Mean techniques for ML & Representation learning for graph data:

▪ Traditional methods: Graphlets, Graph Kernels

▪ Methods for node embeddings: DeepWalk, Node2Vec

▪ Graph Neural Networks: GCN, GraphSAGE, GAT, Theory of GNNs

▪ Knowledge graphs and reasoning: TransE, BetaE

▪ Deep generative models for graphs: GraphRNN

**Choice of a graph representation**

Undirected graph

Directed graph

Sparse graph

Dense graph

Heterogeneous graph

𝑮 =(𝑽,𝑬,𝑹,𝑻)

▪ Nodes with node types 𝑣𝑖 ∈ 𝑉

▪ Edges with relation types 𝑣𝑖 , 𝑟, 𝑣𝑗 ∈ 𝐸

▪ Node type 𝑇 𝑣𝑖

▪ Relation type 𝑟 ∈ 𝑅

Bipartite graph

Unweighted

Weighted

Self-edges(self-loops)

Multigraph (muti-links between two nodes)

Directed Cyclic Graph(DCG)

COO, CSR, adj\_COO, adj\_CSR

稀疏图：数据结构中对于稀疏图的定义为：有很少条边或弧（边的条数远小于的图称为稀疏图，反之边的条数 接近，称为稠密图。采用直观的办法来存储图往往会造成极大的空间浪费，因此需要采取其他方式压缩存储空间。

一、COO Coordinate Format

对于稠密图，我们往往以矩阵的方式存储结点的连接关系。如图1a所示，对于矩阵 matrix，matrix[i][j] = x 表示结点 i 与结点 j 之间的边的长度为 x。我们可以看到在图1a的矩阵 matrix 中，除了少数结点间有边相连，大多数的存储空间都浪费了。

对于稀疏图，最直观的压缩存储方式是只存储矩阵 matrix 中的非零元素以及这些元素的位置，也就是以三元组的方式存储 (i, j, x)。(i, j, x) 同样表示结点 i 与结点 j 之间的边的长度为 x，如图1b所示。

使所有三元组的横坐标单独组成 row 数组，纵坐标单独组成 column 数组，数值单独组成 data 数组就形成了稀疏矩阵的 COO表示，如图1c所示。

A diagram of numbers and arrows

Description automatically generated

图1 COO表示

二、CSR (Compressed Sparse Row)

对于结点数量比较多，并且远大于边数量的稀疏矩阵，上一小节中介绍的COO表示已经能够节省很多存储空间了，那我们还能不能进一步节省更多的空间呢？

当然可以！我们观察上一小节中得到的 COO表示（如图2b所示），row 数组中各三元组的横坐标 按序排列，因此 相同的横坐标会连在一起，这何尝不是一种数据重复？

我们保持 column 数组和 data 数组 不变。将 row 数组改为 row offsets：不再记录每一个元素的横坐标，只记录每一行的第一个元素在 data 数组中的下标，最后再额外记录总的元素个数。如图2c所示。

新的数组 row offsets 就像书签一样，其中的第 i 个元素 ro[i]就代表了 data[ro[i]] 与 data[ro[i+1]] 之间的元素的横坐标为 i（包括data[ro[i]] 但不包括 data[ro[i+1]]）。

如图2中，我们分别用蓝、黄、绿、橙代表不同的四行的元素。 第 0 行的第一个元素为6，data 数组中其下标为 0，故 ro[0]=0。 第 1 行的第一个元素为3，data 数组中其下标为 2，故 ro[1]=2。 第 2 行的第一个元素为5，data 数组中其下标为 4，故 ro[2]=4。 第 3 行的第一个元素为7，data 数组中其下标为 5，故 ro[3]=5。 data 数组中共有 7 个元素，故 ro[4]=7。

A close-up of a chart

Description automatically generated

​图2 CSR表示

三、adj\_coo

第一小节中我们讨论了普通稀疏图的矩阵转化为COO表示。但如果邻接矩阵中只记录了两个结点是否相连，并没有记录边的信息（如图3a），我们便不需要记录 data 数据，如此可以进一步压缩存储空间。

A yellow arrow pointing to a number

Description automatically generated

​图3 adj\_coo表示

四、adj\_csr

与adj\_coo情况类似，如果邻接矩阵中只记录了两个结点是否相连，并没有记录边的信息（如图4a），我们便不需要记录 data 数据，如此可以进一步压缩存储空间。

A yellow arrow pointing to a number

Description automatically generated

​图4 adj\_csr表示

Feature Engineering for ML in Graphs

1. Node-level Features

Node degree:

1. The degree 𝑘𝑣 of node 𝑣 is the number of edges (neighboring nodes) the node has.
2. Node degree counts the neighboring nodes without capturing their importance.

Node centrality

1. Node centrality C𝑣 takes the node importance in a graph into account.
2. Different ways of node model centrality (Engienvector centrality, Betweenness centrality, Closeness centrality…)
3. Eigenvector centrality: A node 𝑣 is important if surrounded by important neighboring nodes 𝑢 ∈ 𝑁(𝑣).

A blue arrows pointing to a number

Description automatically generated with medium confidence

*Models centrality in a recursive manner*

The eigenvector C𝑚𝑎𝑥 corresponding to 𝜆𝑚𝑎𝑥 is used for centrality.

1. Betweenness centrality: A node is important if it lies on many shortest paths between other nodes.

A black text on a white background

Description automatically generated

1. Closeness centrality: A node is important if it has small, shortest path lengths to all other nodes.

A black text with black text

Description automatically generated

Clustering coefficient: Measures how connected 𝑣′𝑠 neighboring nodes are

A black and green text

Description automatically generated

Graphlets: Graphlets are small subgraphs that describe the structure of node 𝑢’s network neighborhood.

A diagram of a molecule

Description automatically generated with medium confidence

*Graph Isomorphism: Two graphs which contain the same number of nodes connected in the same way are said to be isomorphic.*

1. Link-level features

Distance-based feature: Shortest-path distance between two nodes

Local neighborhood overlap: Captures No. of neighboring nodes shared between two nodes 𝒗𝟏 and 𝒗2

A math equations and formulas

Description automatically generated with medium confidence

Global Neighborhood Overlap: Katz index: count the number of walks of all lengths between a given pair of nodes.

Using adjacency matrix powers

A math equations and numbers

Description automatically generated

1. Graph-level features

Graph Kernels: Measure similarity between two graphs

E.P. Graphlet Kernel; Weisfeiler-Lehman Kernel…

Key idea1: Bag-of-Words (BoW) for a graph. Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use Bag-of-\* representation of graph.

1. Graphlet Kernel: Count the number of different graphlets in a graph. Bag-of-graphlets

define the graphlet count vector



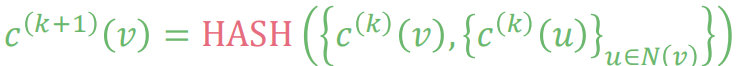
A diagram of a mathematical equation

Description automatically generated with medium confidence

1. WL Kernel: => color refinement Bag-of-colors

Apply 𝐾-step color refinement algorithm to enrich node colors.

▪ Assign an initial color 𝑐0 𝑣 to each node 𝑣.

▪ Iteratively refine node colors by 

▪After 𝐾 steps of color refinement, 𝑐𝐾 𝑣 summarizes the structure of 𝐾-hop neighborhood

The WL kernel value is computed by the inner product of the color count vectors:

A black and white image of a number

Description automatically generated

**Node Embedding**

Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph.

a. Encoder maps from nodes to embeddings

b. Define a node similarity function (i.e., a measure of similarity in the original network)

c. Decoder 𝐃𝐄𝐂 maps from embeddings to the similarity score (i.e. dot product)

d. Optimize the parameters of the encoder so that:A group of symbols on a white background

Description automatically generated

Note:

▪ This is unsupervised/self-supervised way of learning node embeddings (not use labels or features).

▪ These embeddings are task independent (They are not trained for a specific task but can be used for any task)

1. Random Walk Approaches

 Vector 𝐳𝑢:

▪ The embedding of node 𝑢 (what we aim to find).

 Probability 𝑃 (𝑣| 𝐳𝑢) :

▪ The (predicted) probability of visiting node 𝑣 on random walks starting from node 𝑢.

Def: Given a graph and a starting point, we select a neighbor of it at random, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc

1. Estimate probability of visiting node 𝒗 on a random walk starting from node 𝒖 using some random walk strategy R
2. Optimize embeddings to encode these random walk statistics: Similarity in embedding space (Here: dot product=cos(𝜃)) encodes random walk “similarity”
3. Use SDG to find minimums

A diagram of mathematical equations

Description automatically generatedPlus Nagative SampleA math equations and formulas

Description automatically generated with medium confidence

1. Node2vec

Goal: Embed nodes with similar network neighborhoods close in the feature space.

Idea: use flexible, biased random walks that can trade off between local and global views of the network

Local-> by BFS

Global->by DFS

Two parameters:

Return parameter 𝒑:

▪ Return back to the previous node

In-out parameter 𝒒:

▪ Moving outwards (DFS) vs. inwards (BFS)

▪ Intuitively, 𝑞 is the “ratio” of BFS vs. DFS

A diagram of a diagram

Description automatically generated

*Note:*

*No one method wins in all cases*

*node2vec performs better on node classification while alternative methods perform better on link prediction (Goyal and Ferrara, 2017 survey)*

1. DeepWalk

Objective: maximize 𝐳𝑣Τ 𝐳𝑢 for node pairs (𝑢, 𝑣) that are similar

Embedding -> Matrix Factorization

A diagram of a embedding

Description automatically generated

Nodes 𝑢, 𝑣 are similar if they are connected by an edge

This means: 𝐳𝑣Τ𝐳𝑢 = 𝐴𝑢,𝑣, which is the (𝑢, 𝑣) entry of the graph adjacency matrix A

Conclusion: Inner product decoder with node similarity defined by edge connectivity is equivalent to matrix factorization of 𝐴.

A diagram of mathematical equations

Description automatically generated

1. Limitations

Cannot obtain embeddings for nodes not in the training set

Cannot capture structural similarity

Cannot utilize node, edge and graph features

**Page Rank**

A math equation with black text

Description automatically generated

A white background with black and green text

Description automatically generated

A close-up of a math problem

Description automatically generated

Restart & Personized PageRank

Personized: teleports to a specific set of nodes; Nodes can have different probabilities of the surfer landing there

Restart: Topic-specific PageRank where teleport is always to the same node

**A Layer in Graph Neural Networks**

GNN layer = Message + Aggregation

• GCN, GraphSAGE, GAT, …

Idea of a GNN Layer: Compress a set of vectors into a single vector.

A diagram of a computer

Description automatically generated

1. Message computation

Intuition: Each node will create a message, which will be sent to other nodes later

A black text on a white background

Description automatically generated

1. Message aggregation

Intuition: Each node will aggregate the messages from node 𝑣’s neighbors

A black and white text

Description automatically generated

A close-up of a mathematical equation

Description automatically generated

1. Add nonlinearity to computation and aggregation process

**Graph Convolutional Network (GCN)**

Key idea: Generate node embeddings based on local network neighborhoods.

Intuition: Nodes aggregate information from their neighbors using neural networks.

Network neighborhood defines a computation graph.

Model can be of arbitrary depth:

▪ Nodes have embeddings at each layer

▪ Layer-0 embedding of node 𝑣 is its input feature, 𝑥𝑣

▪ Layer-𝑘 embedding gets information from nodes that are 𝑘 hops away

Deep Encoder:

A diagram of a network

Description automatically generated

A diagram of a complex equation

Description automatically generated

*Note:Add self-loop to make sure the updating includes self-feature*

Training process:

1. Supervised setting

A white background with black text

Description automatically generated

A diagram of a machine

Description automatically generated

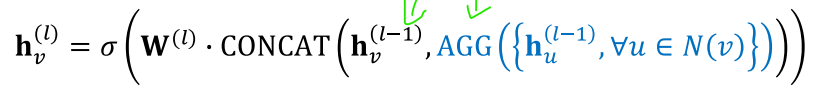
1. Unsupervised setting

A white background with text and symbols

Description automatically generated with medium confidence

*Note: hidden layer (final output layer) represents the node embedding.*

**GraphSAGE**



**Message** is computed within the 

**Two-stage aggregation**:

Stage 1: Aggregate from node neighbors

A black and white text

Description automatically generated

Stage 2: Further aggregate over the node itself



Stege 1 approaches:

* Mean: Take a weighted average of neighbors

A yellow rectangular object with black letters and numbers

Description automatically generated

* Pool: Transform neighbor vectors and apply symmetric vector function Mean(⋅) or Max(⋅)

A yellow rectangle with black letters and numbers

Description automatically generated

* LSTM: Apply LSTM to reshuffled of neighbors

A black text on a white background

Description automatically generated

**L2 Normalization**

Optional: Apply ℓ2 normalization to  at every layer

**A close up of a text

Description automatically generated**

**Graph Attention(GAT)**

**A yellow square with black letters and a square with green text

Description automatically generated**

The attention auv focuses on the important parts of the input data and fades out the rest.

▪ Idea: the NN should devote more computing power on that small but important part of the data. ▪ Which part of the data is more important depends on the context and is learned through training.

**Mechanism**

1. Let a compute attention coefficients euv across pairs of nodes u, v based on their messages:

**A black and white image of a mathematical equation

Description automatically generated**

euv indicates the importance of u′ s message to node v

1. Normalize euv into the final attention weight auv

**A black text with a line

Description automatically generated with medium confidence**

1. Weighted sum based on the final attention weight auv

A black and pink text

Description automatically generated with medium confidence

The approach is agnostic to the choice of a:

e.g., use a simple single-layer neural network

▪ a have trainable parameters (weights in the Linear layer)

A group of black letters

Description automatically generated

**Multi-head attention:** Stabilizes the learning process of attention mechanism

▪ Create multiple attention scores (each replica with a different set of parameters):

A group of black and red letters

Description automatically generated with medium confidence

▪ Outputs are aggregated:

A close up of a number

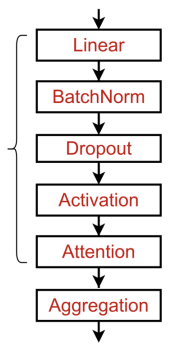
Description automatically generated

**Train GNN Layers in Practice**

**1.Training Process**

In practice, these classic GNN layers are a great starting point

Many modern deep learning modules can be incorporated into a GNN layer:



▪ Batch Normalization: Stabilize neural network training

▪ Dropout: Prevent overfitting

▪Attention/Gating: Control the importance of a message

▪ More: Any other useful deep learning modules

**2.Stacking layers** into a model:

▪ The standard way: Stack GNN layers sequentially

▪ Input: Initial raw node feature 

▪ Output: Node embeddings A close up of letters

Description automatically generated after L GNN layers

▪ The Issue of stacking many GNN layers

**The over-smoothing problem**: all the node embeddings converge to the same value

Solution 1: Increase the expressive power within each GNN layer

Make aggregation / transformation become a deep neural network!

Solution 2: Add layers that do not pass messages

E.g., we can add MLP layers (applied to each node) before and after GNN layers, as pre-process layers and post-process layers

Solution 3: Add skip connections in GNNs

A diagram of a graph

Description automatically generated

e.g., A group of math equations

Description automatically generated with medium confidence

**3.graph manipulation before training**

Graph Feature manipulation

▪ The input graph lacks features → feature augmentation

Add features manually, e.g., Assign unique IDs to nodes, Clustering coefficient, PageRank, Centrality…

Graph Structure manipulation

▪ The graph is too sparse → Add virtual nodes/edges

a. Common approach: Connect 2-hop neighbors via virtual edges use A + A2

b. The virtual node will connect to all the nodes in the graph, After adding the virtual node, all the nodes will have a distance of 2

▪ The graph is too dense → Sample neighbors when doing message passing

▪ The graph is too large → Sample subgraphs to compute embeddings, Scaling up GNN

**Aggregation Choice**

m**ean aggregation** captures the distribution (or proportions) of elements,

**max aggregation** proves to be advantageous to identify representative elements,

and **sum aggregation** enables the learning of structural graph properties ([Xu et al. (2018)](https://arxiv.org/abs/1810.00826)).

Recent works also show that using **multiple aggregations** ([Corso et al. (2020)](https://arxiv.org/abs/2004.05718), [Tailor et al. (2021)](https://arxiv.org/abs/2104.01481)) and **learnable aggregations** ([Li et al. (2020)](https://arxiv.org/abs/2006.07739)) can potentially provide substantial improvements.

Another line of research studies optimization-based and implicitly-defined aggregations ([Bartunov et al. (2022)](https://arxiv.org/abs/2202.12795)).

Furthermore, an interesting discussion concerns the trade-off between representational power (usually gained through learnable functions implemented as neural networks) and the formal property of permutation invariance ([Buterez et al. (2022)](https://arxiv.org/abs/2211.04952)).

**KGR Model**

Torch\_geometric.nn.KGEModel => specifically for Knowledge Graph

focus on learning embeddings for knowledge graphs, where the goal is to represent entities and relationships in a low-dimensional, continuous vector space.

**Dense convolution layers and pooling layers**

Using dense layers in the context of neural networks, including those in graph neural networks like **torch\_geometric.nn**, depends on several factors related to your data, the specific problem you're addressing, and computational considerations. Here are some scenarios where using dense layers might be appropriate:

1. **Small to Medium-sized Graphs**: Dense layers are suitable for graphs that are not too large, as they require a dense representation of the adjacency matrix, which can become very memory-intensive for large graphs.
2. **Fully-connected or Dense Graph Structures**: If your graph is fully-connected or nearly fully-connected, a dense representation might be more natural and efficient compared to a sparse one.
3. **Graphs with Rich Feature Interactions**: In cases where the interactions between all pairs of nodes (or a significant number of them) are important for your task, dense layers can effectively capture these interactions.
4. **High Memory Resources**: If you have access to high memory resources, you can leverage dense layers even for relatively larger graphs, as the memory requirement for dense representations can be significant.
5. **Simple Implementation and Experimentation**: Dense layers can sometimes be simpler to implement and experiment with, especially for those who are more familiar with standard neural network architectures like CNNs.
6. **Specific Graph-based Tasks**: Certain tasks, especially those involving small-scale, fully-featured graphs (like molecular graphs or small social networks), might benefit more from dense layers due to the nature of interactions within the graph.

However, it's important to consider the trade-offs:

* **Memory Efficiency**: Sparse representations are generally more memory-efficient for large and sparse graphs.
* **Scalability**: Dense layers might not scale well to very large graphs due to the quadratic increase in memory requirements with the number of nodes.
* **Computational Efficiency**: Operations on dense matrices can be computationally expensive, especially as the size of the graph grows.

[**Model Summary**](https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#id56)

import torch

from torch\_geometric.nn import GCN, summary

model = GCN(128, 64, num\_layers=2, out\_channels=32)

x = torch.randn(100, 128)

edge\_index = torch.randint(100, size=(2, 20))

print(summary(model, x, edge\_index))

**summary(model*:***[Module](https://pytorch.org/docs/master/generated/torch.nn.Module.html#torch.nn.Module)**, *\**args, max\_depth*:***[int](https://docs.python.org/3/library/functions.html#int)***=*3, leaf\_module*:***[Optional](https://docs.python.org/3/library/typing.html#typing.Optional)[[Union](https://docs.python.org/3/library/typing.html#typing.Union)[[Module](https://pytorch.org/docs/master/generated/torch.nn.Module.html#torch.nn.Module), [List](https://docs.python.org/3/library/typing.html#typing.List)[[Module](https://pytorch.org/docs/master/generated/torch.nn.Module.html#torch.nn.Module)]]]***=*'MessagePassing', *\*\**kwargs)**

**PARAMETERS:**

* **model** (*[torch.nn.Module](https://pytorch.org/docs/master/generated/torch.nn.Module.html" \l "torch.nn.Module" \o "(in PyTorch vmaster (2.1.0a0+gitbe0b12e )))*) – The model to summarize.
* **\*args** – The arguments of the **model**.
* **max\_depth** ([*int*](https://docs.python.org/3/library/functions.html#int)*, optional*) – The depth of nested layers to display. Any layers deeper than this depth will not be displayed in the summary. (default: **3**)
* **leaf\_module** (*[torch.nn.Module](https://pytorch.org/docs/master/generated/torch.nn.Module.html" \l "torch.nn.Module" \o "(in PyTorch vmaster (2.1.0a0+gitbe0b12e ))) or [[torch.nn.Module](https://pytorch.org/docs/master/generated/torch.nn.Module.html" \l "torch.nn.Module" \o "(in PyTorch vmaster (2.1.0a0+gitbe0b12e )))], optional*) – The modules to be treated as leaf modules, whose submodules are excluded from the summary. (default: **[MessagePassing](https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.MessagePassing.html" \l "torch_geometric.nn.conv.MessagePassing" \o "torch_geometric.nn.conv.MessagePassing)**)
* **\*\*kwargs** – Additional arguments of the **model**.