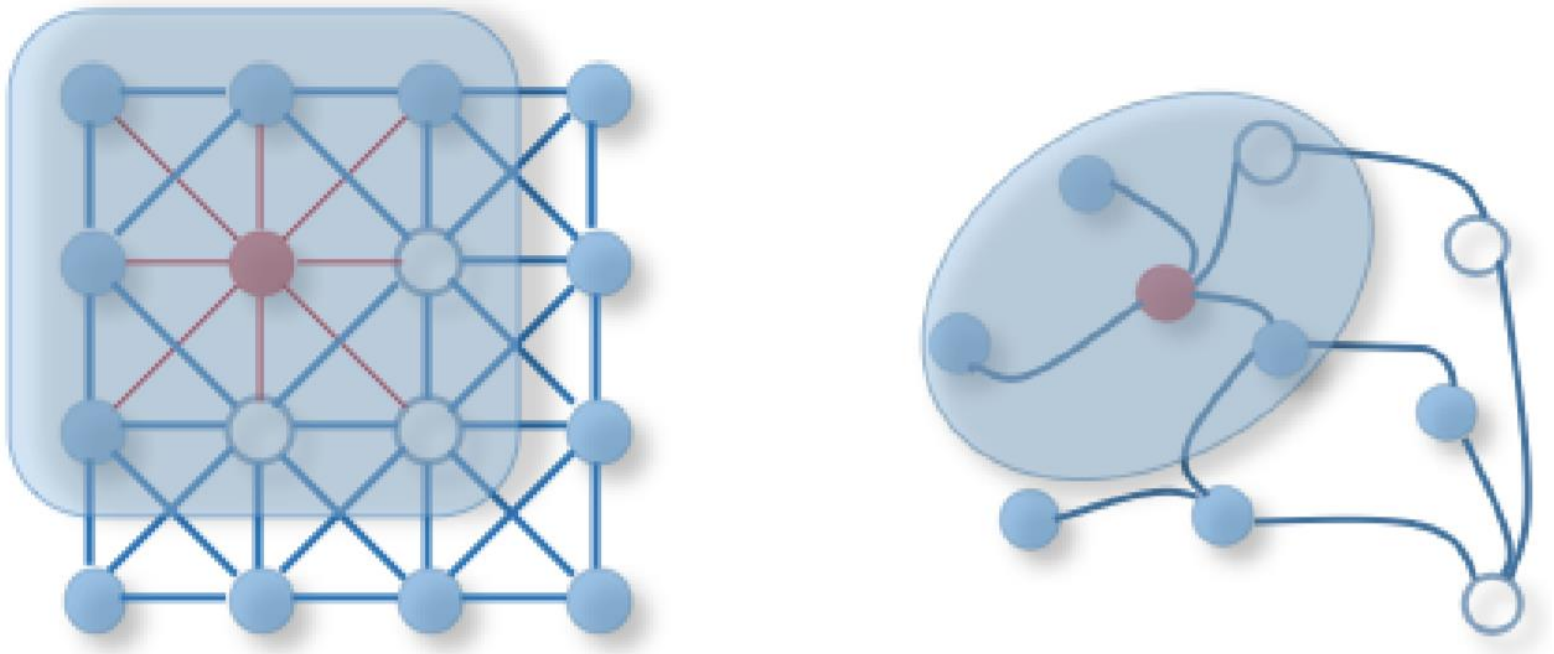
- **Varying structures and tasks**
  - variable size of unordered nodes
  - different number of neighbors
  - (un)weighted, (un)signed
  - Node/Graph focused

New generalization and operation definition

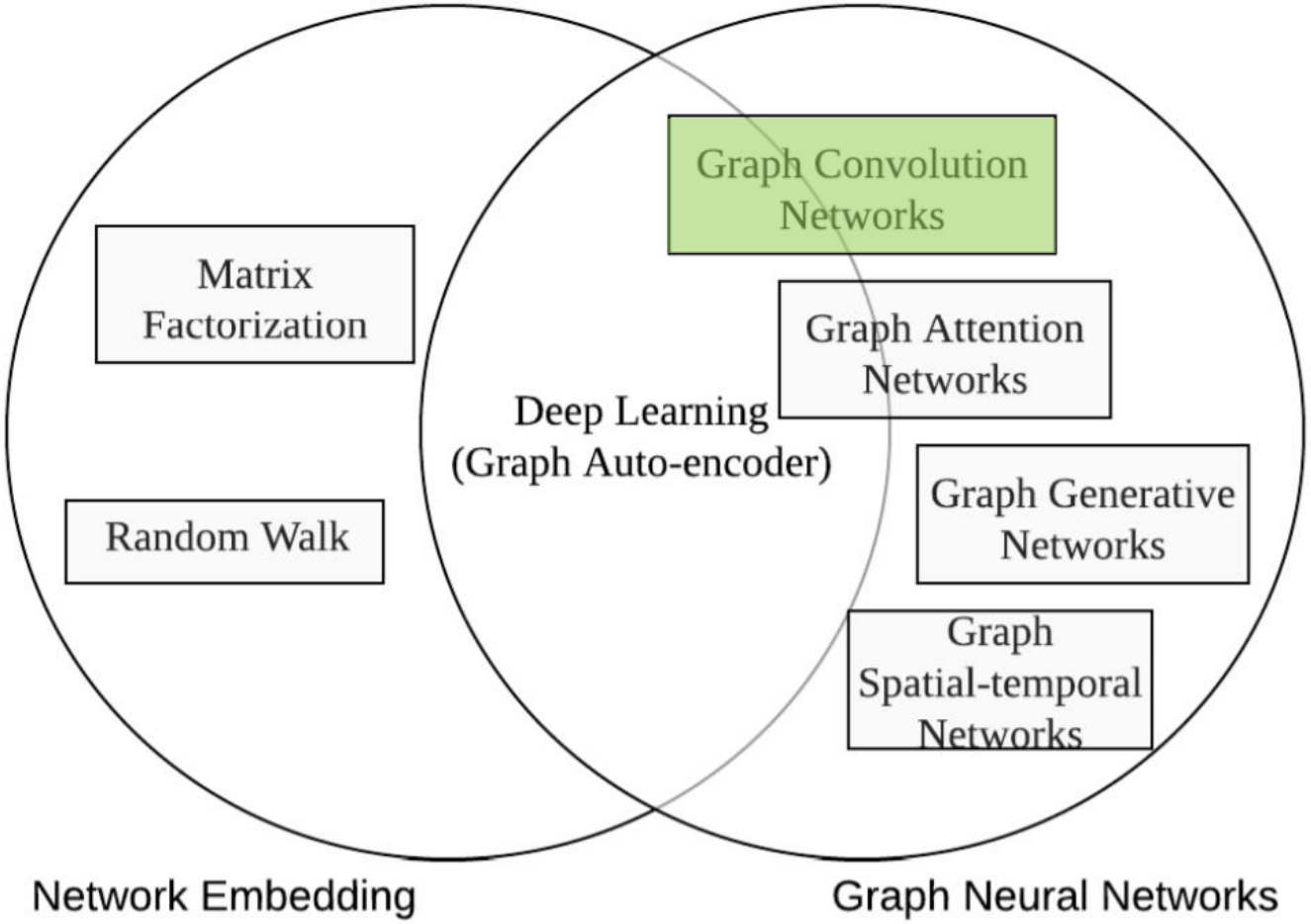
- **Scalability and parallelization**
  - interconnectivity of nodes and edges
  - Interdependency of instances (nodes and its neighbors) ~~independent instances core assumption~~

- **Interdiscipline:** difficult design models

2D Convolution vs. Graph Convolution



Graph neural networks vs. network embedding

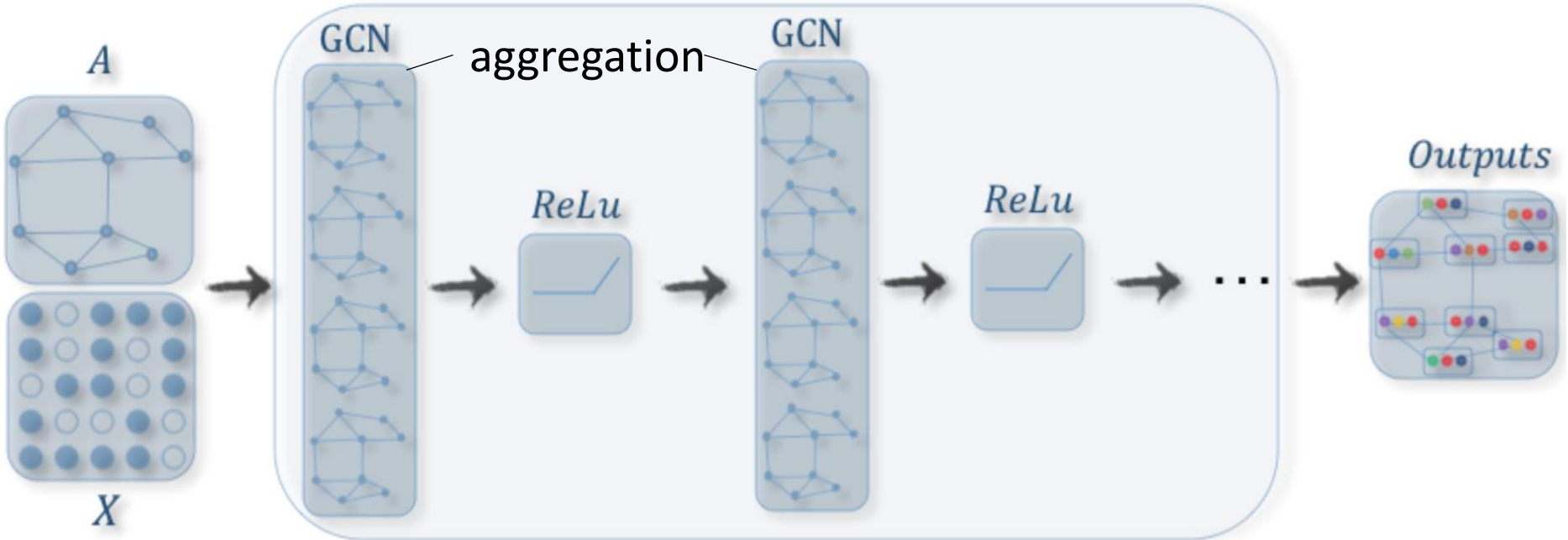


Commonly used notations

Notations	Descriptions
$\|\cdot\|$	The length of a set
$\odot$	Element-wise product.
A^T	Transpose of vector/matrix A.
$[A, B]$	Concatenation of A and B.
$\mathcal{G}$	A graph
V	The set of nodes in a graph
v_i	A node $v_i \in V$
$N(v)$	the neighbors of node v
E	The set of edges in a graph
e_{ij}	An edge $e_{ij} \in E$
$X \in R^{N \times D}$	The feature matrix of a graph.
$x \in R^N$	The feature vector of a graph in case of $D = 1$.
$X_i \in R^D$	The feature vector of the node v_i.
N	The number of nodes, $N = \|V\|$.
M	The number of edges, $M = \|E\|$.
D	The dimension of a node vector.
T	The total number of time steps in time series.

Graph Convolution Networks (GCNs)

- **Generalize** convolution from traditional data (images or grids) to graph data.

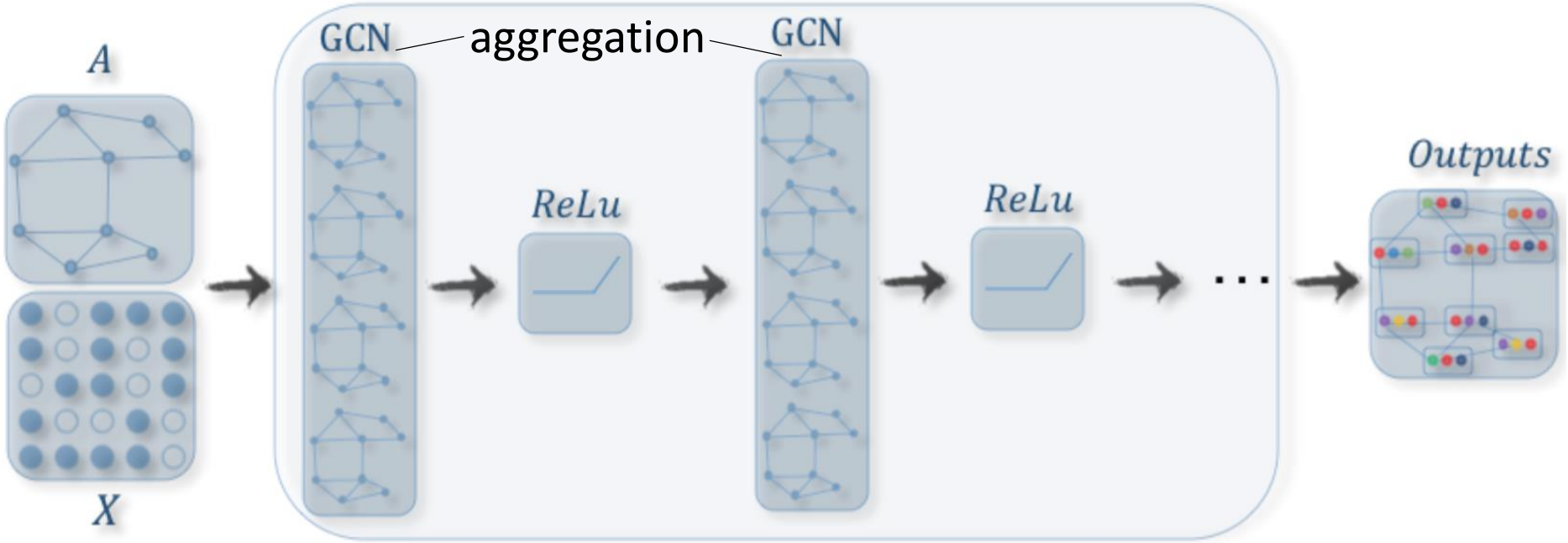


- **Learn a function f** to generate a node v_i's representation

  by aggregating its own features x_i and neighbors' features x_j , where $j \in N(v_i)$.

Graph Convolution Networks (GCNs)

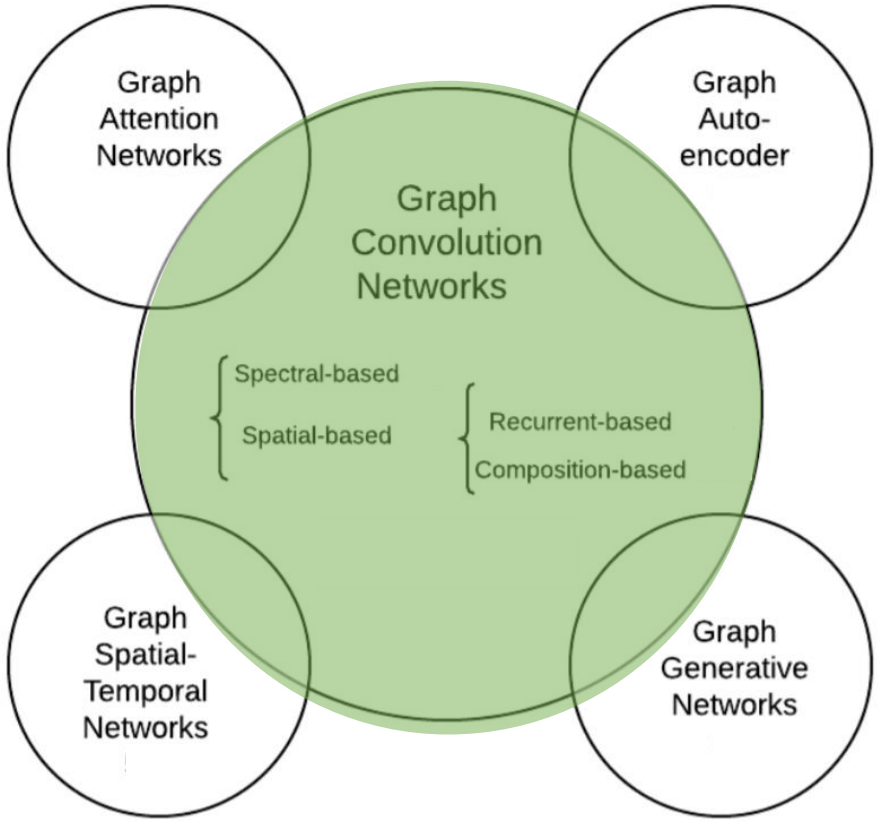
- A GCN layer encapsulates each node's hidden representation
  by aggregating feature information from its neighbors → **Node level**



- Final hidden representation of each node receives messages from a further neighborhood by **stacking** multiple layers

Categorization of Graph Neural Networks

- GCNs play a **central role** in building up many other complex graph neural network models



GCN Frameworks

- GCNs try to define graph convolutions using
  - graph spectral theory
  - spatial locality
- **Input:** graph structure and node content information
- **Output:** focus on different graph analytics task

Different graph analytics task

- **Node-level**

  node regression and classification tasks (GCN+ MLP/softmax)

- **Edge-level**

  edge classification and link prediction (two nodes' latent representations from GCN as input)

- **Graph-level**

  graph classification (pooling is used to coarse a graph into sub-graphs or to sum/average over the node representations)

End-to-end Training Frameworks

- **Semi-supervised learning for node-level classification**

  single network with **partial** nodes being labeled

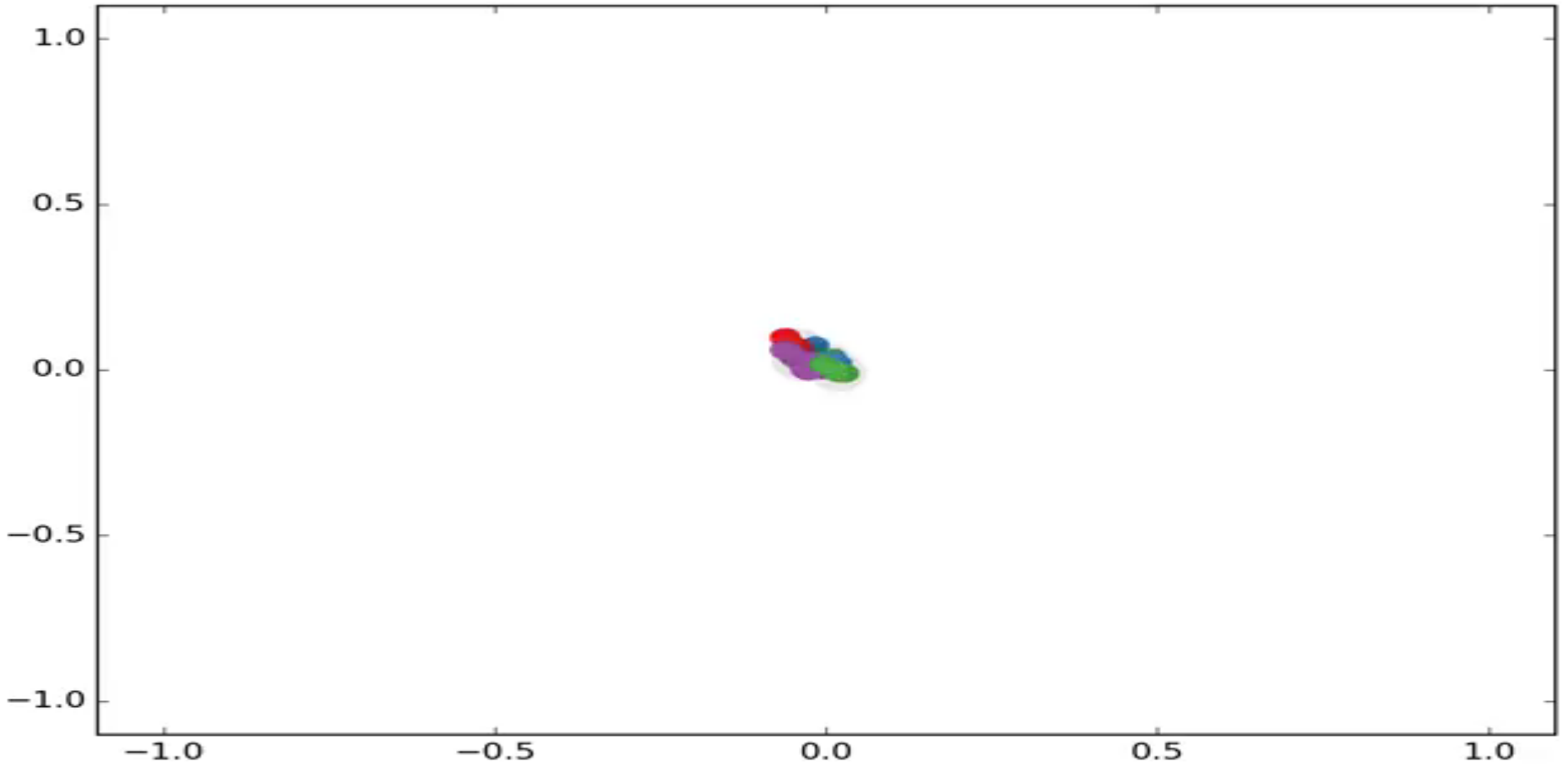
  identifies the class labels for the unlabeled nodes

  stacks a couple of GCN followed by a softmax layer

- Supervised learning for graph-level classification

- Unsupervised learning for graph embedding

Demo: Semi-supervised classification with GCNs



Demo Link

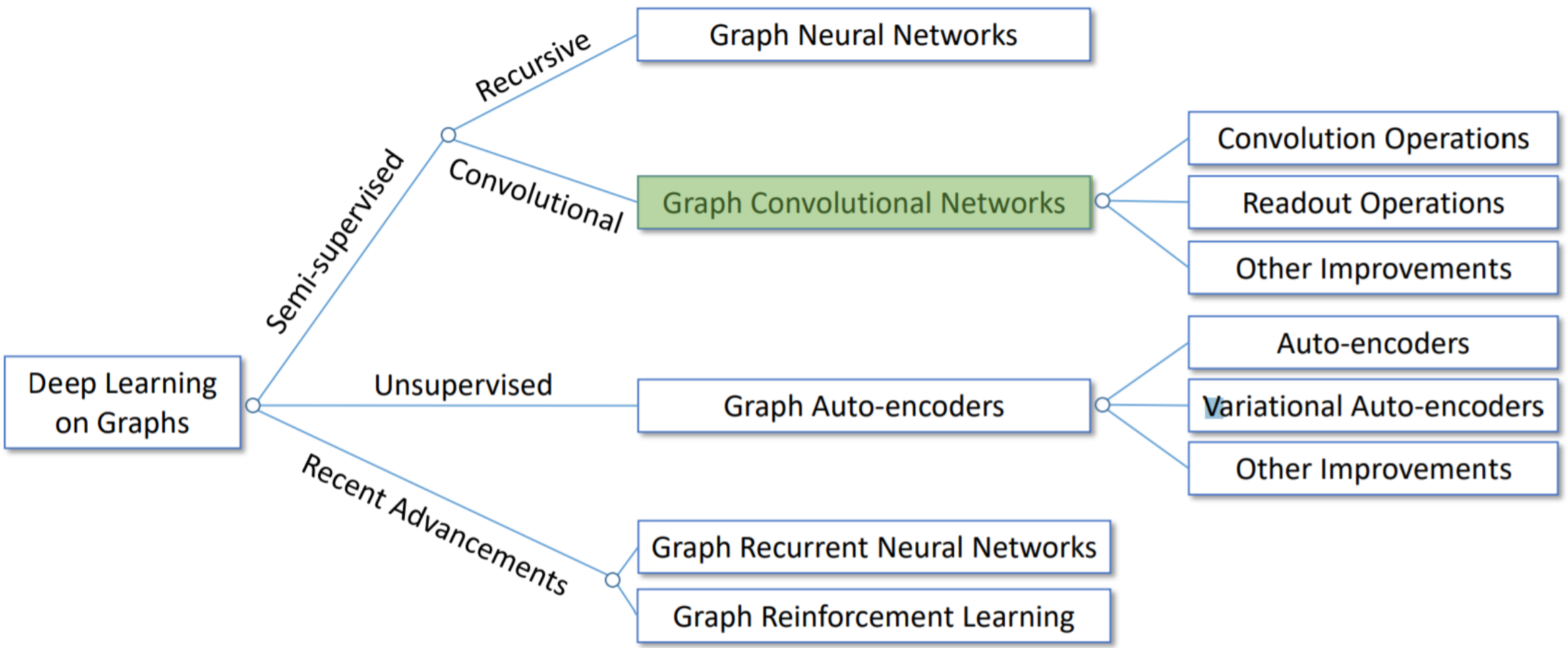
End-to-end Training Frameworks

- Semi-supervised learning for node-level classification
- **Supervised learning for graph-level classification**
  - given a graph, predicts the **class label**(s) for an entire graph
  - combines both GCN layers and **pooling**
  - obtains **representation** for **each node** in every single graph by convolution
  - **pooling** which summarizes the representation vectors of all nodes in a graph
  - Finally applying linear layers and a **softmax** layer
- Unsupervised learning for graph embedding

End-to-end Training Frameworks

- Semi-supervised learning for node-level classification
- Supervised learning for graph-level classification
- **Unsupervised learning for graph embedding**
  - **no class labels** are available in graphs
  - exploit **edge-level** information in two ways
    - autoencoder framework
    - negative sampling

Deep Learning Methods on Graphs

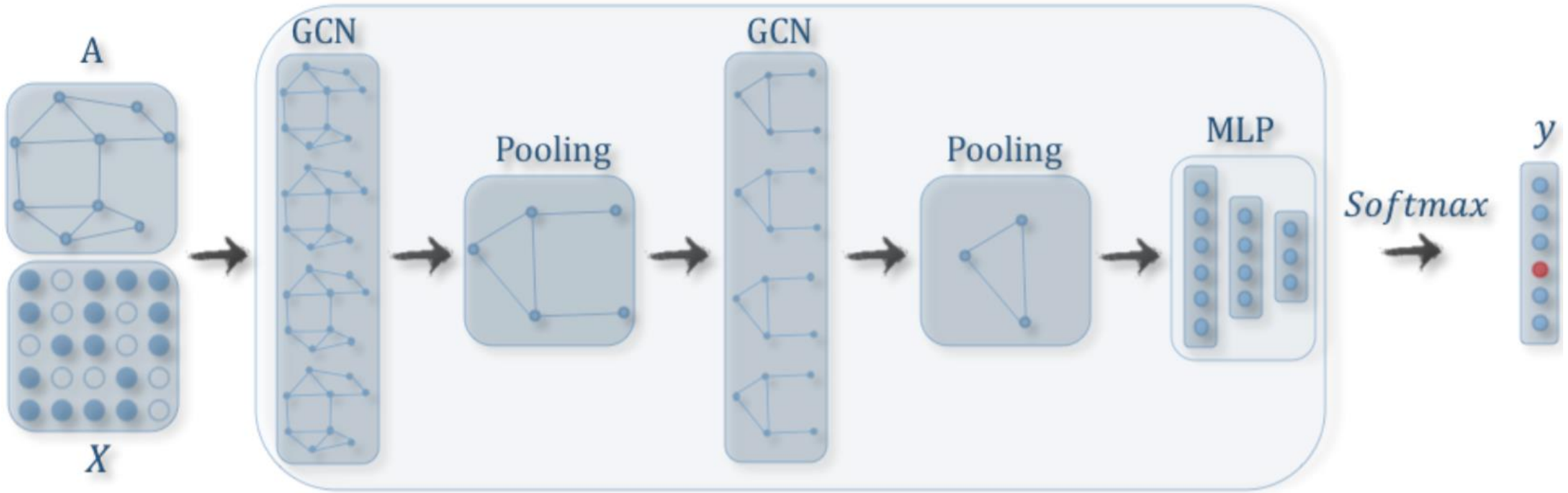


Distinctions of Deep Learning Methods on Graphs

Category	Type	Node Attributes/Labels	Counterparts in Traditional Domains
Graph Neural Networks	Semi-supervised	Yes	Recursive Neural Networks
Graph Convolutional Networks	Semi-supervised	Yes	Convolutional Neural Networks
Graph Autoencoders	Unsupervised	Partial	Autoencoders/Variational Autoencoders
Graph Recurrent Neural Networks	Various	Partial	Recurrent Neural Networks
Graph Reinforcement Learning	Semi-supervised	Yes	Reinforcement Learning

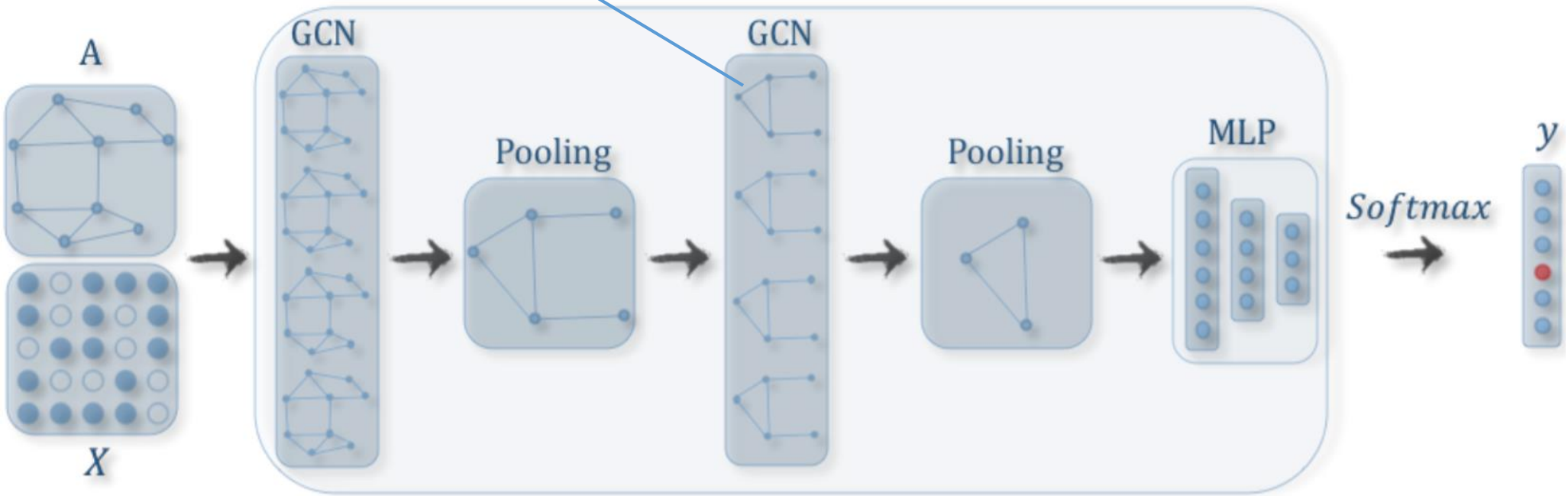
GCN + Pooling

- GCNs operate on the **node level**
- Graph **pooling** can be mixed with the GCN layer
  to coarsen graphs into **high-level** sub-structures



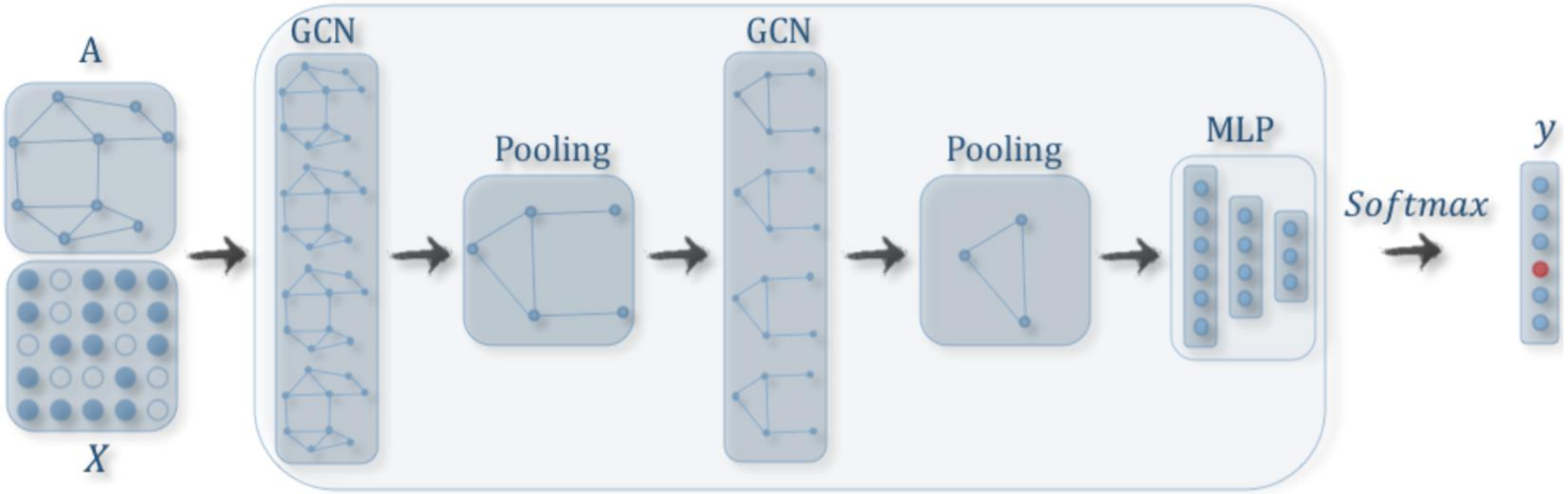
GCN + Pooling

- GCN layer is followed by a pooling layer to **coarsen** a graph into sub-graphs
- node representations on coarsened graphs represent **higher graph-level** representations.



GCN + Pooling

- Output layer is a linear layer with the SoftMax function
  to calculate the **probability** for each graph label
- Extract **graph-level** representations and
  to perform graph classification tasks



GCN Approaches

- **Fundamental** of many complex graph neural network models

- GCNs approaches
  - **spectral-based:**
    - **introducing filters** from the perspective of graph signal processing
    - graph convolution operation is interpreted as **removing noise** from graph signals.
  - **spatial-based:**
    - aggregates feature information from **neighbors**

Spectral-based GCNs

Definition. Let $g_\theta = diag(U^T g)$ as filter

Graph convolution of the input signal $\boldsymbol{x}$ with a filter $\boldsymbol{g_\theta}$ is defined as

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta \mathbf{U}^T \mathbf{x}$$

Spectral-based GCNs

- Solid foundation in graph **signal processing**
- All spectral-based GCNs follow this **definition**.
- The key difference is in the **choice** of the filter $\boldsymbol{g_\theta}$.
- **Inefficient** for big graphs

  because of loading the **whole** graph into the memory to perform GC

Spectral CNN

- Proposed the first spectral convolution neural network
- assumes the filter $\boldsymbol{g_\theta} = \boldsymbol{\Theta_{i,j}^k}$ is a set of learnable parameters
- considers graph signals of **multi**-dimension

Spectral CNN

Defines a graph convolution layer as

$$\mathbf{X}_{:,j}^{k+1} = \sigma(\sum_{i=1}^{f_{k-1}} \mathbf{U}\mathbf{\Theta}_{i,j}^{k}\mathbf{U}^{T}\mathbf{X}_{:,i}^{k}) \quad (j = 1, 2, \cdots, f_k)$$

$\boldsymbol{X_k} \in \boldsymbol{R^{N \times f_{k-1}}}$: input graph signal

$\boldsymbol{N}$: number of nodes

$\boldsymbol{f_{k-1}}$: number of input channels

$\boldsymbol{f_k}$: number of output channels

$\boldsymbol{\Theta_k^{i,j}}$: diagonal matrix filled with learnable parameters

$\boldsymbol{\sigma}$: nonlinear transformation

Spectral CNN Drawback

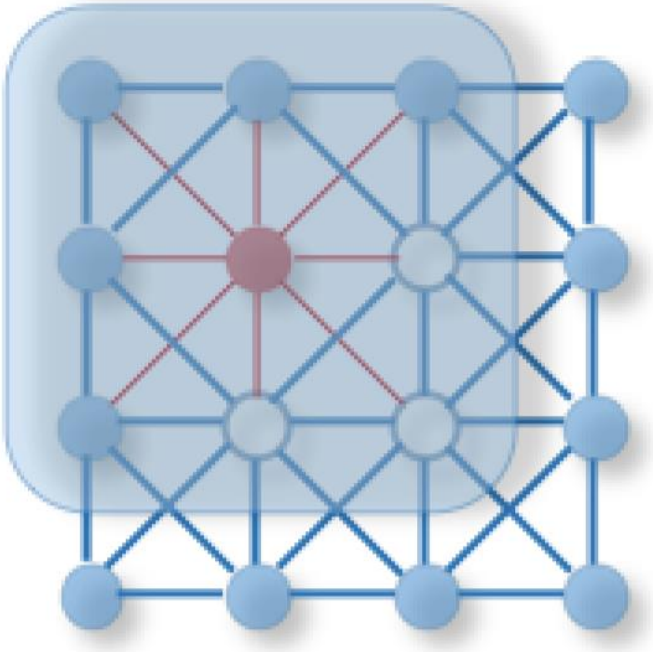
Since it relies on the **Eigen-decomposition** of the Laplacian matrix:

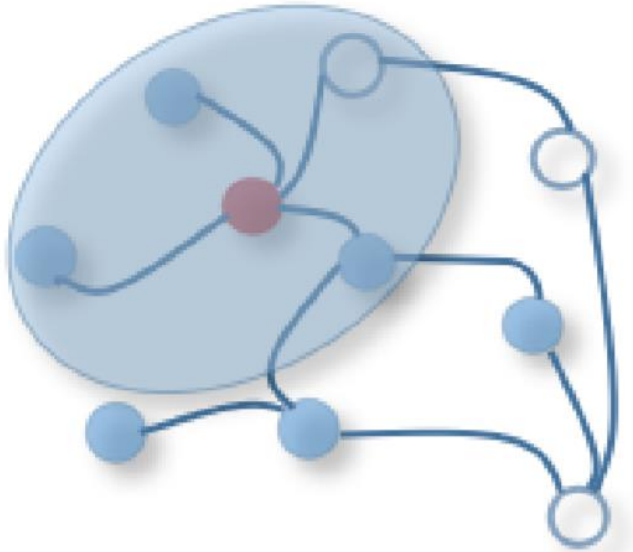
- Any **perturbation** to a graph results in a change of Eigen basis.
- The learned filters are **domain dependent**

  they cannot be applied to a graph with a different structure.
- Eigen decomposition requires $O(N^3)$ computation and $O(N^2)$ memory

Relation of images with graphs

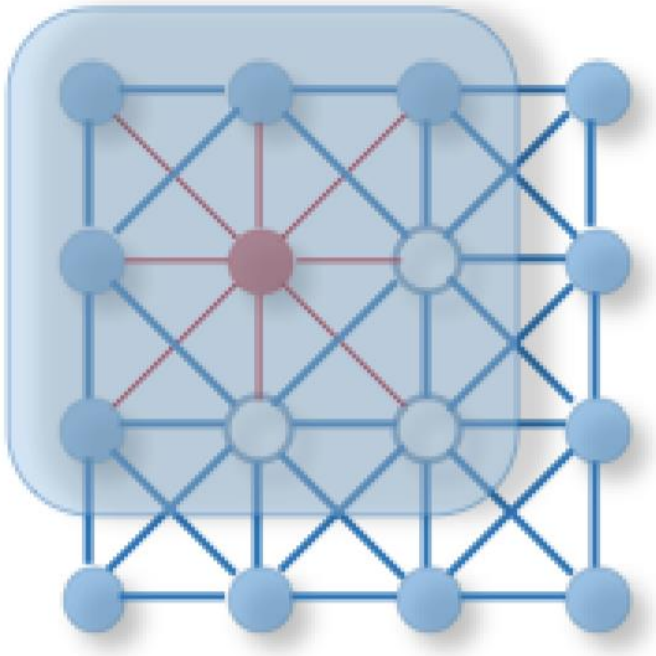
- Images are a **special form** of a graph
- Each pixel represents a node





Relation of images with graphs

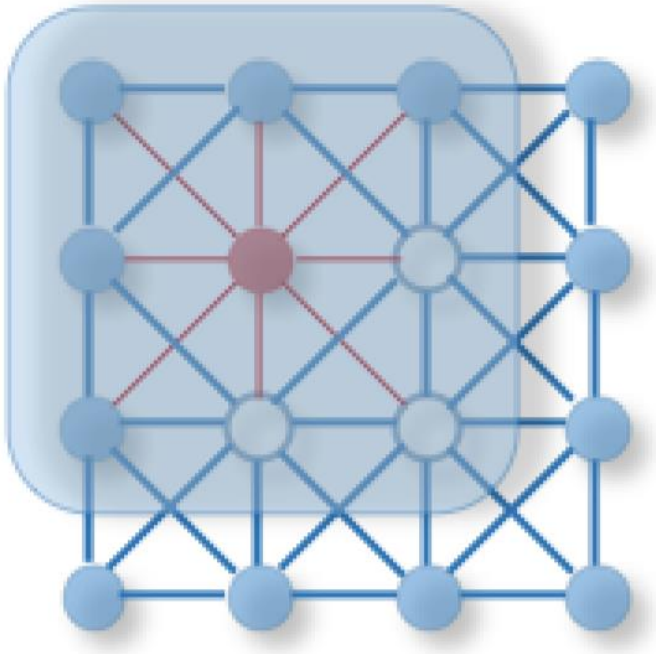
- Each pixel is directly connected to its **nearby** pixels
- The positions of the pixels indicate an **ordering** of a node's neighbors



Relation of images with graphs

- A filter is applied to each patch

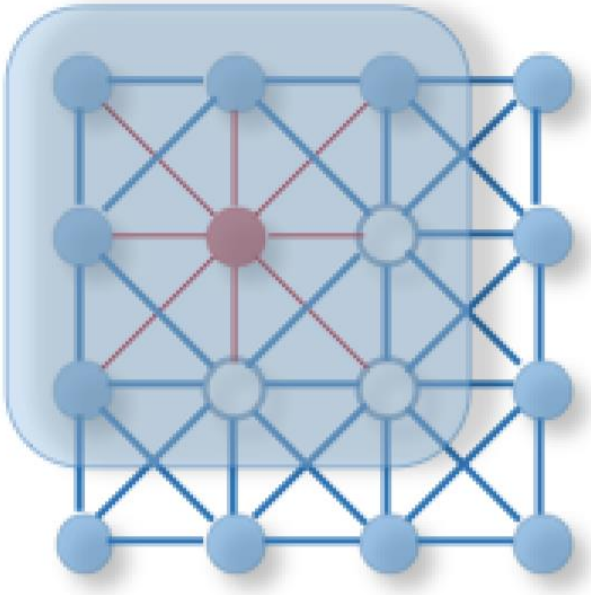
  by taking the **weighted average** of pixel values of the central node and its neighbors across each channel

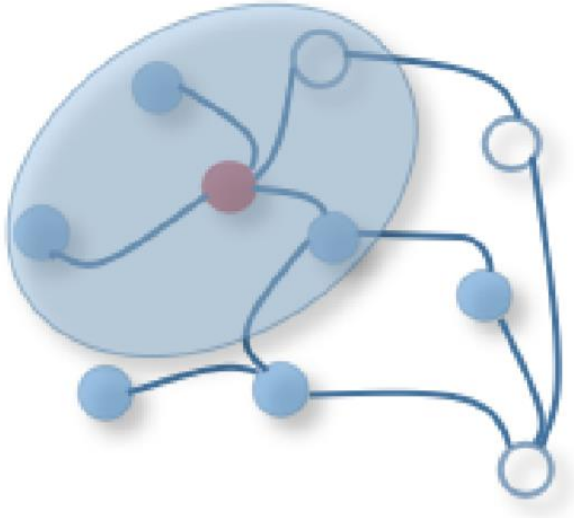


Relation of images with graphs

- **Similar** approach for a general graph
- Spatial-based GCN takes

  the **aggregation** of the central node and its neighbors representation
- Get a **new representation** for the node





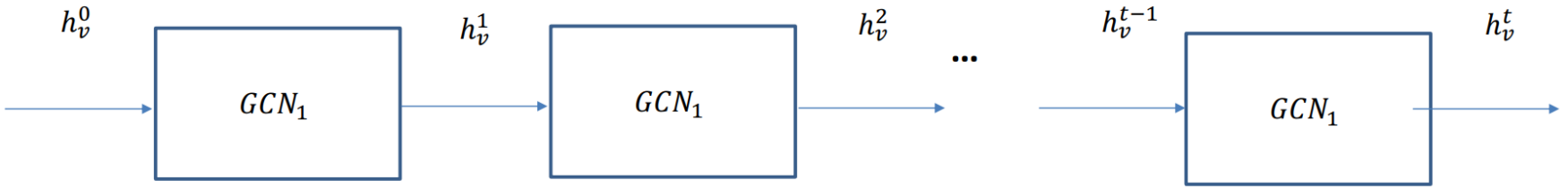
Spatial-based GCNs

- Based on a node's **spatial relations** (Like CNN on images)

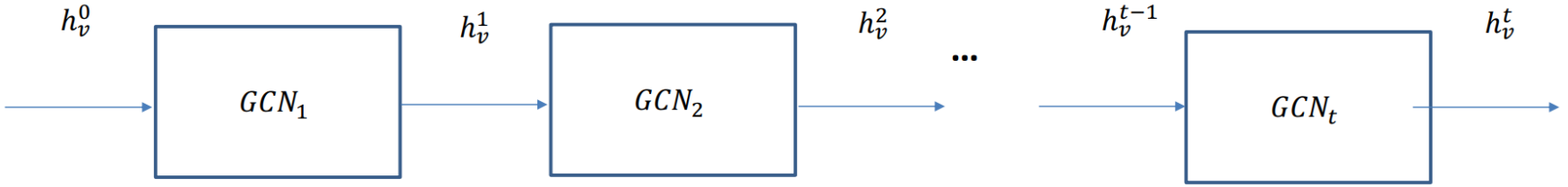
- **Stack** multiple graph convolution layer together
  to **explore** the depth and breadth of a node

  1. **Recurrent-based:**
     apply the **same** GC layer to update hidden representations
  2. **composition-based:**
     apply a **different** GC layer to update hidden representations

Recurrent-based v.s. Composition-based Spatial GCNs



(a) Recurrent-based



(b) Composition-based

Recurrent-based Spatial GCNs

- **Update** a node's latent representation **recursively**
  until a stable fixed point is reached

- This is done by
  - Imposing **constraints** on recurrent functions
  - Employing **gate** recurrent unit architectures
  - Updating node latent representations **asynchronously** and **stochastically**

Stochastic Steady-state Embedding (SSE)

- **Updates** the node latent representations **stochastically** in an **asynchronous** fashion to improve the learning efficiency

- **Recursively** estimates node latent representations
  and **updates** the parameters with **sampled batch** data

- Ensure **convergence** to steady states by defining
  recurrent function of SSE as a weighted average of the historical states and new states

$$\mathbf{h_v}^t = (1-\alpha)\mathbf{h_v}^{t-1} + \alpha \mathbf{W_1}\sigma(\mathbf{W_2}[\mathbf{x_v}, \sum_{u \in N(v)} [\mathbf{h_u}^{t-1}, \mathbf{x_u}]])$$

Learning with Stochastic Fixed Point Iteration

Initialize parameters,$\{\mathbf{h}_v^0\}_{v\in\mathbf{V}}$
for $k = 1$ ***to*** K **do**
  for $t = 1$ ***to*** T **do**
    Sample n nodes from the whole node set $\mathbf{V}$
    Use Equation to update hidden
    representations of sampled n nodes
  end
  for $p = 1$ ***to*** P **do**
    Sample m nodes from the labeled node set $\mathbf{V}$
    Forward model according to Equation
    Back-propagate gradients
  end
end

Composition Based Spatial GCNs

- Update the nodes' representations by **stacking multiple** graph convolution layers

  - **GraphSage:**
    - Introduces the **aggregation function** to define graph convolution.
    - The aggregation function **assembles a node's neighborhood** information.
    - The function must be **invariant to permutations** of node orderings (e.g. mean, sum and max function).

GraphSage

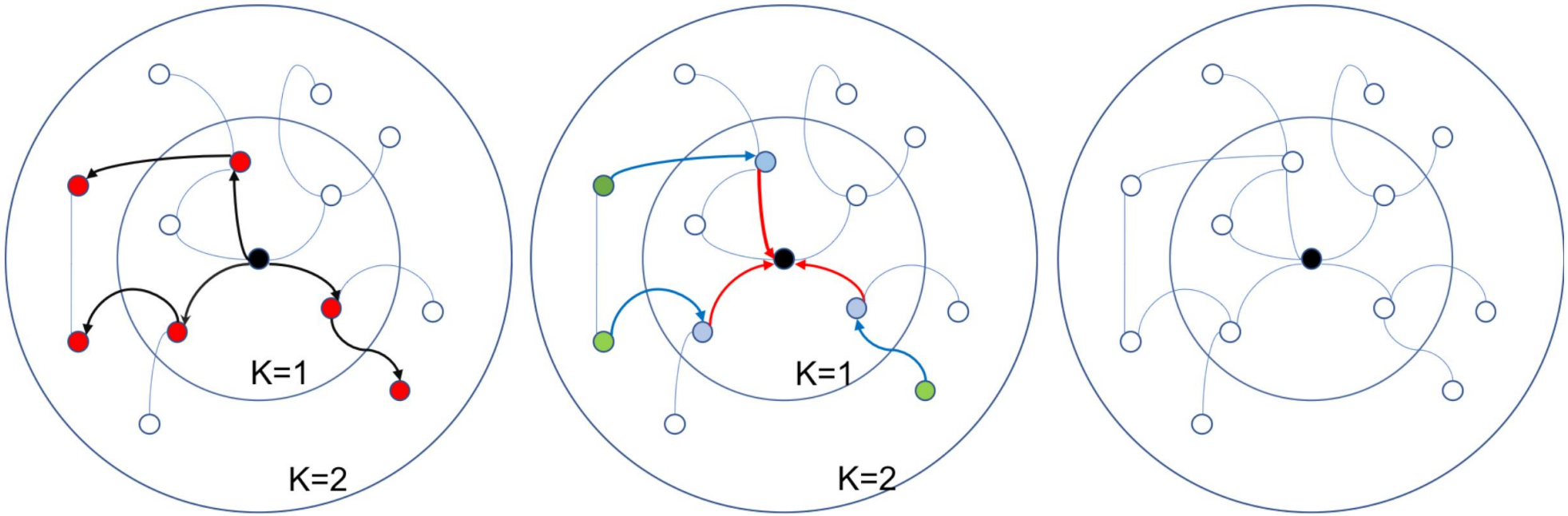
- The graph convolution operation is defined as,

$$\mathbf{h}_v^t = \sigma(\mathbf{W}^t \cdot aggregate_t(\mathbf{h}_v^{t-1}, \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}(v)\})$$

- Proposes a **batch-training** algorithm Instead of updating states over all nodes

- Improves **scalability** for large graphs

Learning Process of GraphSage

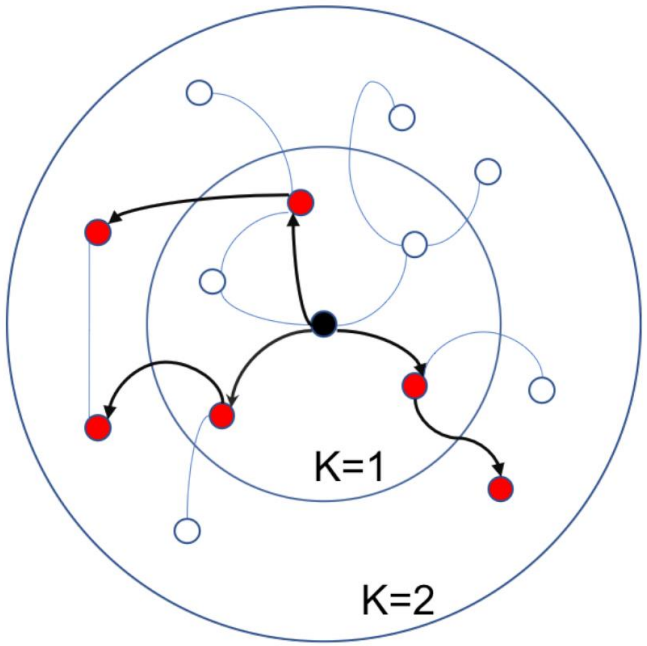


1.Sample neighborhood

2. Aggregate feature information from neighbors

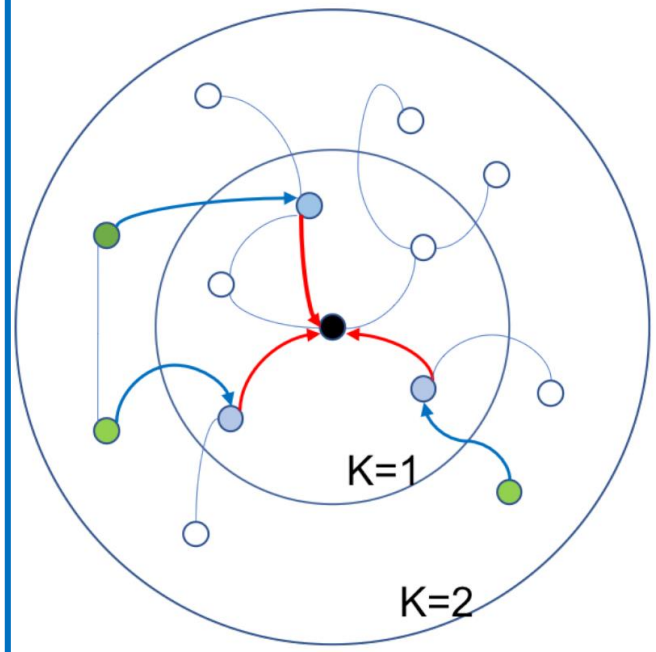
3. Predict node labels

Learning Process of GraphSage

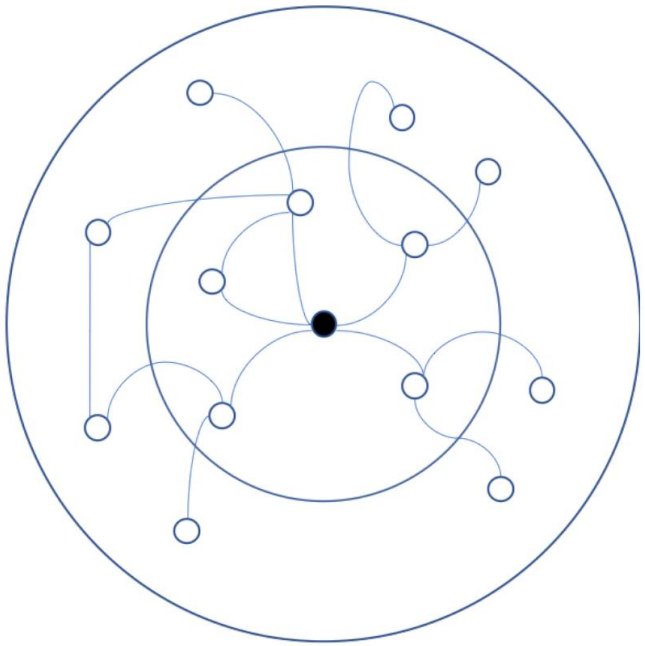


1.Sample neighborhood

samples a node's local k-hop neighborhood with fixed-size

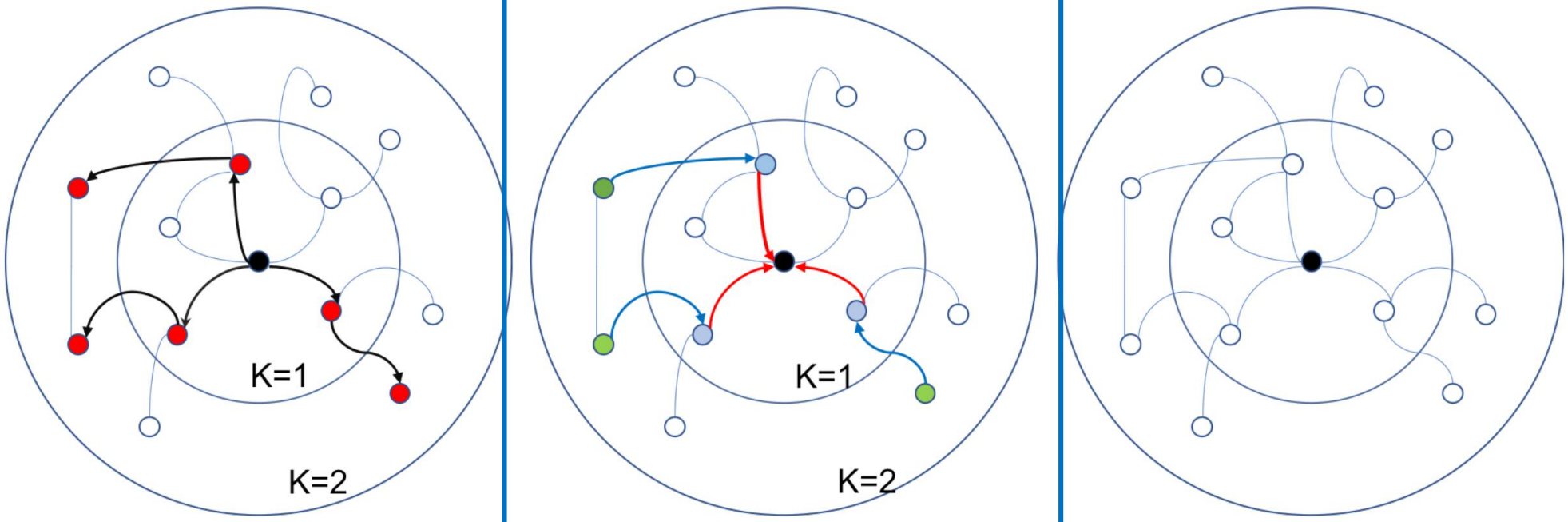


2. Aggregate feature information from neighbors



3. Predict node labels

Learning Process of GraphSage



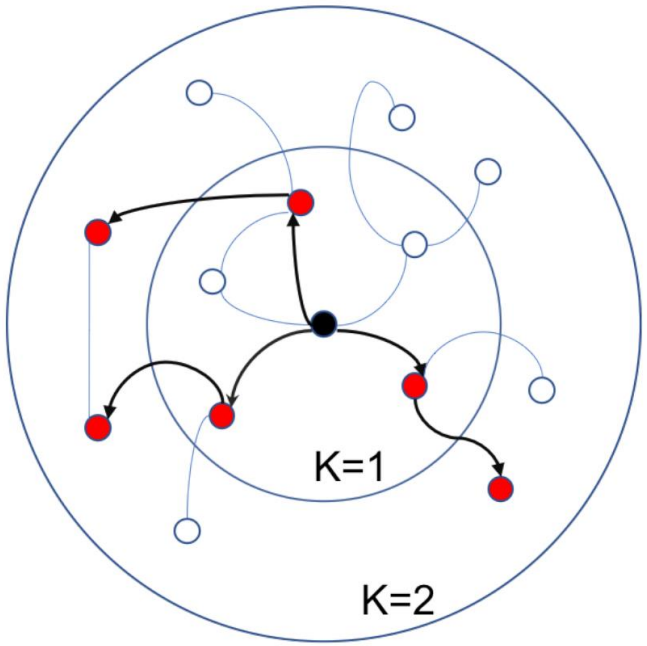
1.Sample neighborhood

2. Aggregate feature information from neighbors

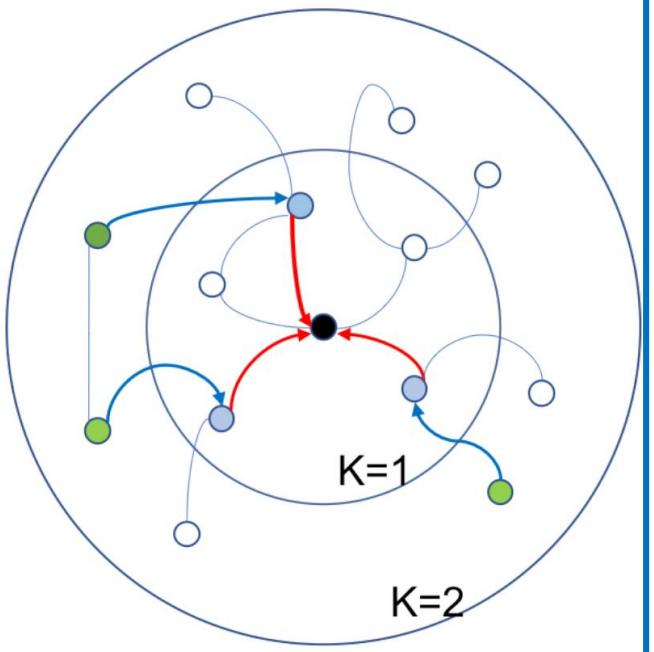
derives the central node's final state by aggregating its neighbors feature information

3. Predict node labels

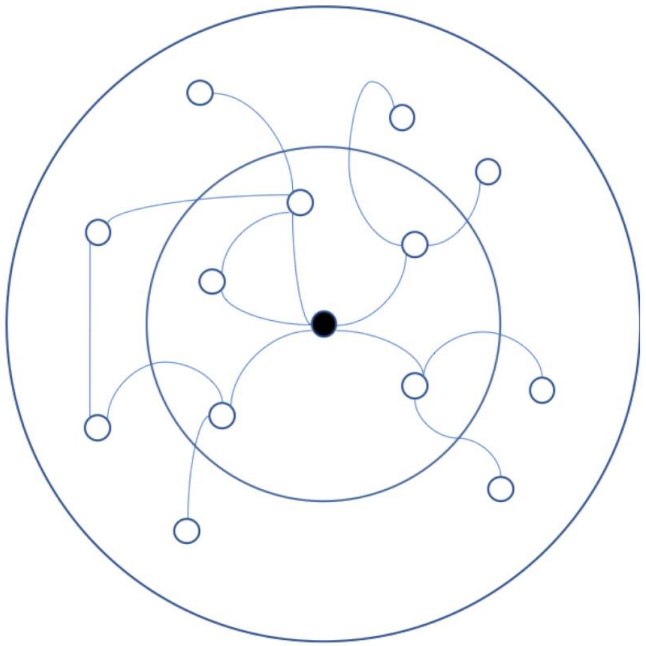
Learning Process of GraphSage



1.Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict node labels

uses the central node's final state to make predictions and backpropagate errors

Learning Process of GraphSage

- Denote s_t as the number of neighbors to be sampled at t^{th} hop, then its time complexity in **one batch** is $O(\prod_{t=1}^{T} s_t)$.
- The computation cost **increases exponentially** with the increase of t.
- This **prevents** from having a deep architecture.
- In practice found that with $\boldsymbol{t = 2}$ already achieves high performance.

Spatial-based GCNs Summary

- Defines graph convolutions via **aggregating** feature information from **neighbors**.
- Different ways of stacking graph convolution layers
    1. **recurrent-based:** try to obtain nodes' **steady** states
    2. **composition-based:** try to incorporate **higher orders** of neighborhood information
- In each layer, both two groups have to update hidden states over all nodes during training.
- **Inefficient to store all the intermediate state**s into memory, so training strategies including sub-graph training
    - SSE for recurrent-based
    - GraphSage for composition based

Comparison Between Spectral and Spatial Models

- Spectral-based models
  - ✓achieved early impressive results in many graph analytics tasks
  - ✓have a theoretical foundation in graph signal processing
  - ✓theoretically design new GCNs by designing new graph signal filters
- Spectral-based models' drawback
  - ➢Efficiency
  - ➢Generality
  - ➢Flexibility

Comparison Between Spectral and Spatial Models: Efficiency

- Spectral:

  Computational **cost increases** dramatically with the **graph size** because

  they either need to perform **eigenvector** computation

  or handle the **whole graph** at the same time

  So, **difficult** to parallel or scale to large graphs.

- Spatial :

  potential to **handle large** graphs as

  aggregating the neighboring nodes.

  The computation can be performed in a **batch** of nodes

  If the number of neighboring nodes increases, **sampling techniques** can be used

Comparison Between Spectral and Spatial Models: Generality

- Spectral:

  assumed a **fixed graph**

  making them **generalize poorly** to new or different graphs

- Spatial:

  convolution **locally** on each node

  weights can be easily **shared across** different locations and structures.

Comparison Between Spectral and Spatial Models: Flexibility

- Spectral-based:

  limited to work on **undirected** graphs

  no clear definition of the **Laplacian** matrix on directed graphs

  transform directed graphs to undirected graphs.

- Spatial:

  more flexible to deal with **multi-source inputs** such as edge features and edge directions

  inputs can be **incorporated** into the aggregation function

Interested?

mamjad2@uic.edu