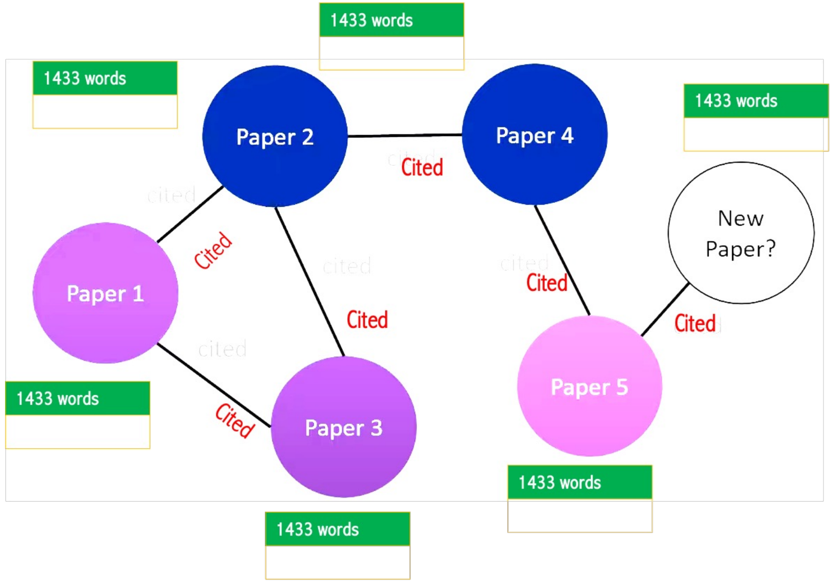
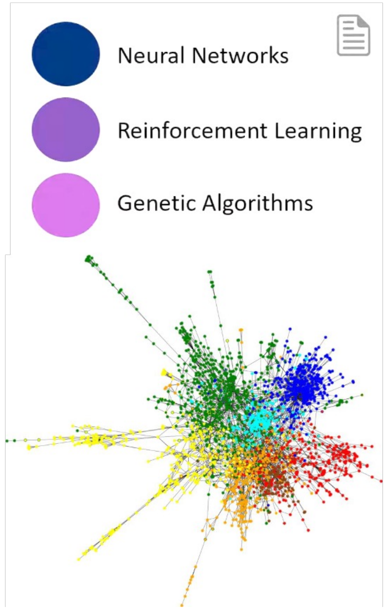
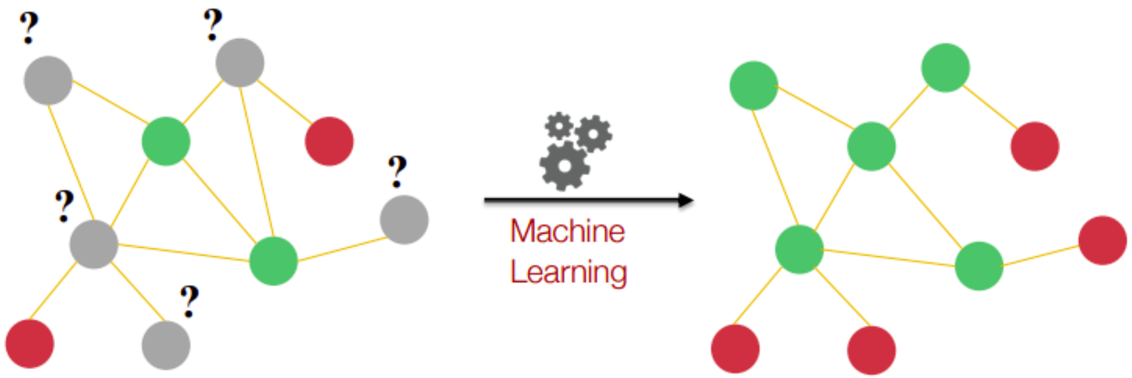
AI VIETNAM
All-in-One Course

Introduction to Graph Neural Network
(Node Classification: Cora Citation Dataset)







Vinh Dinh Nguyen
PhD in Computer Science

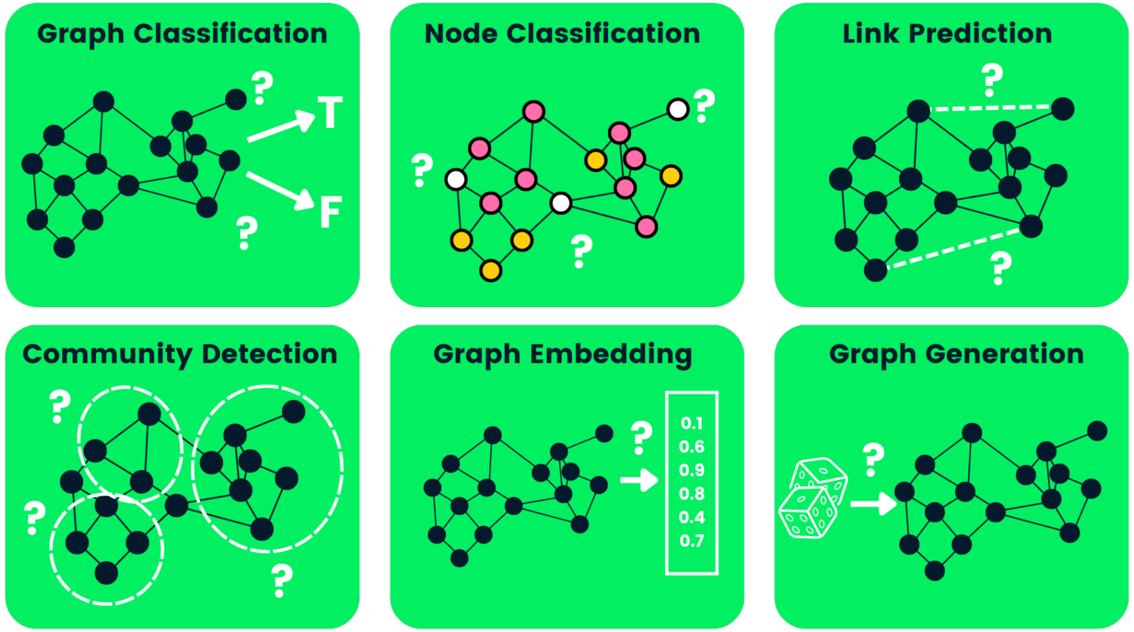
Outline

- Objective
- Introduction to Graph Data
- Graph Data with Neural Network
- Node Classification Problem: Cora Citation Dataset
- Summary

Outline

- Objective
- Introduction to Graph Data
- Graph Data with Neural Network
- Node Classification Problem: Cora Citation Dataset
- Summary

Objective

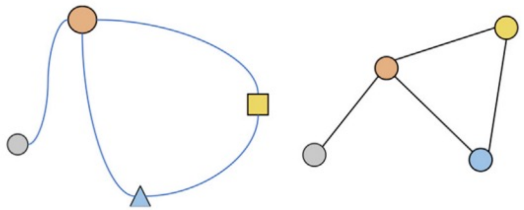


1. What is Graph Data Around Us
2. Understand Graph Neural Network
3. Understand Graph Convolutional Neural Network
4. Node Classification with Cora Citation Dataset

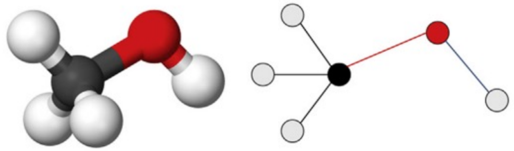
Outline

- Objective
- Introduction to Graph Data
- Graph Data with Neural Network
- Node Classification Problem: Cora Citation Dataset
- Summary

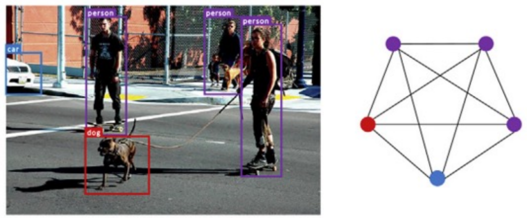
Applications of GNNs



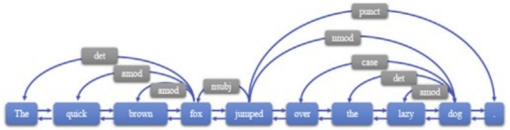
(a) Physics



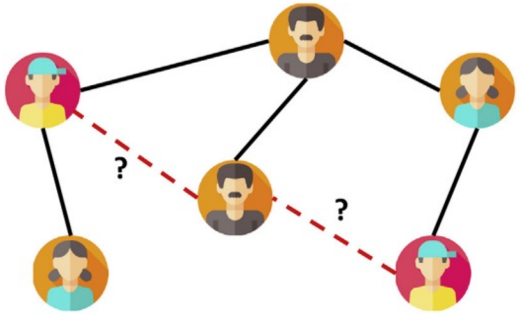
(b) Molecule



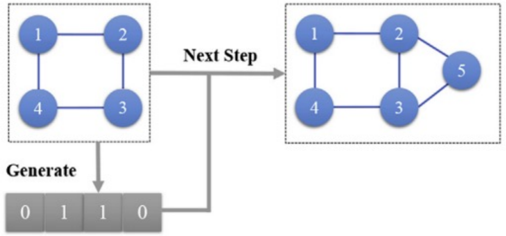
(c) Image



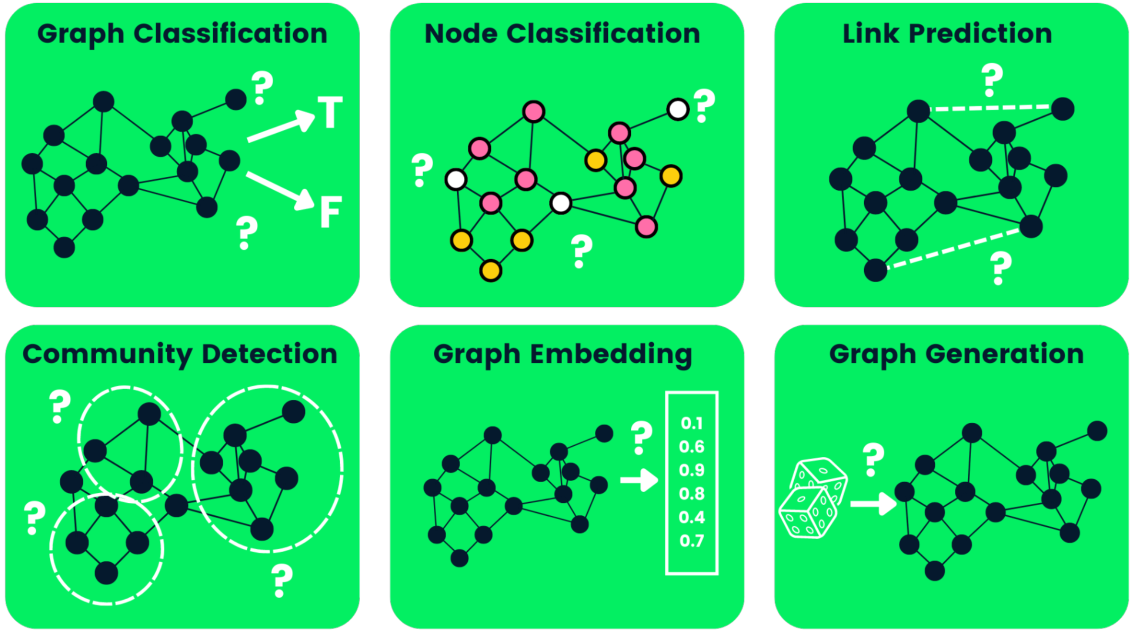
(d) Text



(e) Social Network

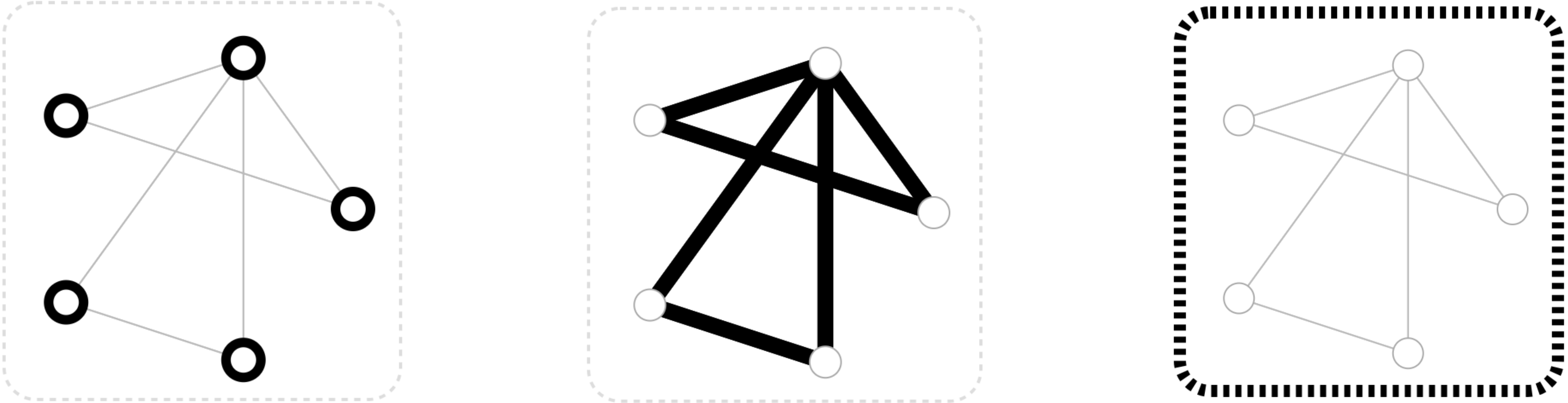


(f) Generation



Graph Definition

A graph represents the relations (edges) between a collection of entities (nodes).

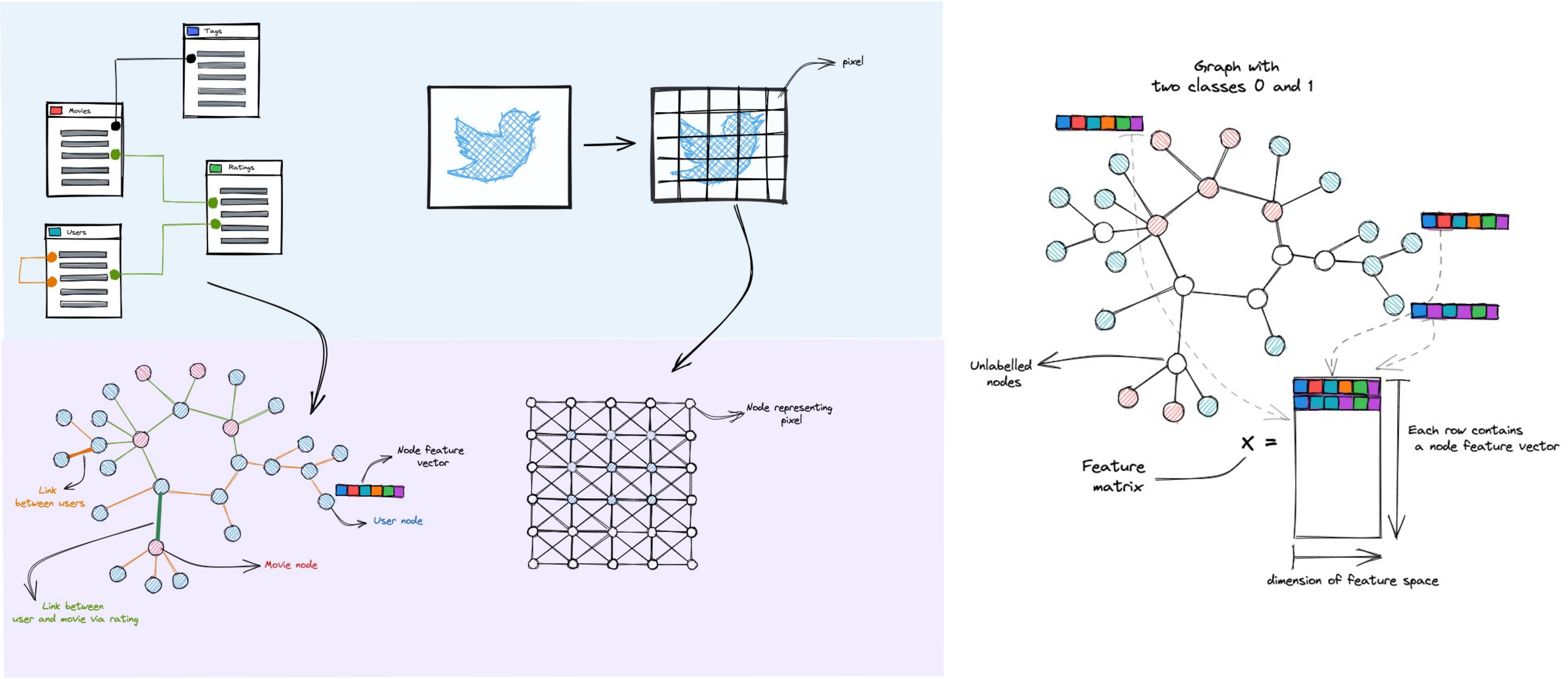


V Vertex (or node) attributes
e.g., node identity, number of neighbors

E Edge (or link) attributes and directions
e.g., edge identity, edge weight

U Global (or master node) attributes
e.g., number of nodes, longest path

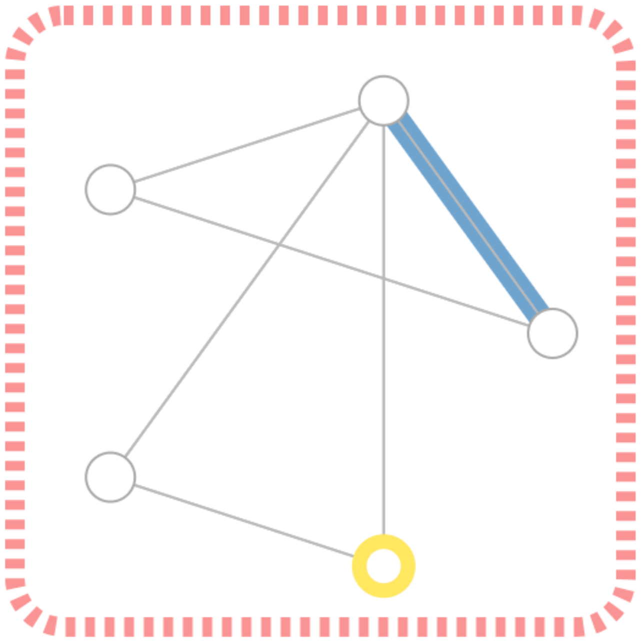
Graphs are everywhere



https://towardsdatascience.com/over-smoothing-issue-in-graph-neural-network-bddc8fbc2472

Graph Definition

To further describe each node, edge or the entire graph, we can store information in each of these pieces of the graph

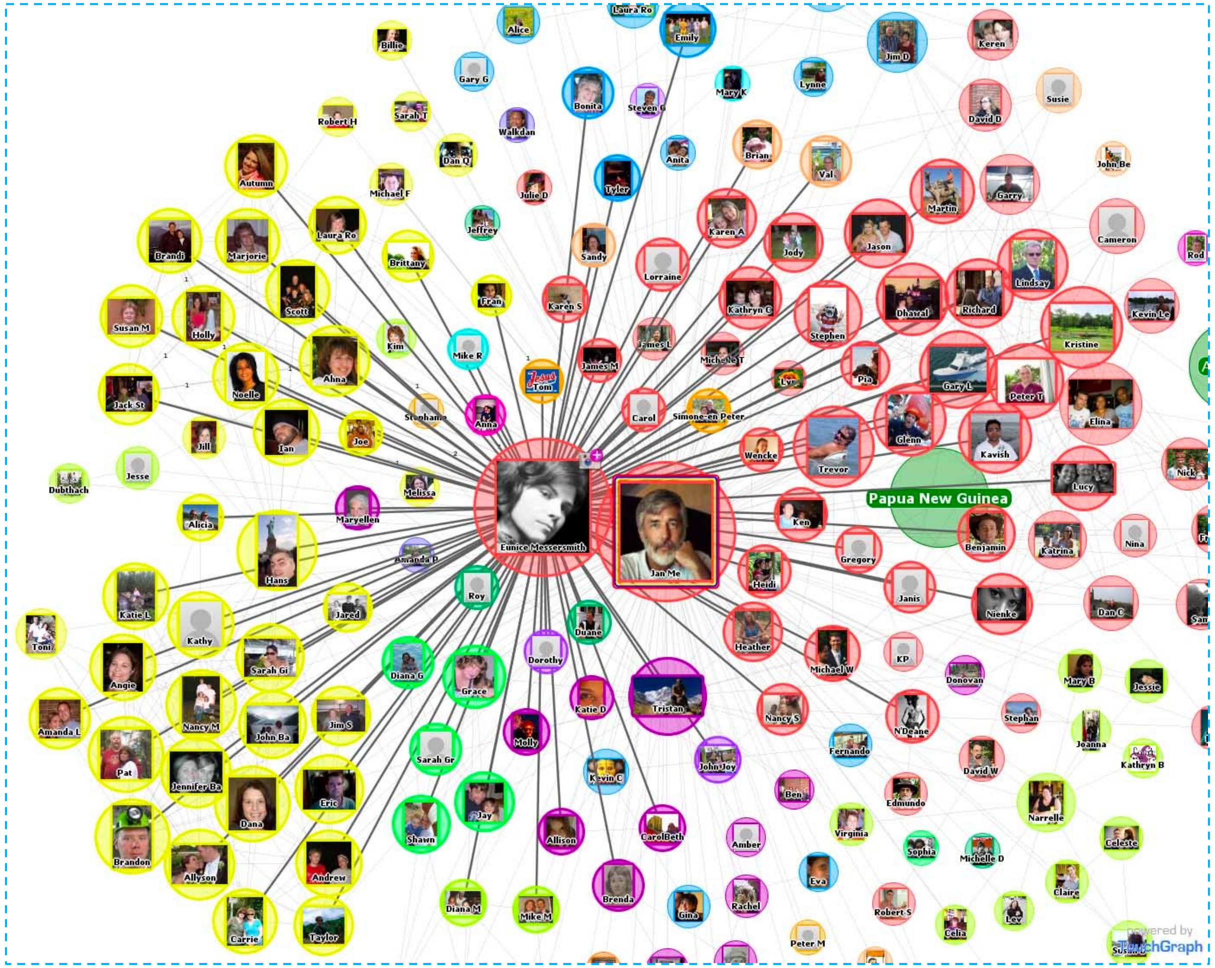


Vertex (or node) embedding

Edge (or link) attributes and embedding

Global (or master node) embedding

Graphs and where to find them



Social networks

Two types of data that you might not think could be modeled as graphs:

Image

Graphs are all around us

Text

Image as Graphs

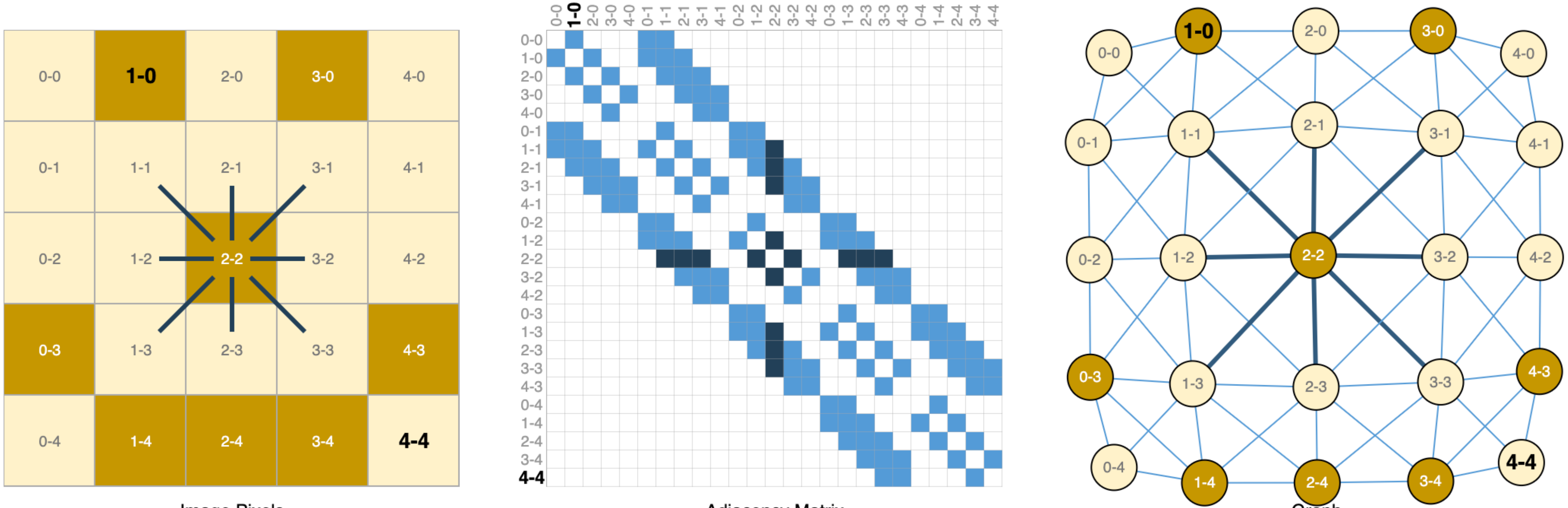


Image Pixels

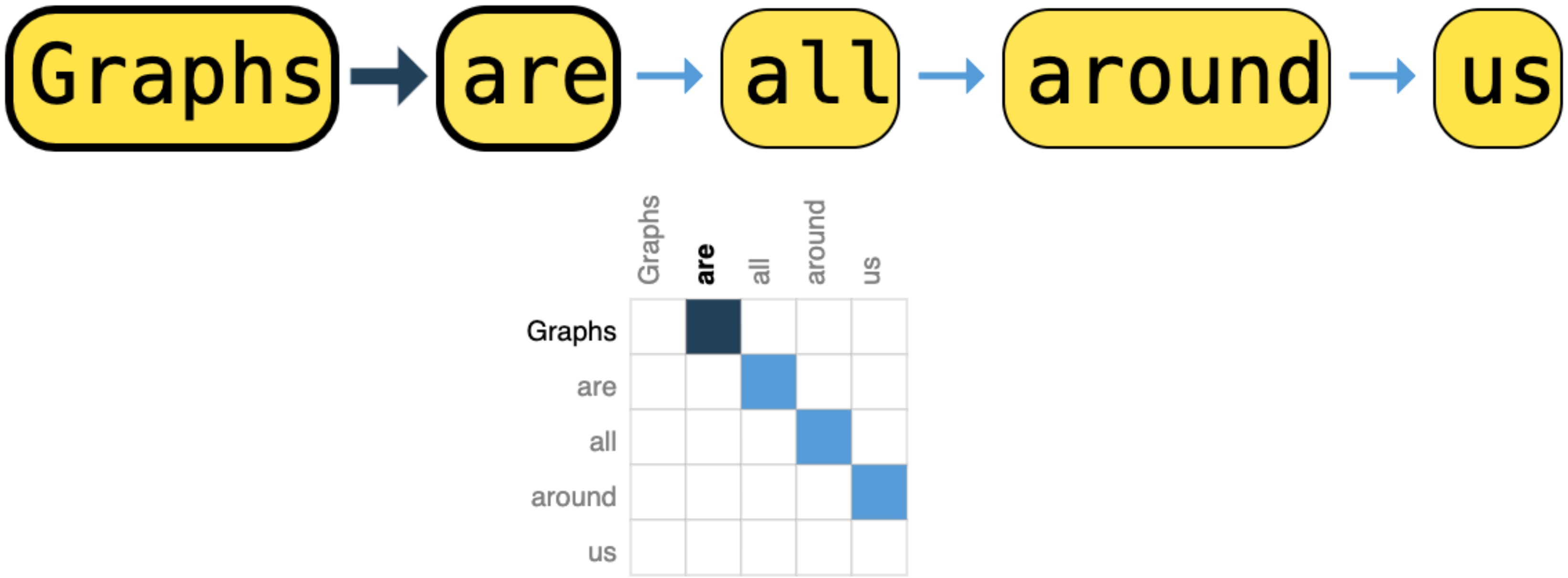
Adjacency Matrix

Graph

Another way to think of images is as graphs with regular structure, where each pixel represents a node and is connected via an edge to adjacent pixels. Each non-border pixel has exactly 8 neighbors, and the information stored at each node is a 3-dimensional vector representing the RGB value of the pixel.

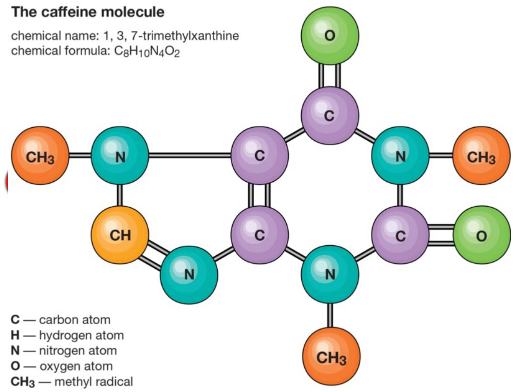
Source: https://distill.pub/2021/gnn-intro/

Text as Graphs

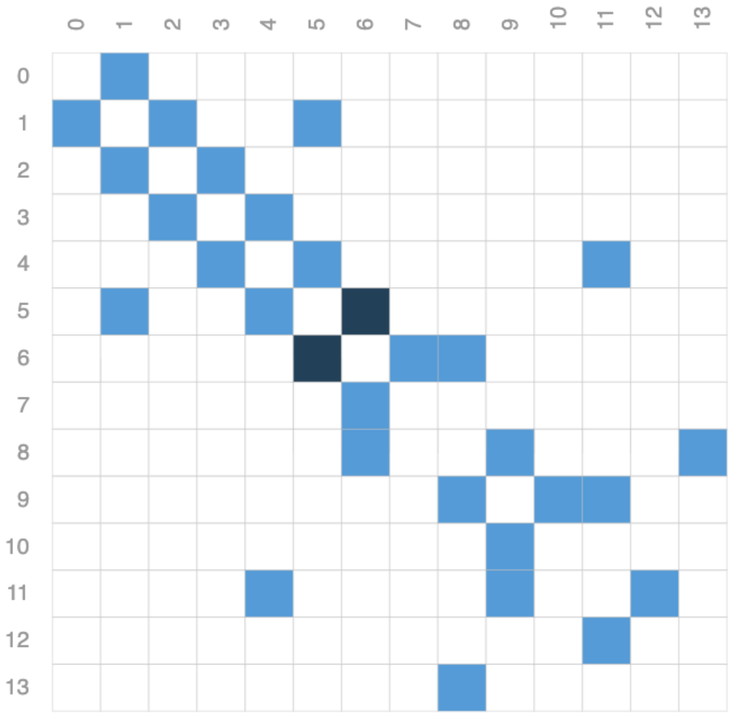


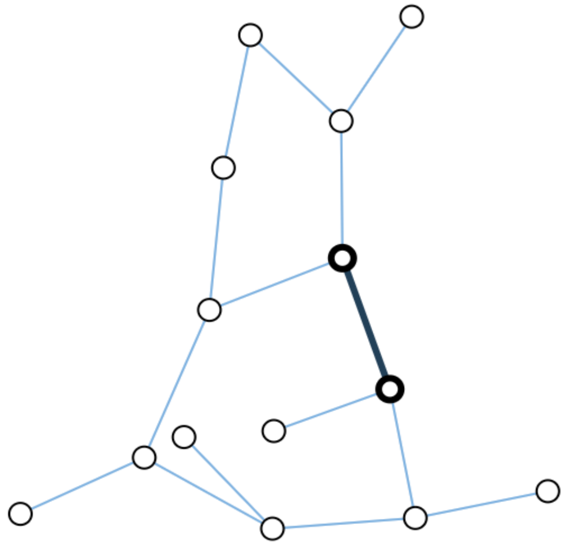
We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.

Other Graph Data



Cafeine molecule



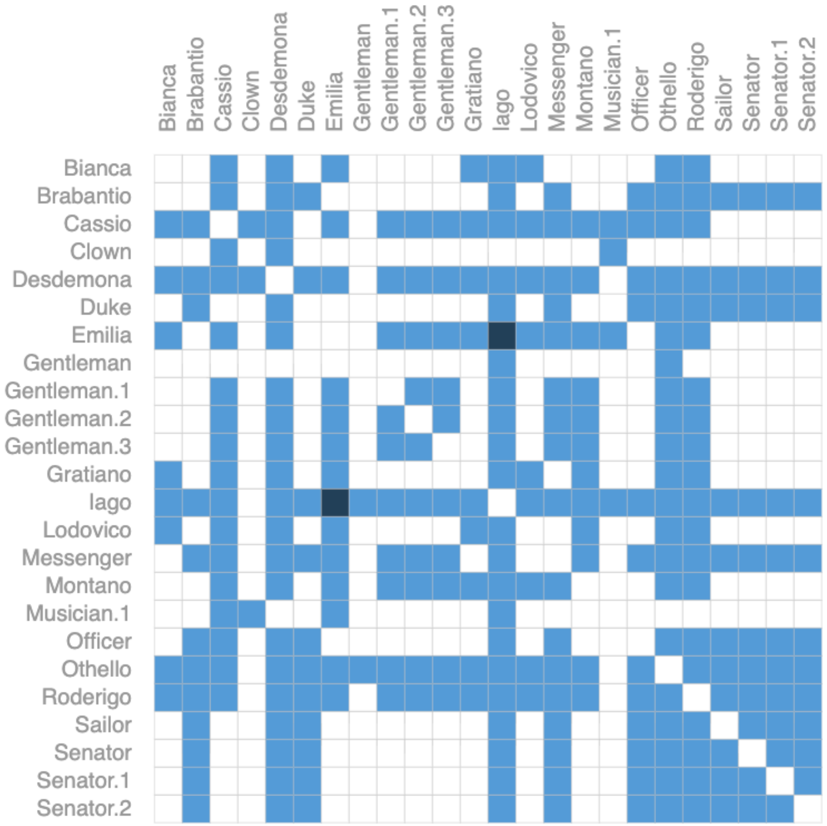


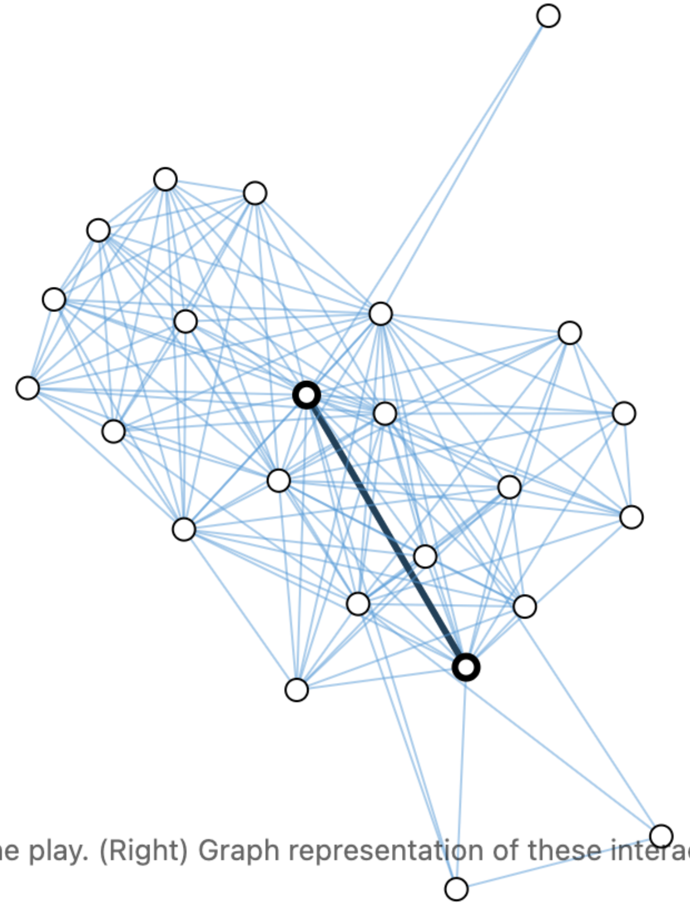
Grap model

Adjacency matrix

Molecules as graphs

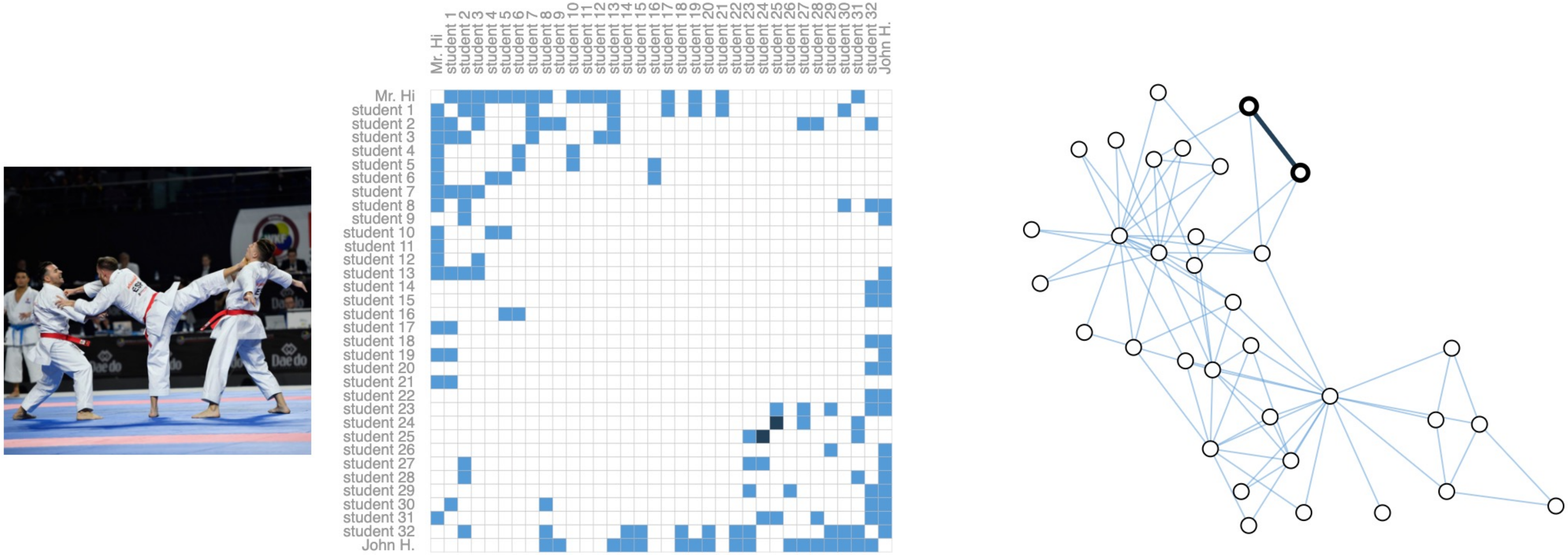
Other Graph Data





(Left) Image of a scene from the play “Othello”. (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

Other Graph Data



(Left) Image of karate tournament. (Center) Adjacency matrix of the interaction between people in a karate club. (Right) Graph representation of these interactions.

Other Graph Data

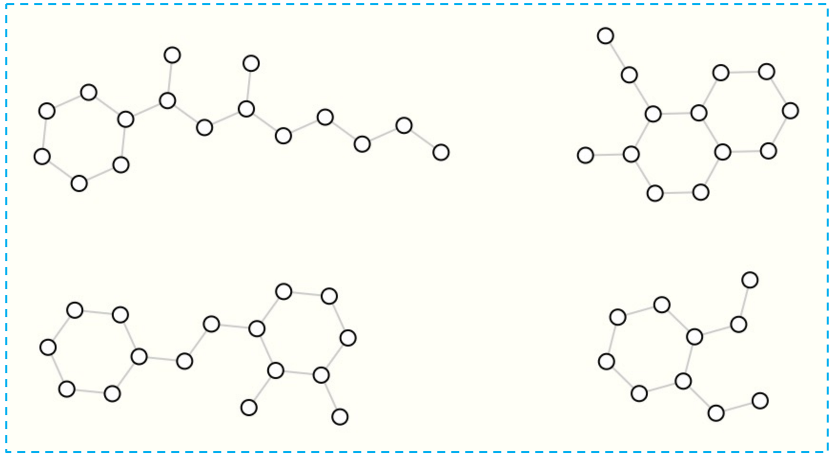
					Edges per node (degree)		
Dataset	**Domain**	**graphs**	**nodes**	**edges**	**min**	**mean**	**max**
karate club	Social network	1	34	78		4.5	17
qm9	Small molecules	134k	≤ 9	≤26	1	2	5
Cora	Citation network	1	23,166	91,500	1	7.8	379
Wikipedia links, English	Knowledge graph	1	12M	378M		62.24	1M

Summary statistics on graphs found in the real world. Numbers are dependent on featurization decisions.

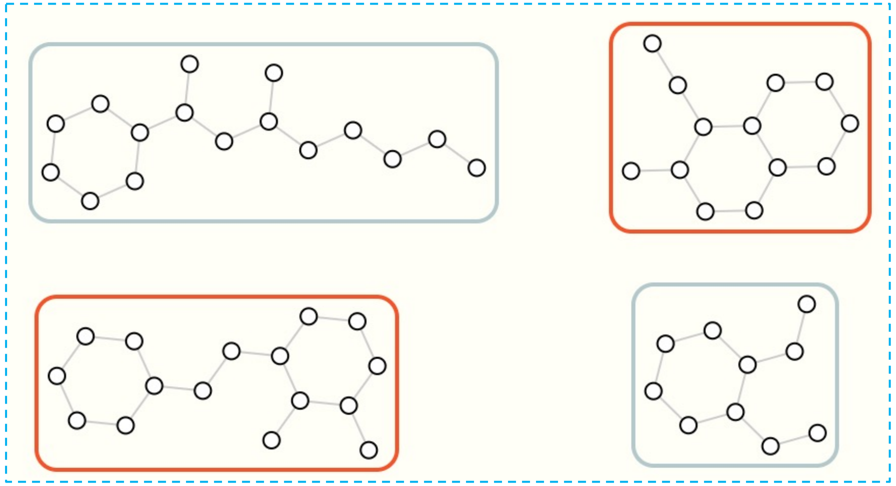
Tasks on Graph Data

Graph-level task

Similar to image classification problems



Input: graphs



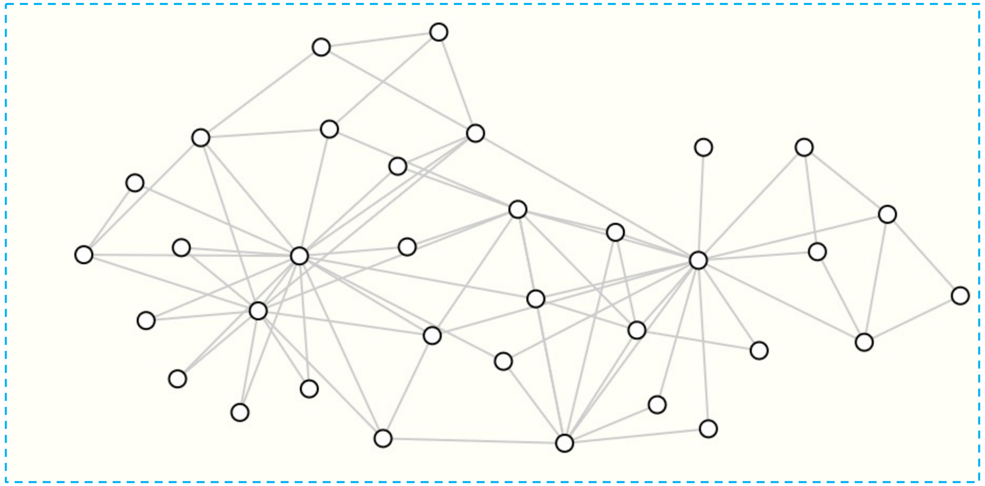
Output: labels for each graph, (e.g., "does the graph contain two rings?")

In a graph-level task, our goal is to predict the property of an entire graph. For example, for a molecule represented as a graph, we might want to predict what the molecule smells like, or whether it will bind to a receptor implicated in a disease.

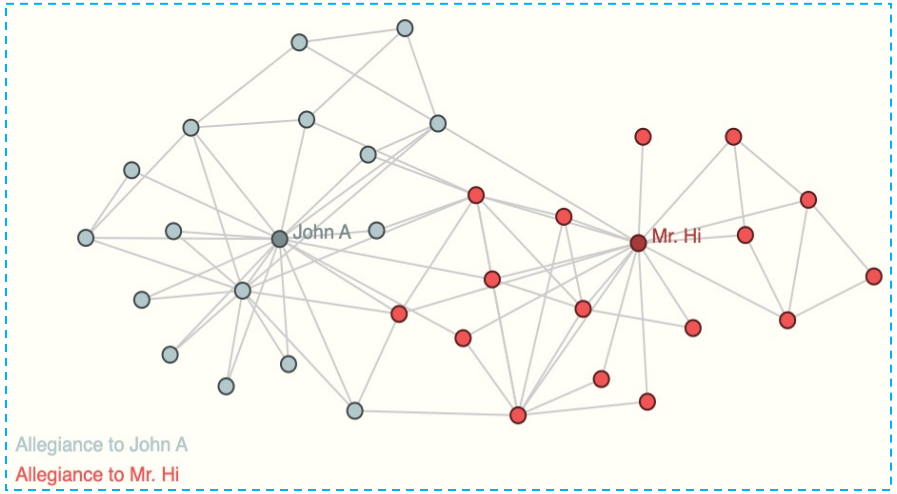
Tasks on Graph Data

Node-level task

Similar to image segmentation problems



Input: graph with unlabled nodes

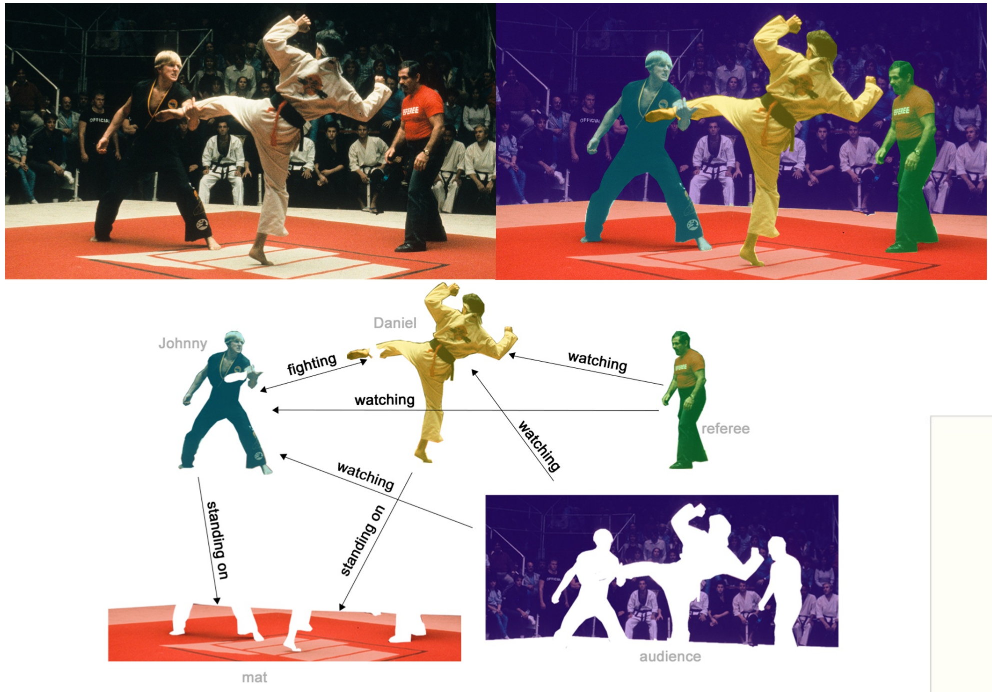


Output: graph node labels

Node-level tasks are concerned with predicting the identity or role of each node within a graph.

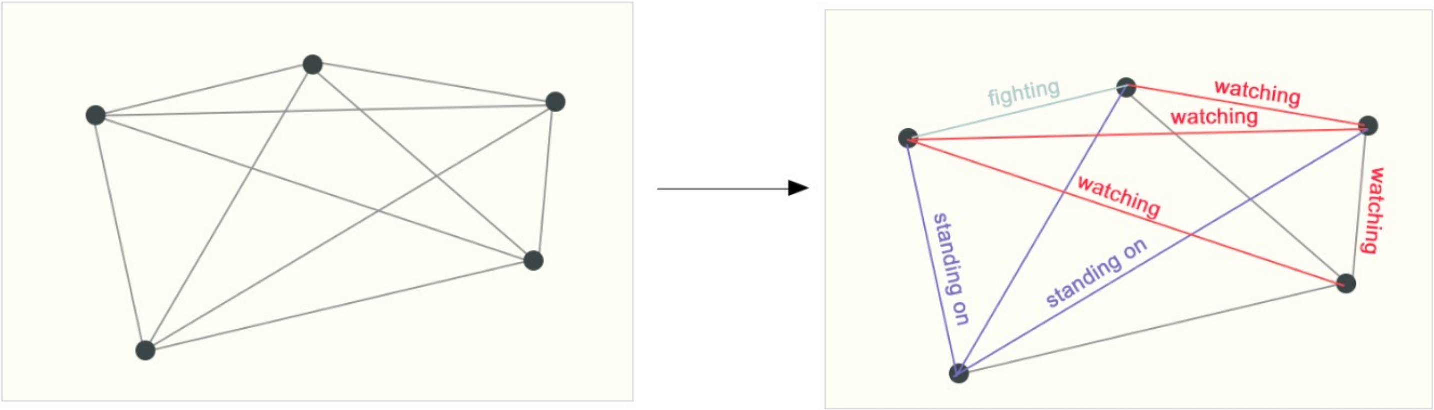
Tasks on Graph Data

Edge-level task



Similar to image scene understanding

Node-level tasks are concerned with predicting the identity or role of each node within a graph.

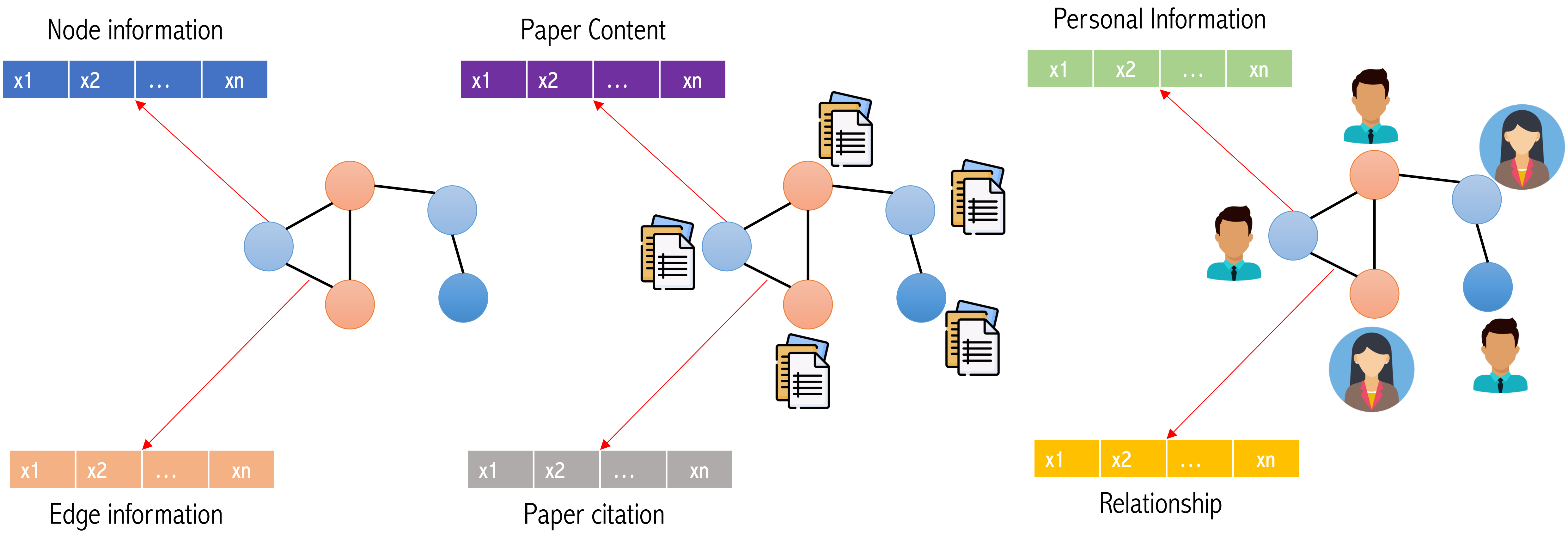


Input: fully connected graph, unlabeled edges

Output: labels for edges

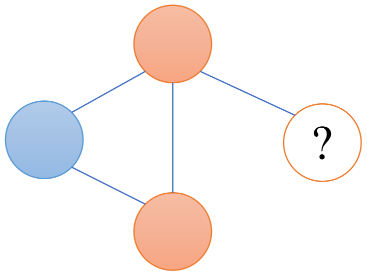
AI VIETNAM
All-in-One Course

A Simple Graph



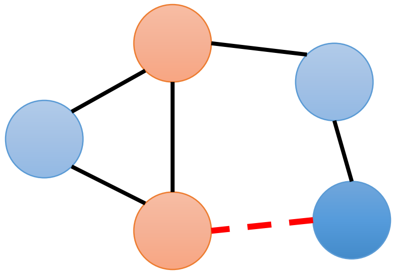
Example

Nodel level prediction



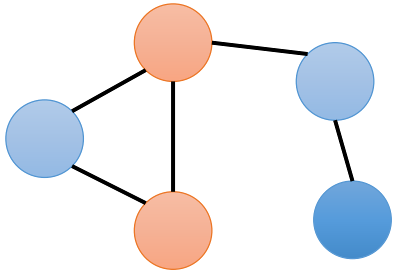
Does this student smoke?
(unlable node)

Edge level predictions
(Link prediction)



Next Youtube
Video?

Graph Level
Predictions

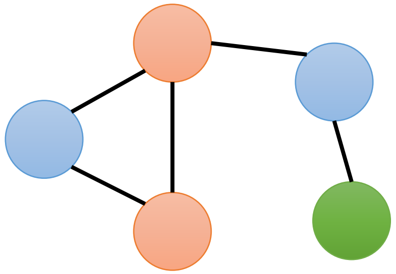


Is this molecule a suitable
drug?

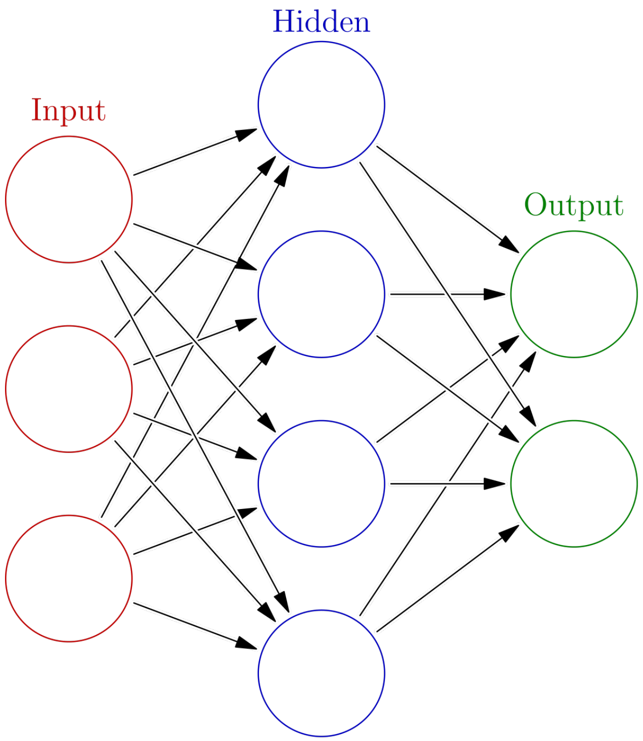
Outline

- Objective
- Introduction to Graph Data
- **Graph Data with Neural Network**
- Node Classification Problem: Cora Citation Dataset
- Summary

Graph Data with Neural Network

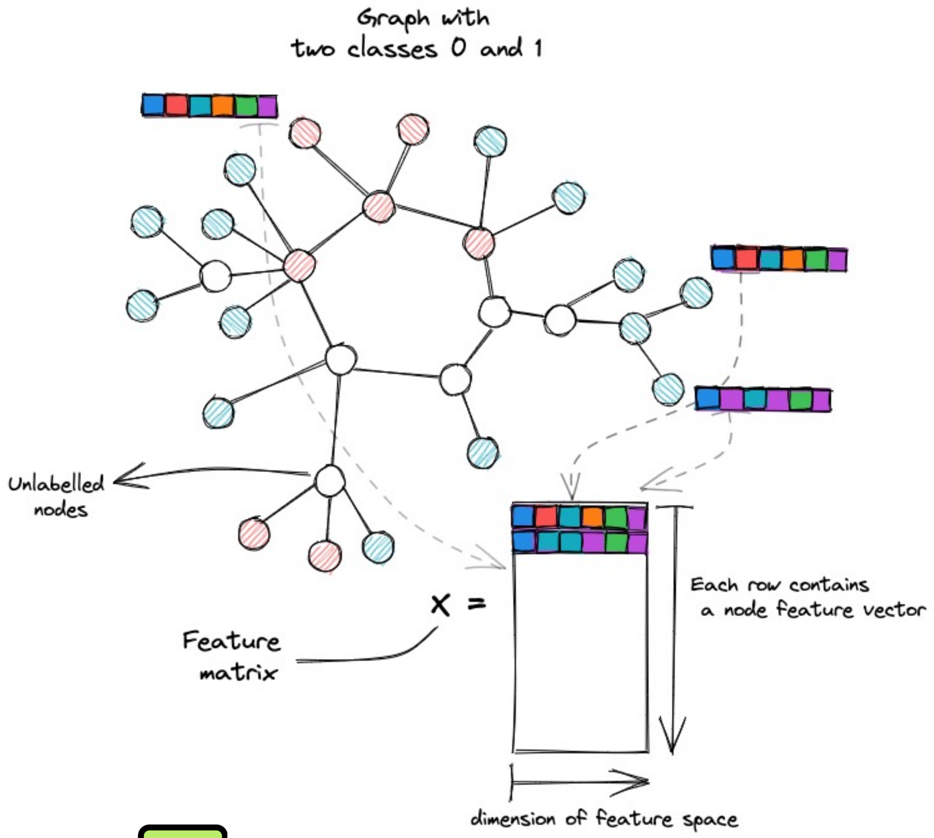


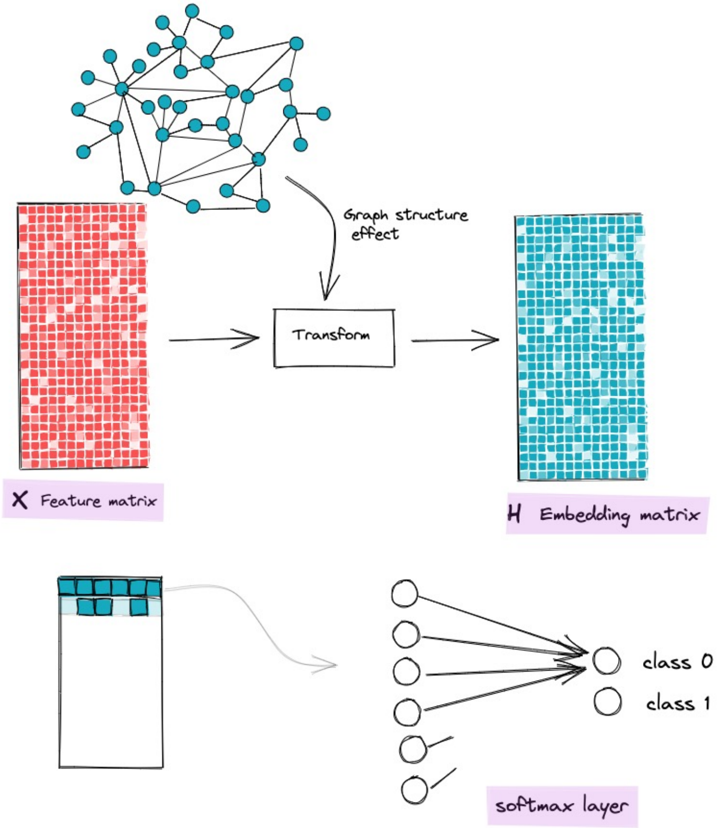
G(V,E)



So, how do we go about solving these different graph tasks with neural networks? The first step is to think about how we will represent graphs to be compatible with neural networks.

Graph Data with Neural Network

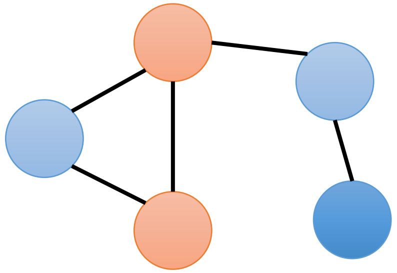


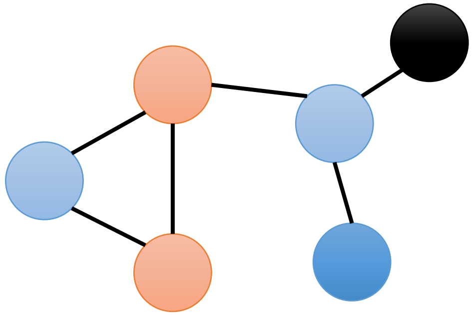


The goal of GNN is to transform node features to features that are aware of the graph structure

Problem with Graph Data

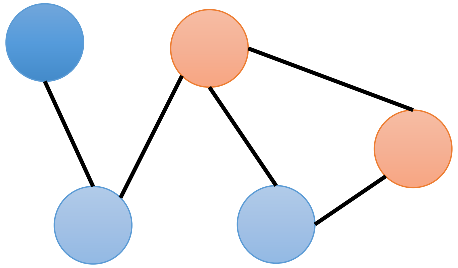
Difference in Size and Shape



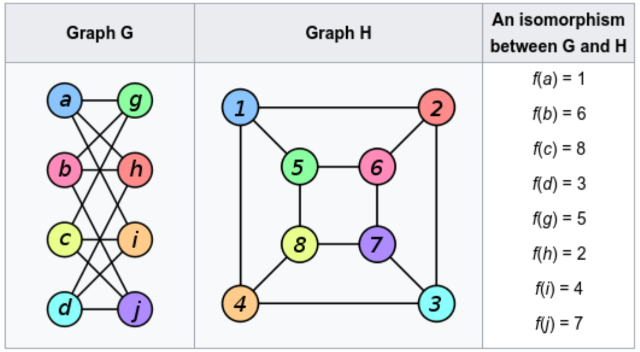


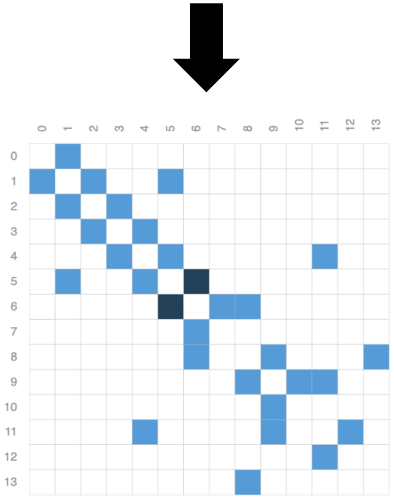
How to handel arbitrary input graph shape?

Isomorphism



Permutaton invariant

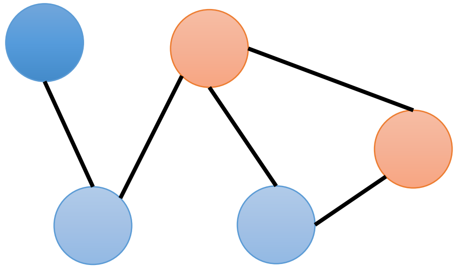




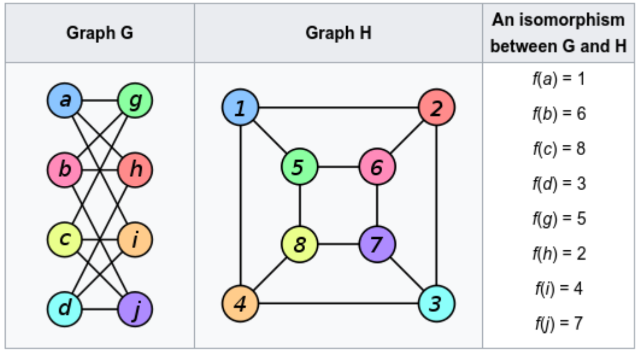
Cannot use adjacency matrix as an input

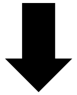
Problem with Graph Data

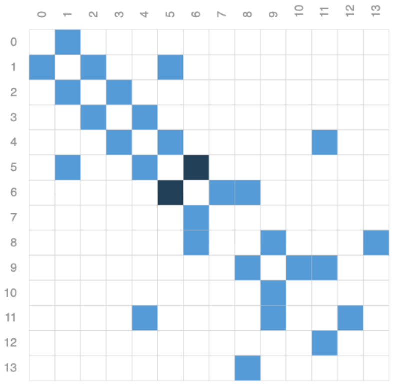
Isomorphism



Permutaton invariant



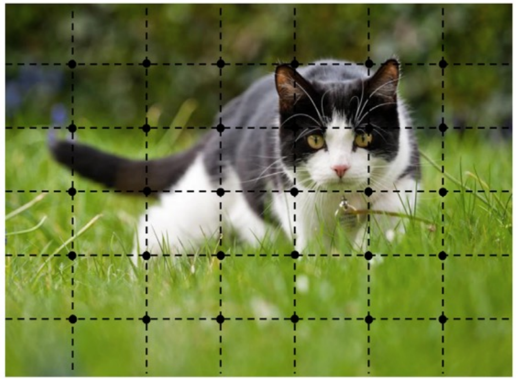




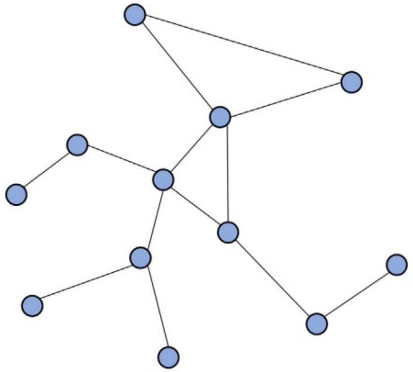
Cannot use adjacency matrix as an input

Grid Structure

Information between nodes



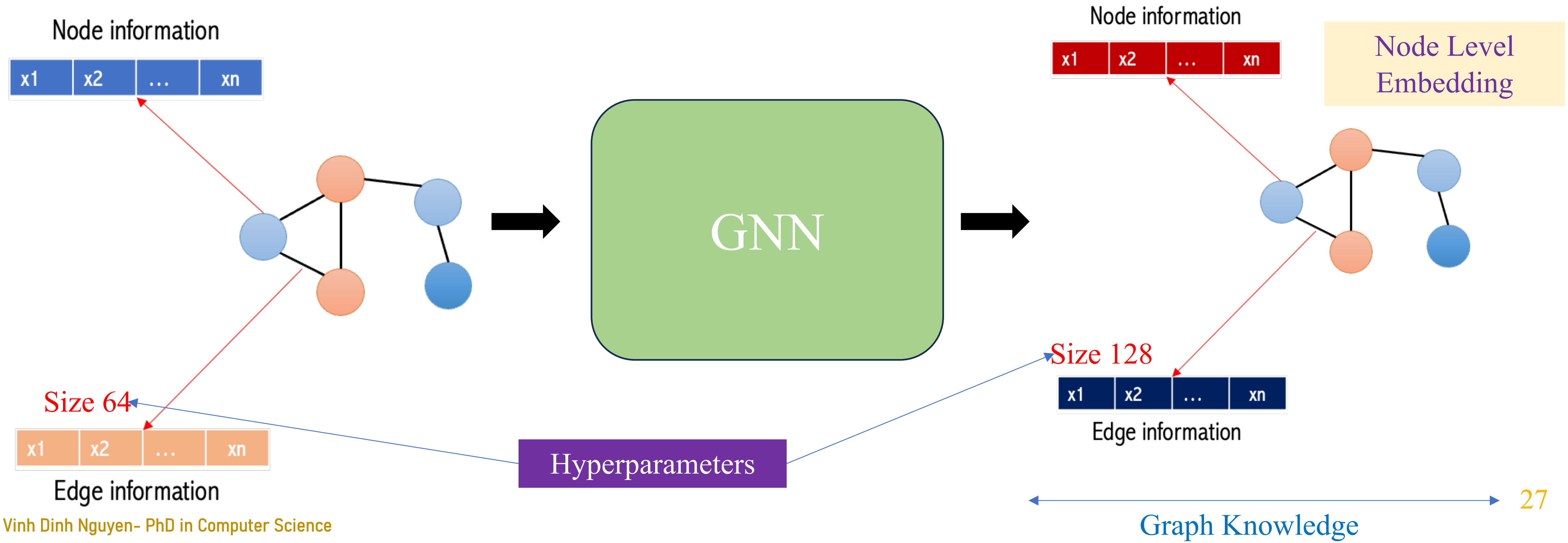
Eulidean space



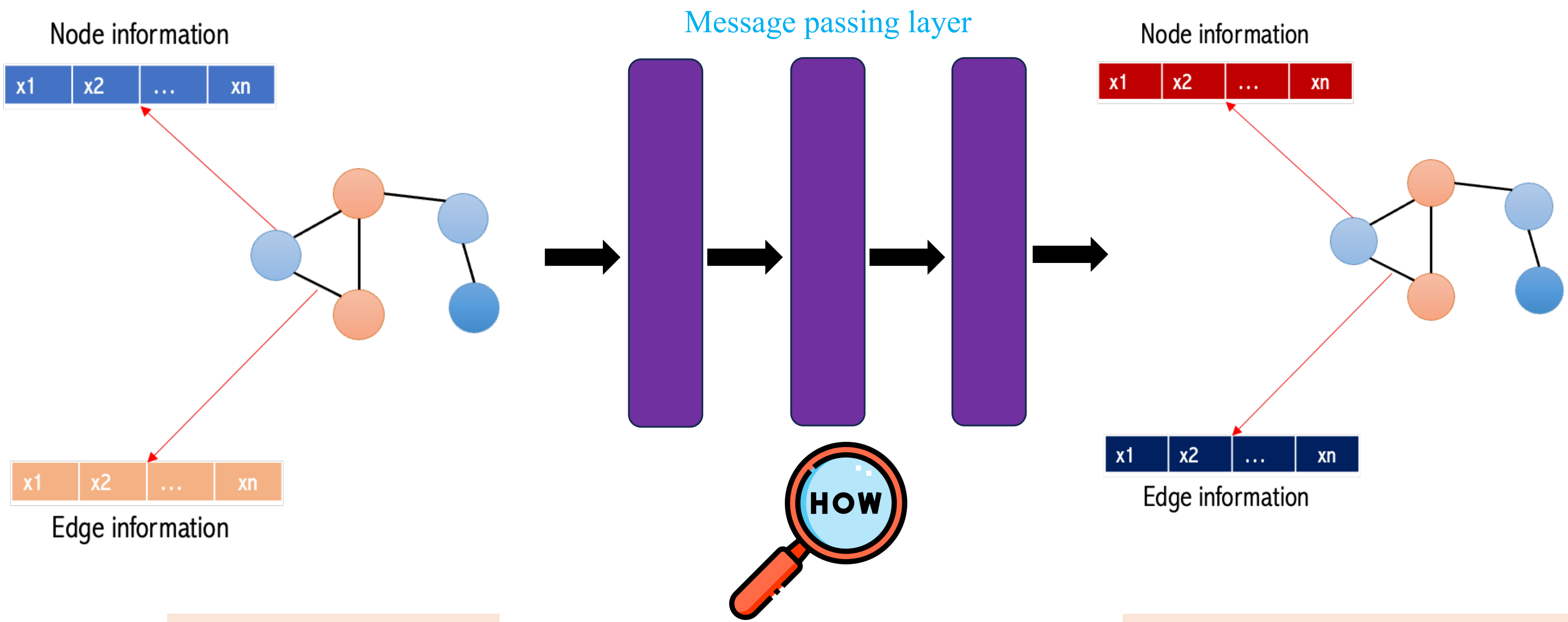
Non-Euclidean space

Fundamental Idea of GNNs

Learning a for neural network suitable representation of graph data
<Representation Learning>>



How do Graph Neural Network Work?

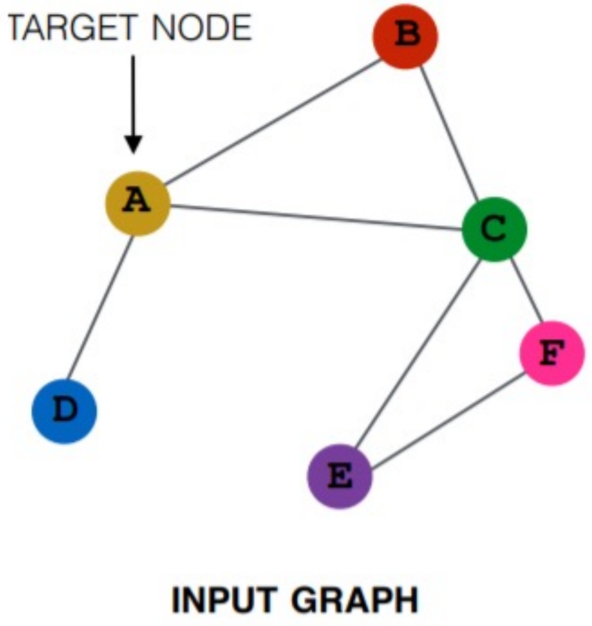


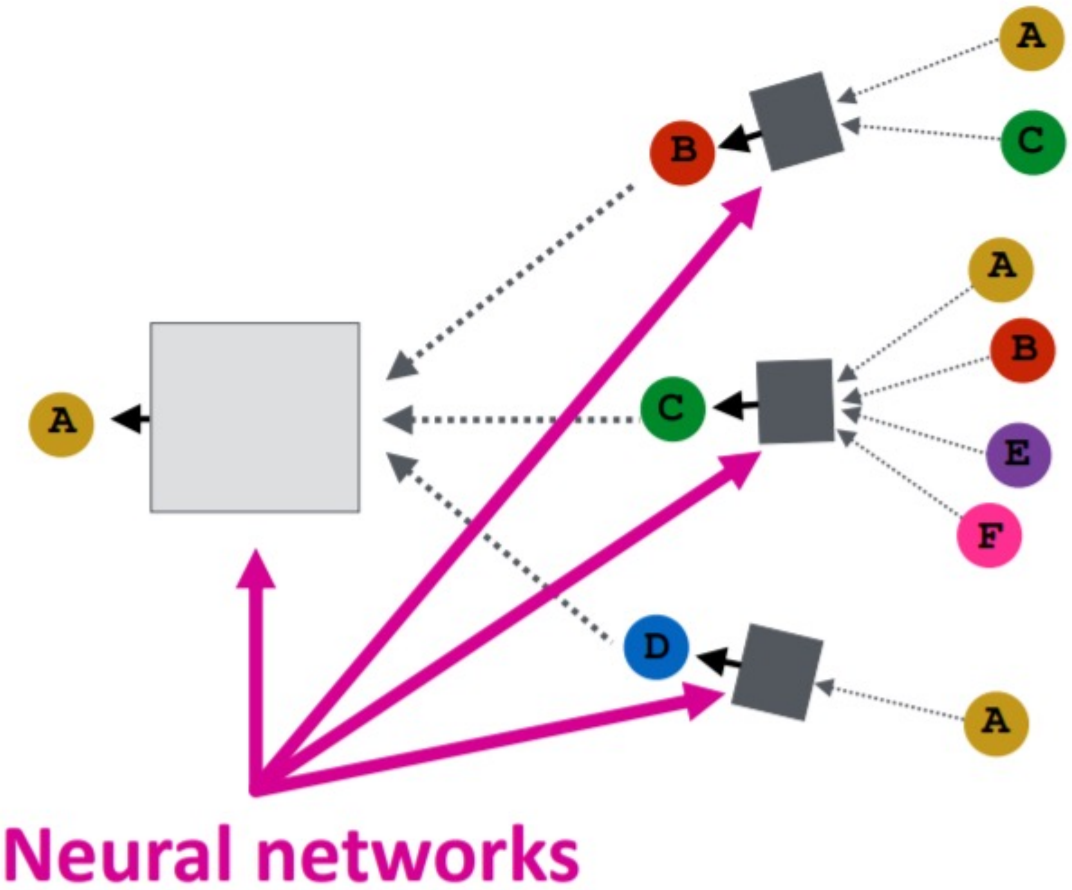
Edge and Node Information

1.Sample Neighborhood 2. Aggregation 3. Update

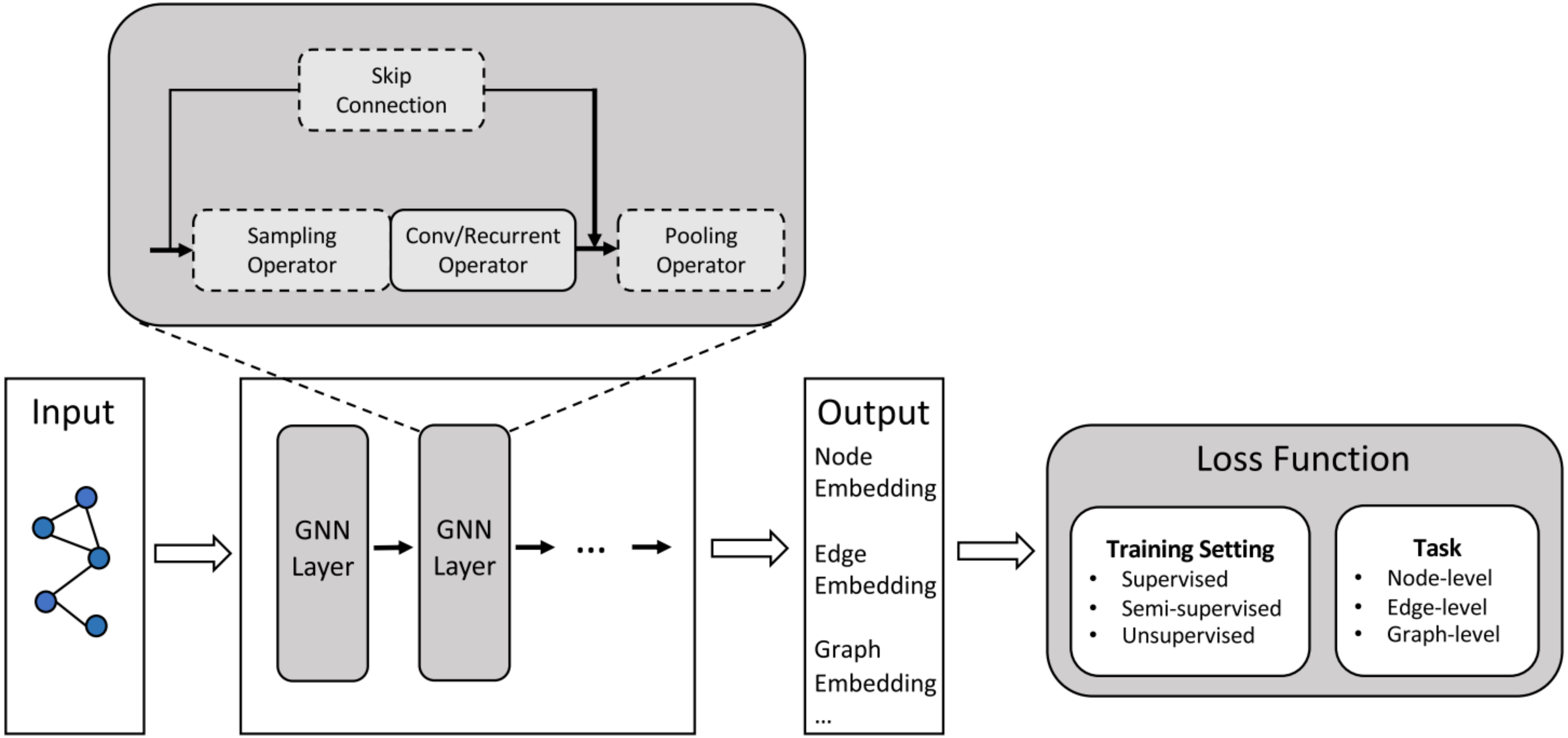
Node Level Embedding Information

How do Graph Neural Network Work?





How do Graph Neural Network Work?



1. Find graph structure.

4. Build model using computational modules.

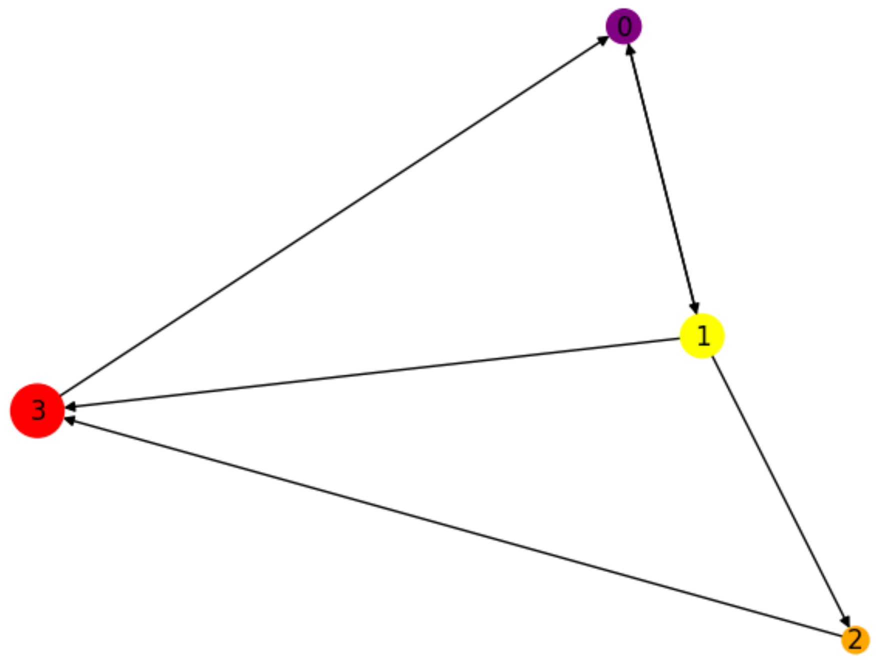
3. Design loss function.

2. Specify graph type and scale.

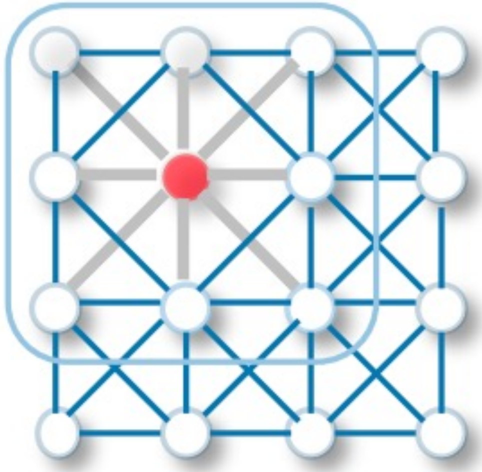
https://arxiv.org/pdf/1812.08434.pdf

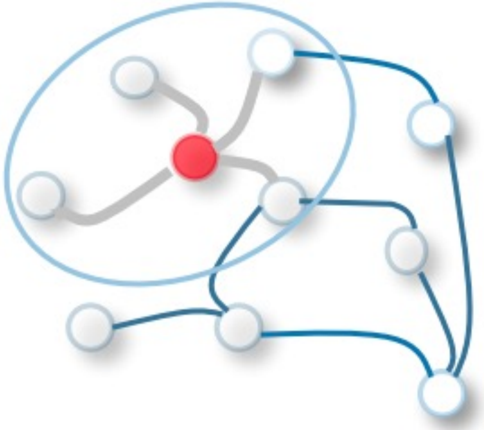
Create a graph using NetworkX

```python
import networkx as nx
H = nx.DiGraph()

#adding nodes
H.add_nodes_from([
  (0, {"color": "purple", "size": 250}),
  (1, {"color": "yellow", "size": 400}),
  (2, {"color": "orange", "size": 150}),
  (3, {"color": "red", "size": 600})
])

#adding edges
H.add_edges_from([
  (0, 1), (1, 2), (1, 0), (1, 3), (2, 3), (3,0)
])

node_colors = nx.get_node_attributes(H, "color").values()
colors = list(node_colors)
node_sizes = nx.get_node_attributes(H, "size").values()
sizes = list(node_sizes)

#Plotting Graph
nx.draw(H, with_labels=True, node_color=colors, node_size=sizes)
```



CNN Vs. GNN: Messsage Passing





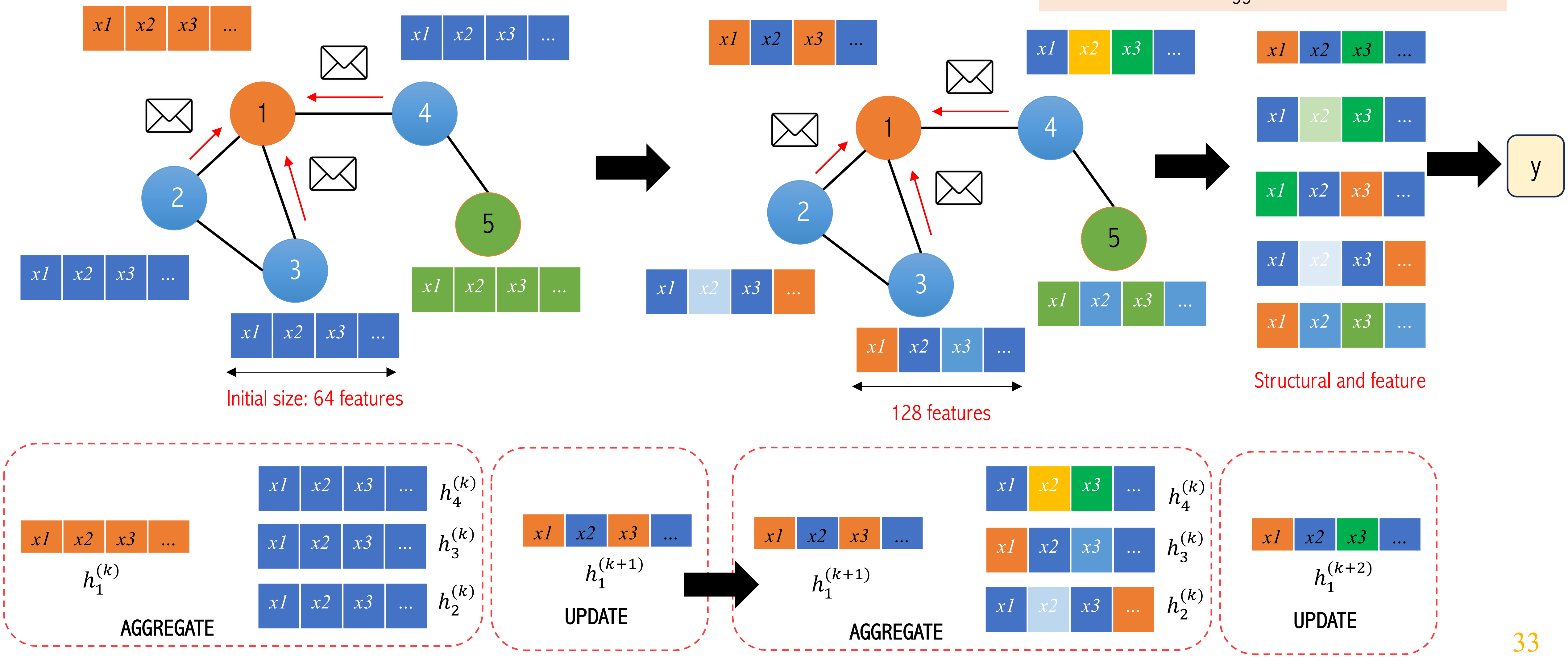
2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size

Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the 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

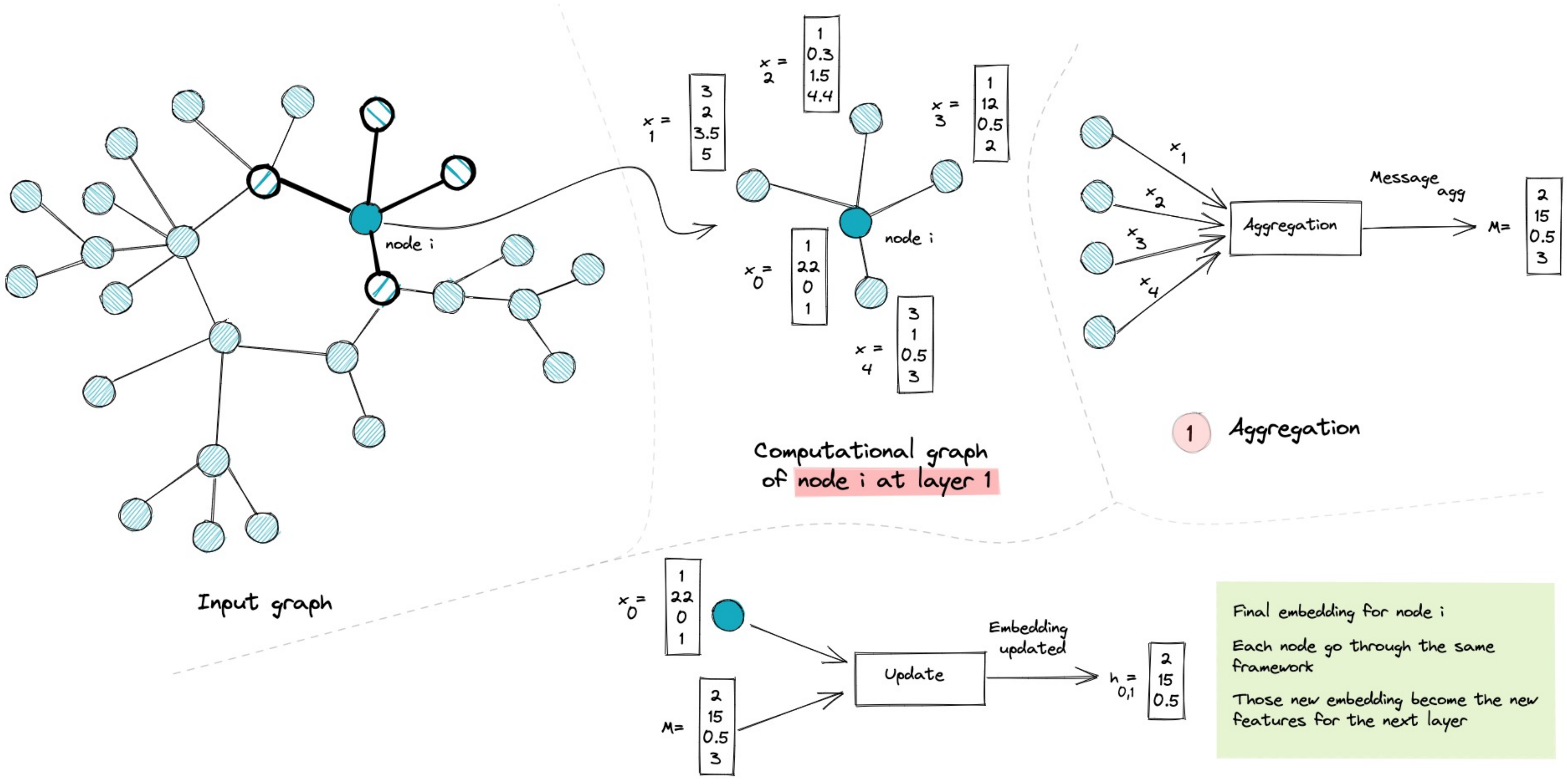
https://arxiv.org/pdf/1901.00596.pdf

Message Passing: Behind the Scene

The local feature aggreation <=> CNN Kernvel



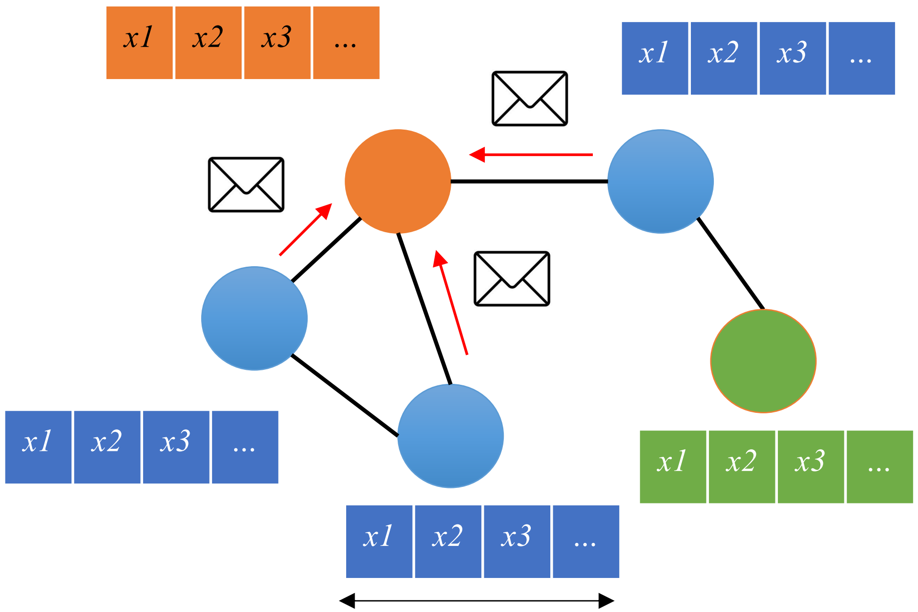
Graph: Example

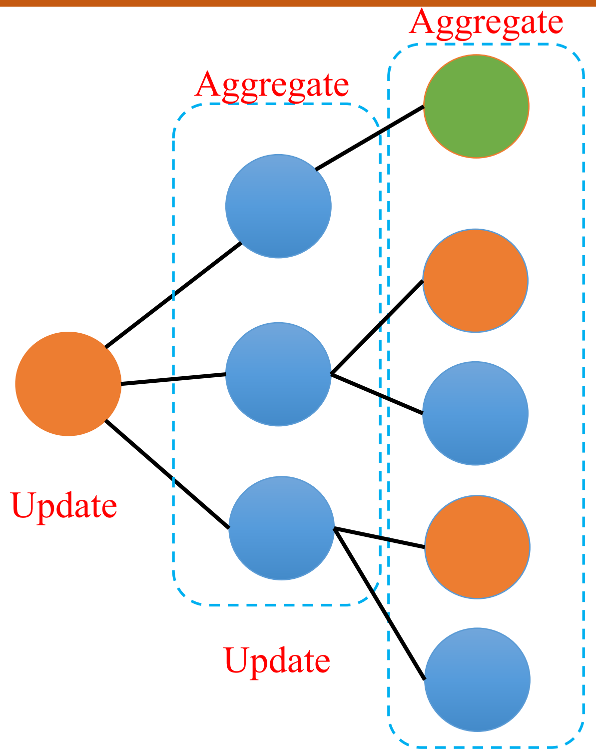


https://towardsdatascience.com/over-smoothing-issue-in-graph-neural-network-bddc8fbc2472

2 Update

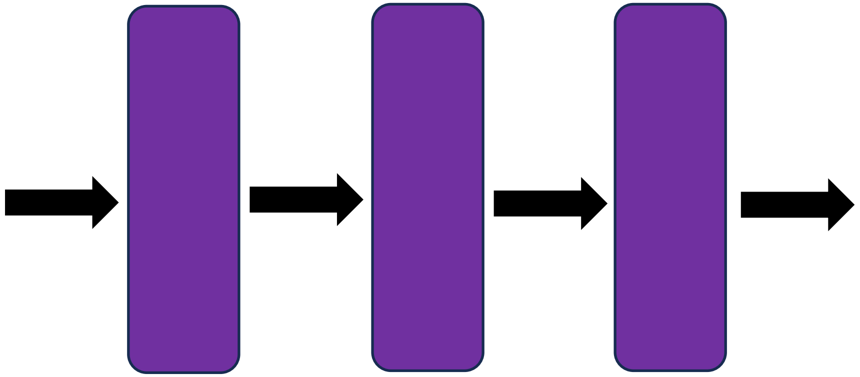
Computation Graph Representation





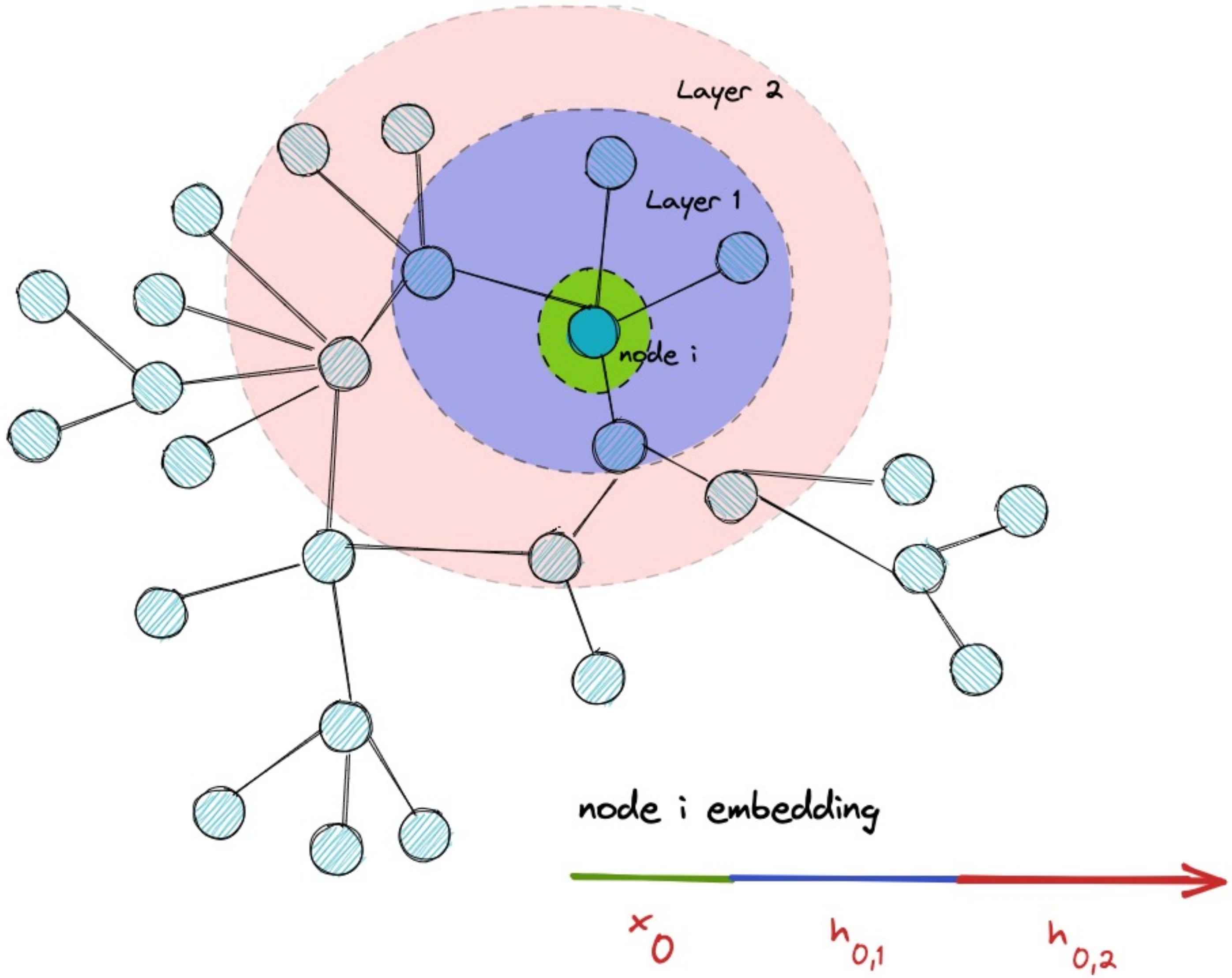
Computationnal Graph for Node v

The number of layers defined by how many neighborhood nodes

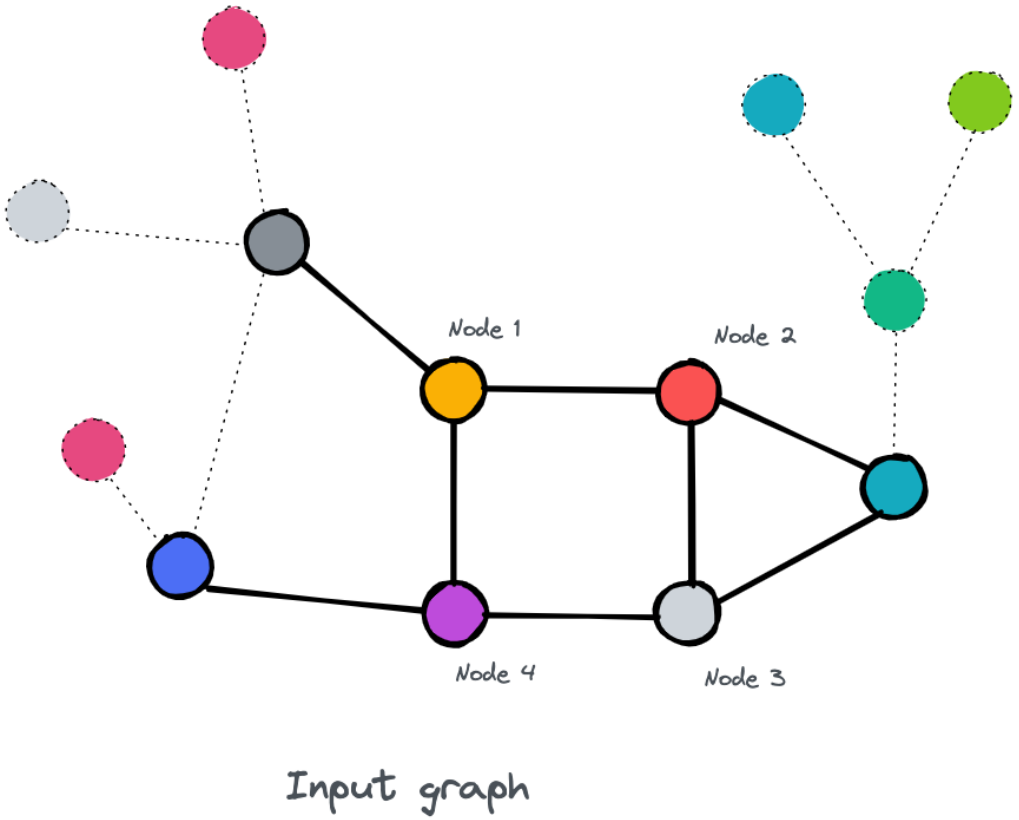


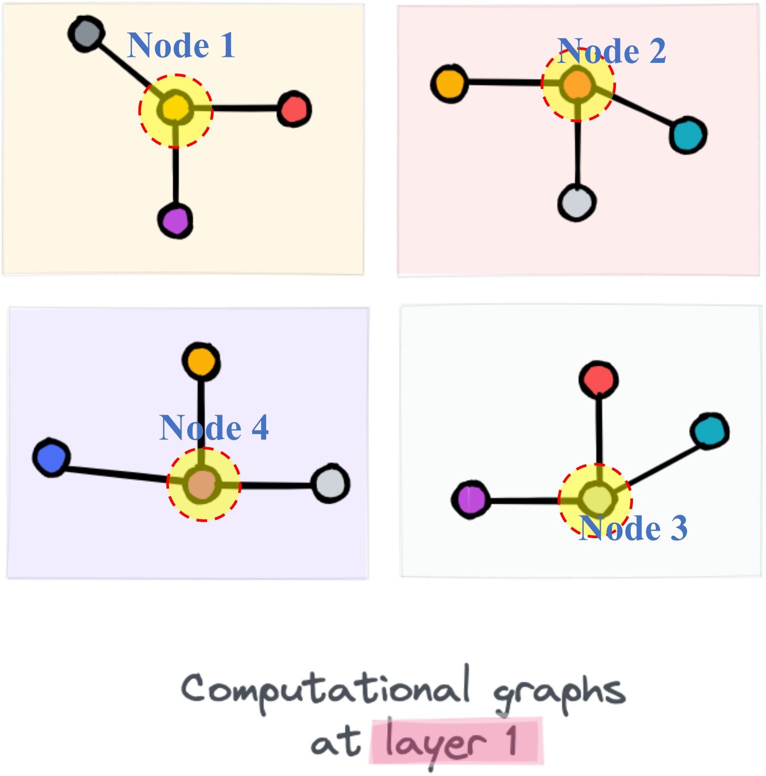
The number of MP-Layer is a hyper parameters

Graph: Example



Over-smoothing in GNN

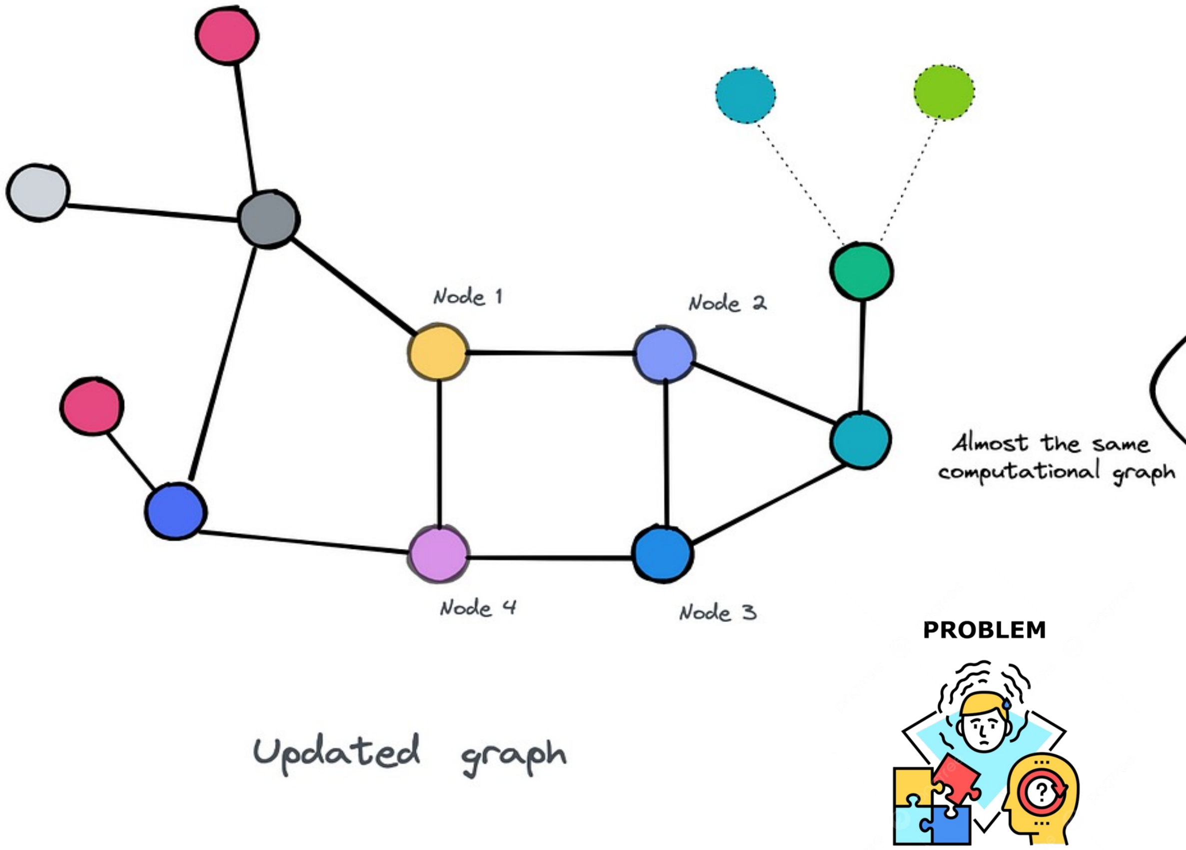


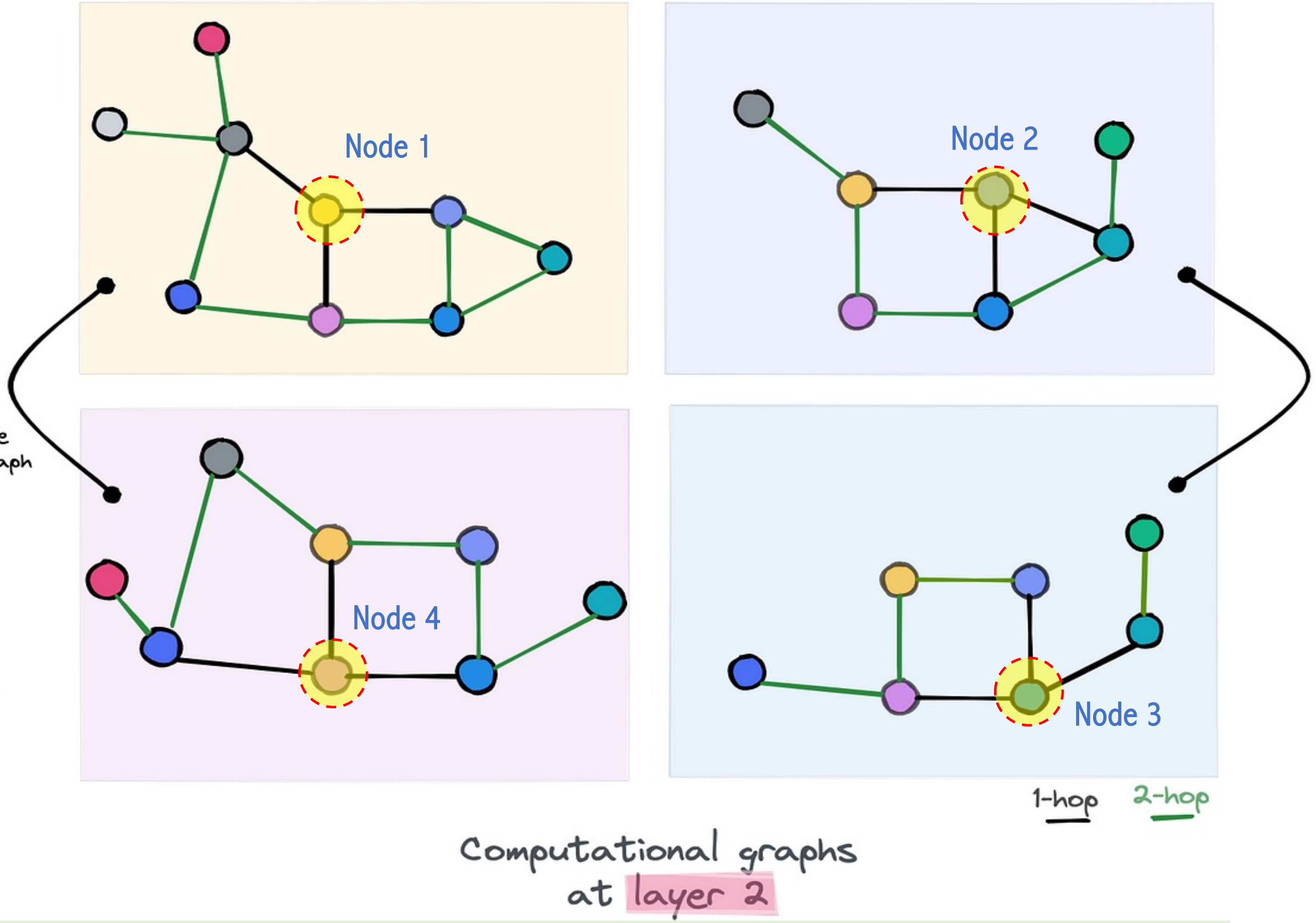


Node 2 and node 3 have almost access to the same information -> We can predict that their embeddings will be slightly similar.

Node 1 and Node 4, they interact with each other but have different neighbors -> We may predict that their new embeddings will be different.

Over-smoothing in GNN

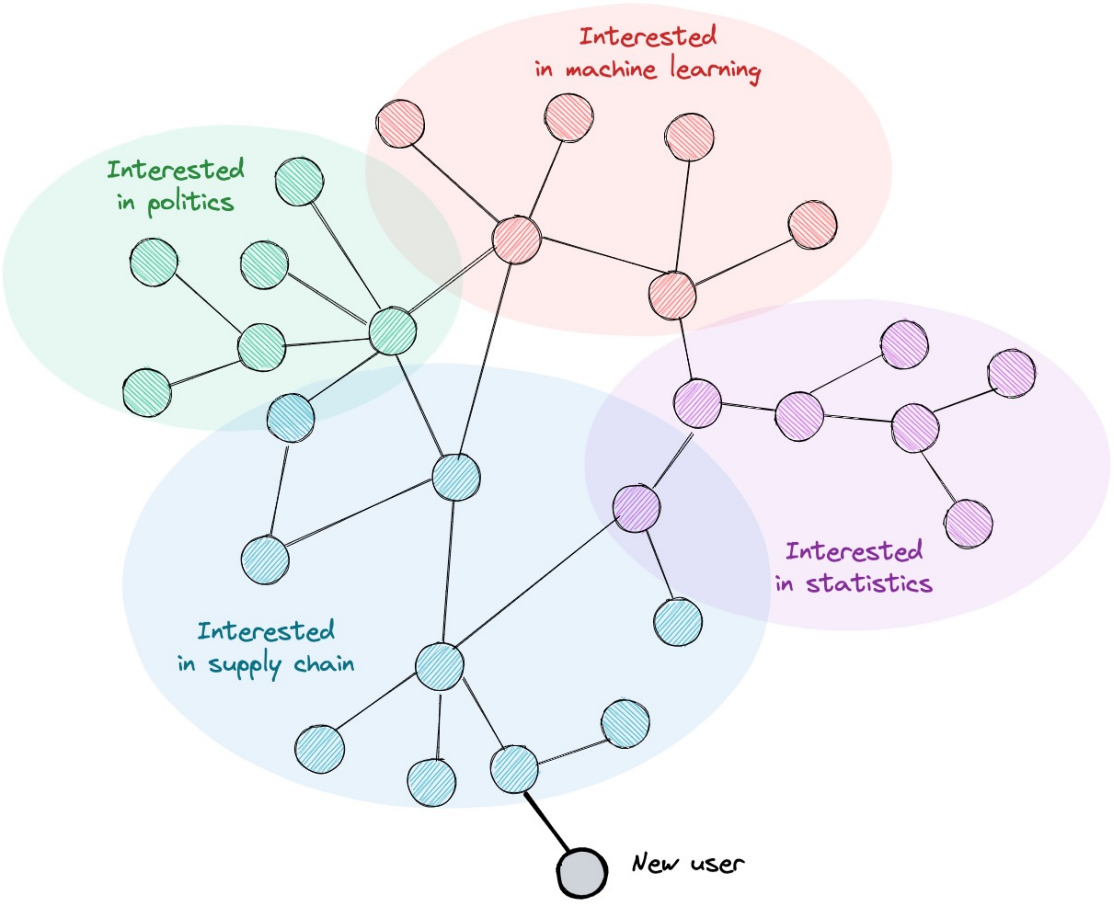




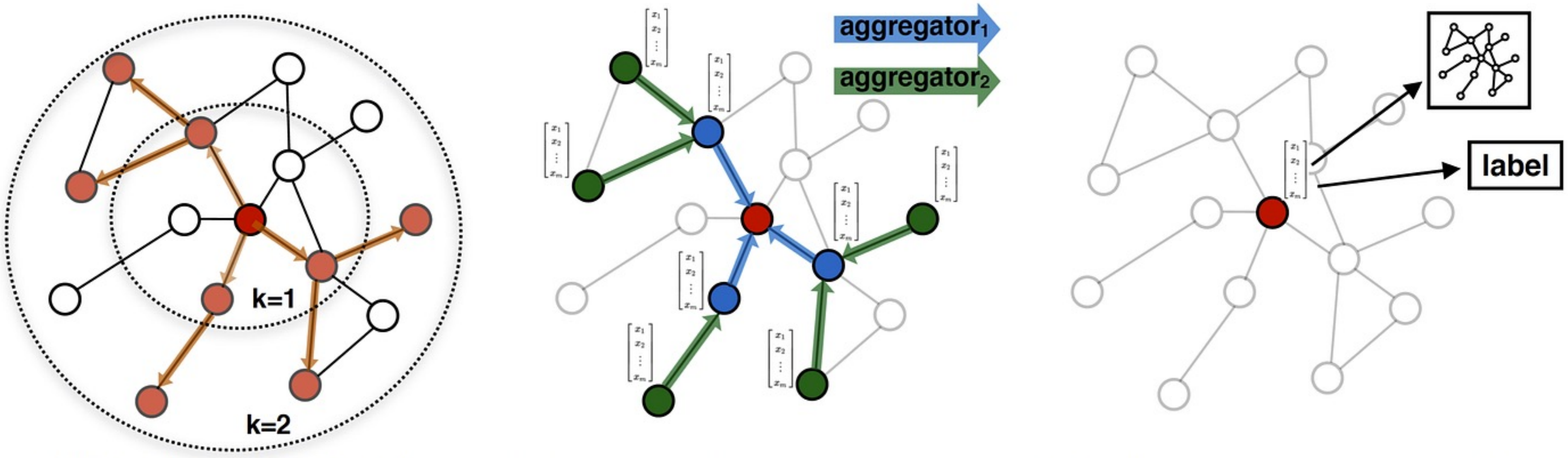
The computational graphs of nodes 1,4, and 2,3 are almost the same respectively

Over-smoothing in GNN

Graph Neural Networks, or GNNs, are really good at working with data that is organized in a graph structure. But sometimes, when we add more layers to a GNN architecture, it doesn't work as well as we would like. This is called over-smoothing.



Why over-smoothing happens?



1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

Reason 1 → More number of layers (depth)
Reason 2 → Default nature of GNNs

How to detect over-smoothing?

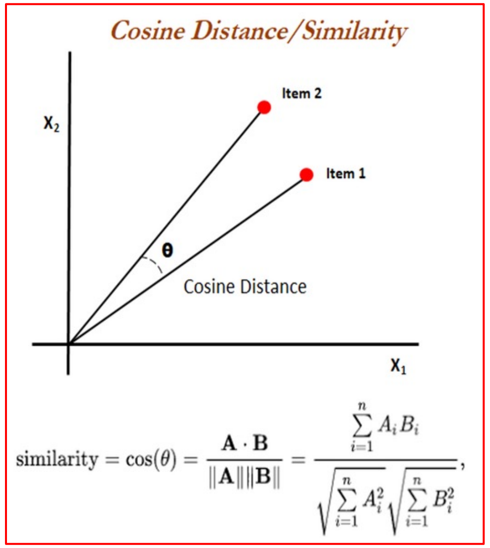
Metric 1: MAD(Mean Average Distance)

MAD computes the Mean Average Distance (MAD) between node representations (embeddings) in the graph

Model	2	3	4	5	6
ARMA	0.629	0.860	0.608	0.305	0.004
ChebGCN	0.557	0.756	0.138	0.024	0.018
DNA	0.665	0.352	0.347	0.172	0.096
FeaSt	0.778	0.770	0.677	0.182	0.072
GAT	0.794	0.704	0.232	0.047	0.005
GCN	0.796	0.765	0.714	0.602	0.289
GGNN	0.661	0.078	0.021	0.033	0.039
GraphSAGE	0.925	0.816	0.632	0.303	0.053
HighOrder	0.629	0.145	0.023	0.004	0.012
HyperGraph	0.828	0.742	0.493	0.046	0.023

#Model Layer

$$D_{ij} = 1 - \frac{\boldsymbol{H}_{i,:} \cdot \boldsymbol{H}_{j,:}}{|\boldsymbol{H}_{i,:}| \cdot |\boldsymbol{H}_{j,:}|} \qquad i, j \in [1, 2, \cdots, n],$$



Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View

Deli Chen,[1] Yankai Lin,[2] Wei Li,[1] Peng Li,[2] Jie Zhou,[2] Xu Sun[1]
[1]MOE Key Lab of Computational Linguistics, School of EECS, Peking University
[2]Pattern Recognition Center, WeChat AI, Tencent Inc., China
{chendeli,liweitj47,xusun}@pku.edu.cn, {yankailin,patrickpli,withtomzhou}@tencent.com,

How to detect over-smoothing?

Metric 2 : MADGap

$$\mathrm{MADGap} = \mathrm{MAD}^{rmt} - \mathrm{MAD}^{neb}, \qquad (5)$$

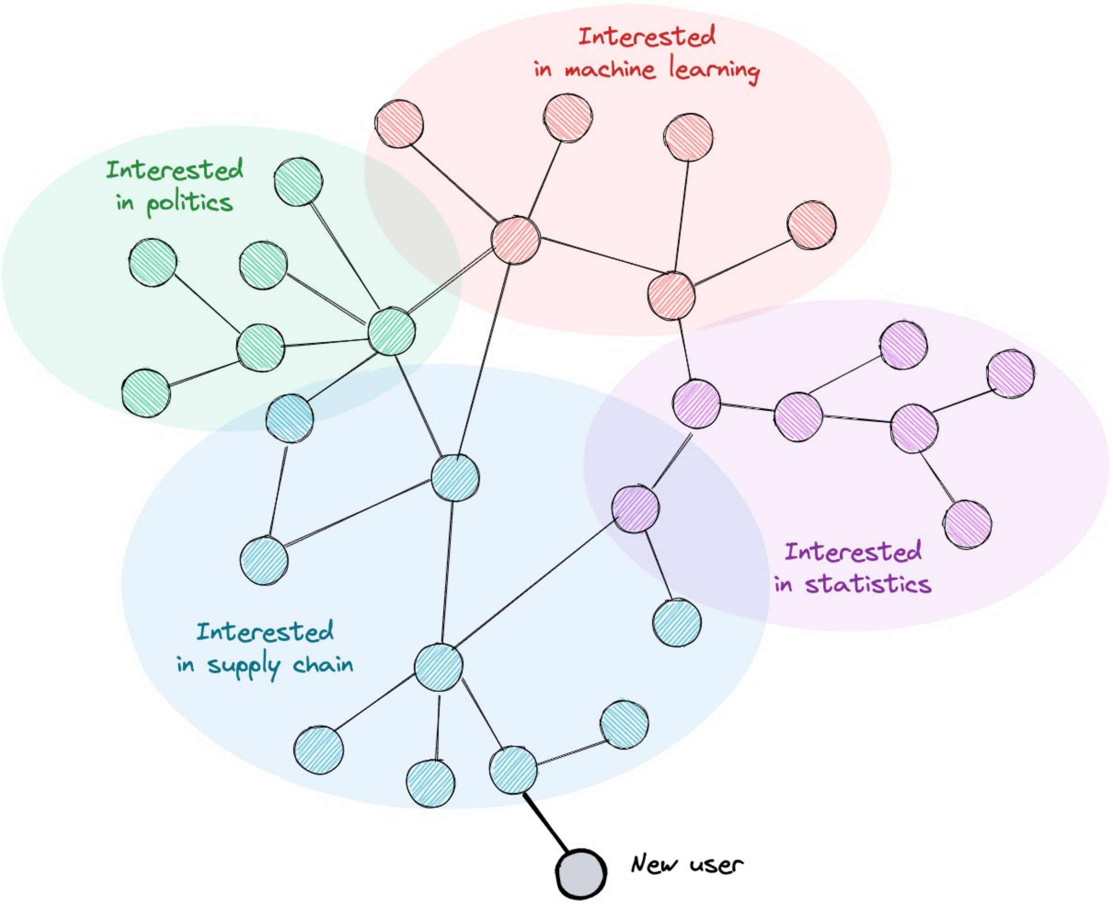
where MAD^{rmt} is the MAD value of the remote nodes in the graph topology and MAD^{neb} is the MAD value of the neighbouring nodes.

It is based on the main hypothesis that when nodes interact, they have access to either important information from nodes of the same class or noise from nodes of other classes.

Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View

Deli Chen,[1] Yankai Lin,[2] Wei Li,[1] Peng Li,[2] Jie Zhou,[2] Xu Sun[1]
[1]MOE Key Lab of Computational Linguistics, School of EECS, Peking University
[2]Pattern Recognition Center, WeChat AI, Tencent Inc., China
{chendeli,liweitj47,xusun}@pku.edu.cn, {yankailin,patrickpli,withtomzhou}@tencent.com,

Over-smoothing in GNN



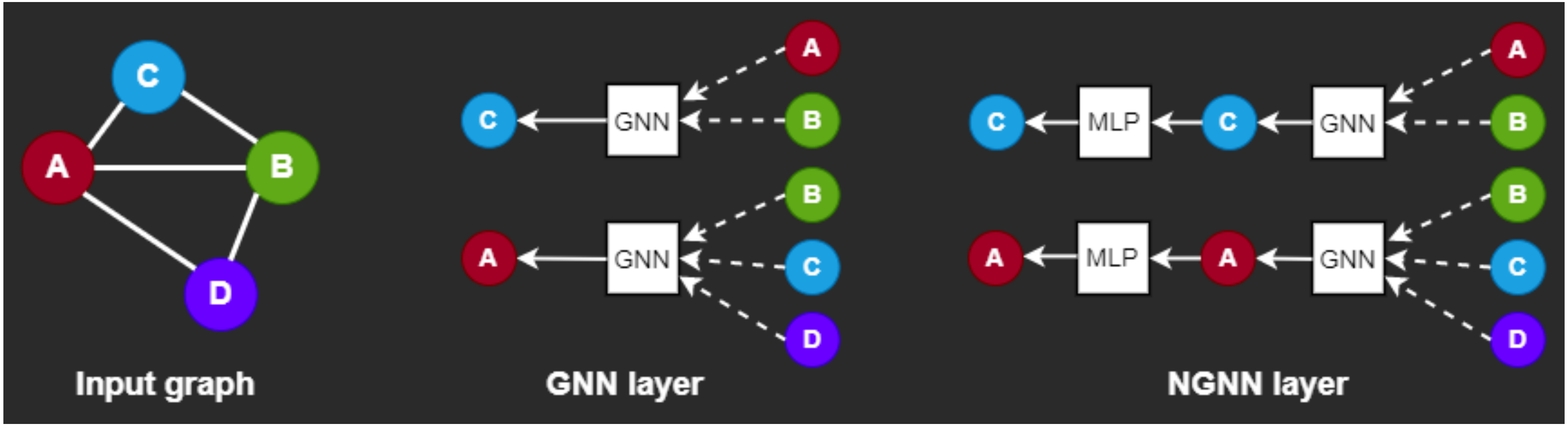
How to reduce the effect of over-smoothing.

We encounter a trade-off between a low-efficiency model and a model with more depth but less expressivity in terms of node representations

Imagine that we're dealing with a social network graph with thousands of nodes. Some new users just signed in to the platform and subscribed to their friend's profiles. Our goal is to find topic suggestions to fill their feed.

Over-smoothing in GNN

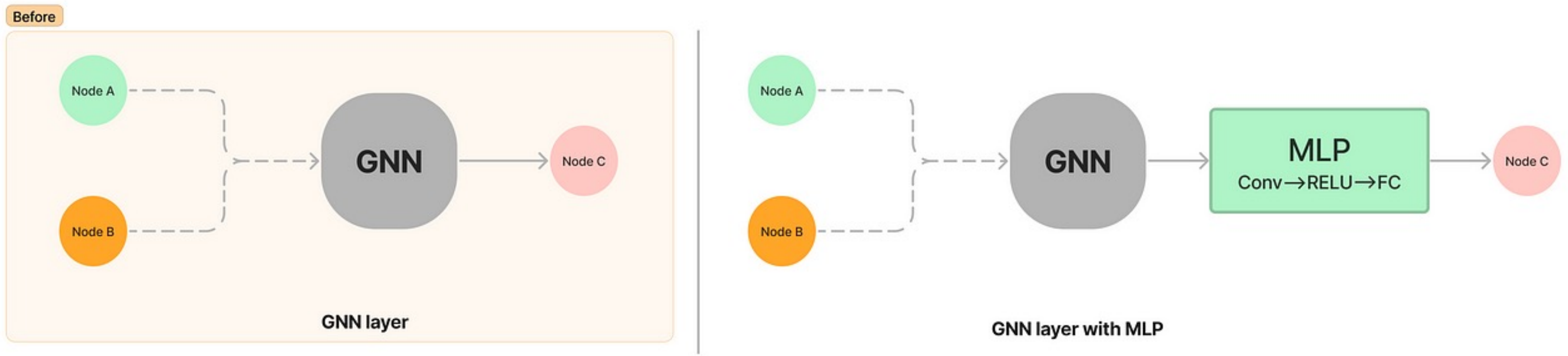
Solution → Inserting nonlinear feedforward neural network layer(s) within each GNN layer.



https://www.dgl.ai/blog/2022/11/28/ngnn.html

Over-smoothing in GNN

Solution → Inserting nonlinear feedforward neural network layer(s) within each GNN layer.



https://www.dgl.ai/blog/2022/11/28/ngnn.html

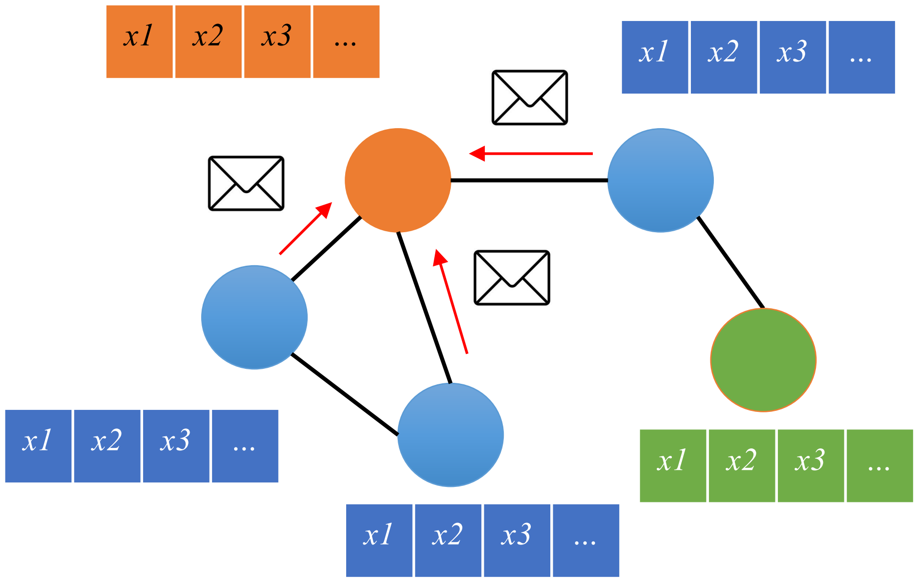
Over-smoothing in GNN

Solution → Inserting nonlinear feedforward neural network layer(s) within each GNN layer.

Dataset	Metric	Model		Performance
ogbn-proteins	ROC-AUC(%)	GraphSage+Cluster Sampling	Vanilla	67.45 ± 1.21
			+NGNN	**68.12 ± 0.96**
ogbn-products	Accuracy(%)	GraphSage	Vanilla	78.27 ± 0.45
			+NGNN	**79.88 ± 0.34**
		GAT+Neighbor Sampling	Vanilla	79.23 ± 0.16
			+NGNN	**79.67 ± 0.09**
ogbl-collab	hit@50(%)	GCN	Vanilla	49.52 ± 0.70
			+NGNN	**53.48 ± 0.40**
		GraphSage	Vanilla	51.66 ± 0.35
			+NGNN	**53.59 ± 0.56**
ogbl-ppa	hit@100(%)	SEAL-DGCNN	Vanilla	48.80 ± 3.16
			+NGNN	**59.71 ± 2.45**
		GCN	Vanilla	18.67 ± 1.32
			+NGNN	**36.83 ± 0.99**

https://www.dgl.ai/blog/2022/11/28/ngnn.html

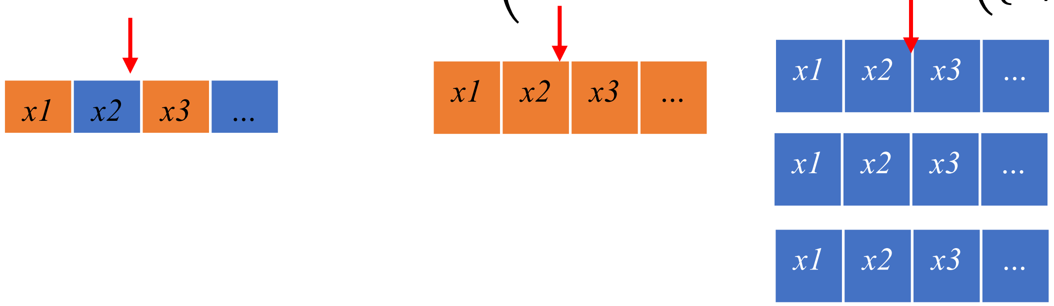
Message Passing: Math is Fun

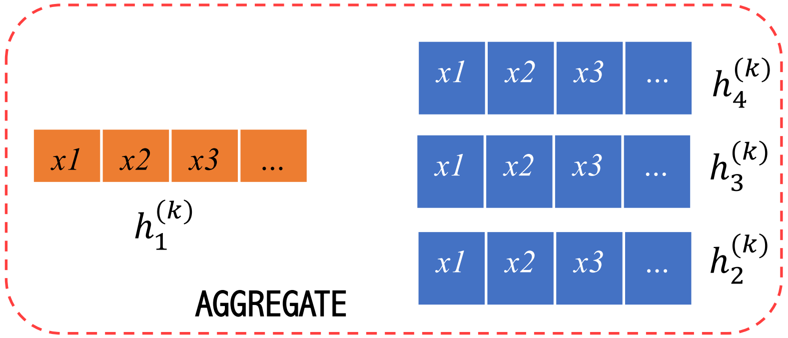


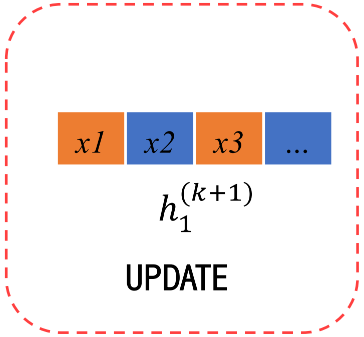
Mean
Max
Neural Network
Recurrent Neural Network

Mean
Max
Normalization Sum
Neural Network

$$h_u^{k+1} = UPDATE^{(k)}\left(h_u^{k+1}, AGGREGATE^{(k)}\left(\{h_V^k\}, \forall v \in \mathrm{N}(u)\right)\right)$$







Message Passing: Variants

AGGREGATE
(permutation invariant)

UPDATE

Graph Convolutional Networks, Kipf and Welling [2016]

Self-loop

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right)$$

Sum of normalized neighbor embeddings

Multi-Layer-Perceptron as Aggregator, Zaheer et al. [2017]

Aggregated message

$$\mathbf{m}_{\mathcal{N}(u)} = \boxed{\mathrm{MLP}_\theta}\left(\sum_{v \in N(u)} \mathrm{MLP}_\phi(\mathbf{h}_v)\right)$$

trainable!

Send states through a MLP

Graph Attention Networks, Veličković et al. [2017]

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v \qquad \alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}]\right)}$$

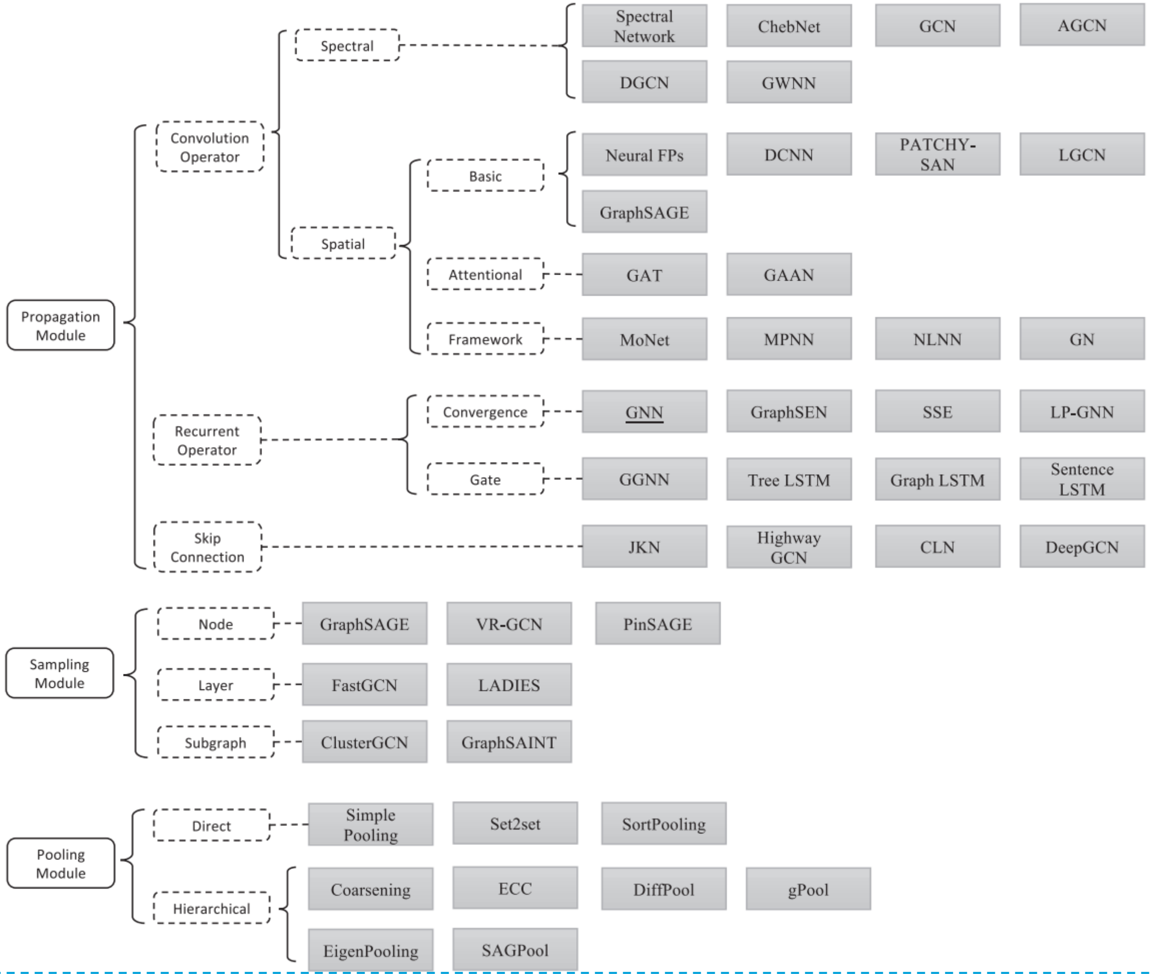
Attention weights

Gated Graph Neural Networks, Li et al. [2015]

$$\mathbf{h}_u^{(k)} = \mathrm{GRU}(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)})$$

Recurrent update of the state

Message Passing: Variants



Different variants of recurrent operators.

Variant	Aggregator	Updater
GGNN	$\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{h}^{t-1}_k + \mathbf{b}$	$\mathbf{z}^t_v = \sigma(\mathbf{W}^z\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^z\mathbf{h}^{t-1}_v)$ $\mathbf{r}^t_v = \sigma(\mathbf{W}^r\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^r\mathbf{h}^{t-1}_v)$ $\tilde{\mathbf{h}}^t_v = \tanh(\mathbf{W}\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}(\mathbf{r}^t_v \odot \mathbf{h}^{t-1}_v))$ $\mathbf{h}^t_v = (1-\mathbf{z}^t_v)\odot\mathbf{h}^{t-1}_v + \mathbf{z}^t_v\odot\tilde{\mathbf{h}}^t_v$
Tree LSTM (Child sum)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^i\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \mathbf{U}^f\mathbf{h}^{t-1}_k$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^o\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^u\mathbf{h}^{t-1}_k$	$\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{h}^{ti}_{\mathcal{N}_v} + \mathbf{b}^i)$ $\mathbf{f}^t_{vk} = \sigma(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{h}^{tf}_{\mathcal{N}_v k} + \mathbf{b}^f)$ $\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{h}^{to}_{\mathcal{N}_v} + \mathbf{b}^o)$ $\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{h}^{tu}_{\mathcal{N}_v} + \mathbf{b}^u)$ $\mathbf{c}^t_v = \mathbf{i}^t_v\odot\mathbf{u}^t_v + \sum_{k\in\mathcal{N}_v}\mathbf{f}^t_{vk}\odot\mathbf{c}^{t-1}_k$ $\mathbf{h}^t_v = \mathbf{o}^t_v\odot\tanh(\mathbf{c}^t_v)$
Tree LSTM (N-ary)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^i_l\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \sum_{l=1}^{K}\mathbf{U}^f_{kl}\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^o_l\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^u_l\mathbf{h}^{t-1}_{vl}$	
Graph LSTM in (Peng et al., 2017)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^i_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \mathbf{U}^f_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^o_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^u_{m(v,k)}\mathbf{h}^{t-1}_k$	

Graph neural networks: A review of methods and applications

Jie Zhou [a,1], Ganqu Cui [a,1], Shengding Hu [a], Zhengyan Zhang [a], Cheng Yang [b], Zhiyuan Liu [a,*], Lifeng Wang [c], Changcheng Li [c], Maosong Sun [a]

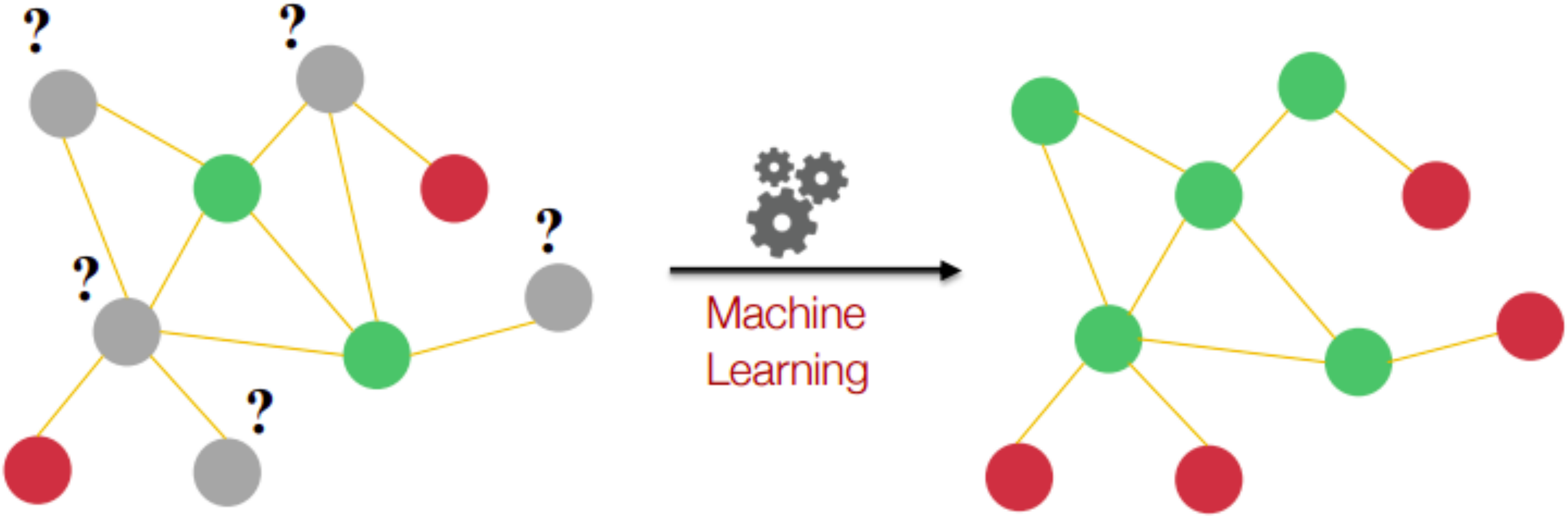
[a] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[b] School of Computer Science, Beijing University of Posts and Telecommunications, China
[c] Tencent Incorporation, Shenzhen, China

QUIZ TIME

Outline

- Objective
- Introduction to Graph Data
- Graph Data with Neural Network
- Node Classification Problem: Cora Citation Dataset
- Summary

Node Classification Problem



GNN: Node Classification

Knowledge Graphs and Node Classification

We have one large graph and not many individual graphs (like molecules)
We infere on unlabeled nodes in this large graph and hence perform node-level predictions
--> We have to use different nodes of the graph depending on what we want to do

Dataset Introduction: Cora Citation Dataset in PyTorch Geometric

The Cora dataset consists of 2708 scientific publications classified into one of seven classes.
Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.

- The dictionary consists of 1433 unique words.
- Nodes = Publications (Papers, Books ...)
- Edges = Citations Node Features = word vectors
- 7 Labels = Pubilcation
- type e.g. Neural_Networks, Rule_Learning, Reinforcement_Learning, Probabilistic_Methods...

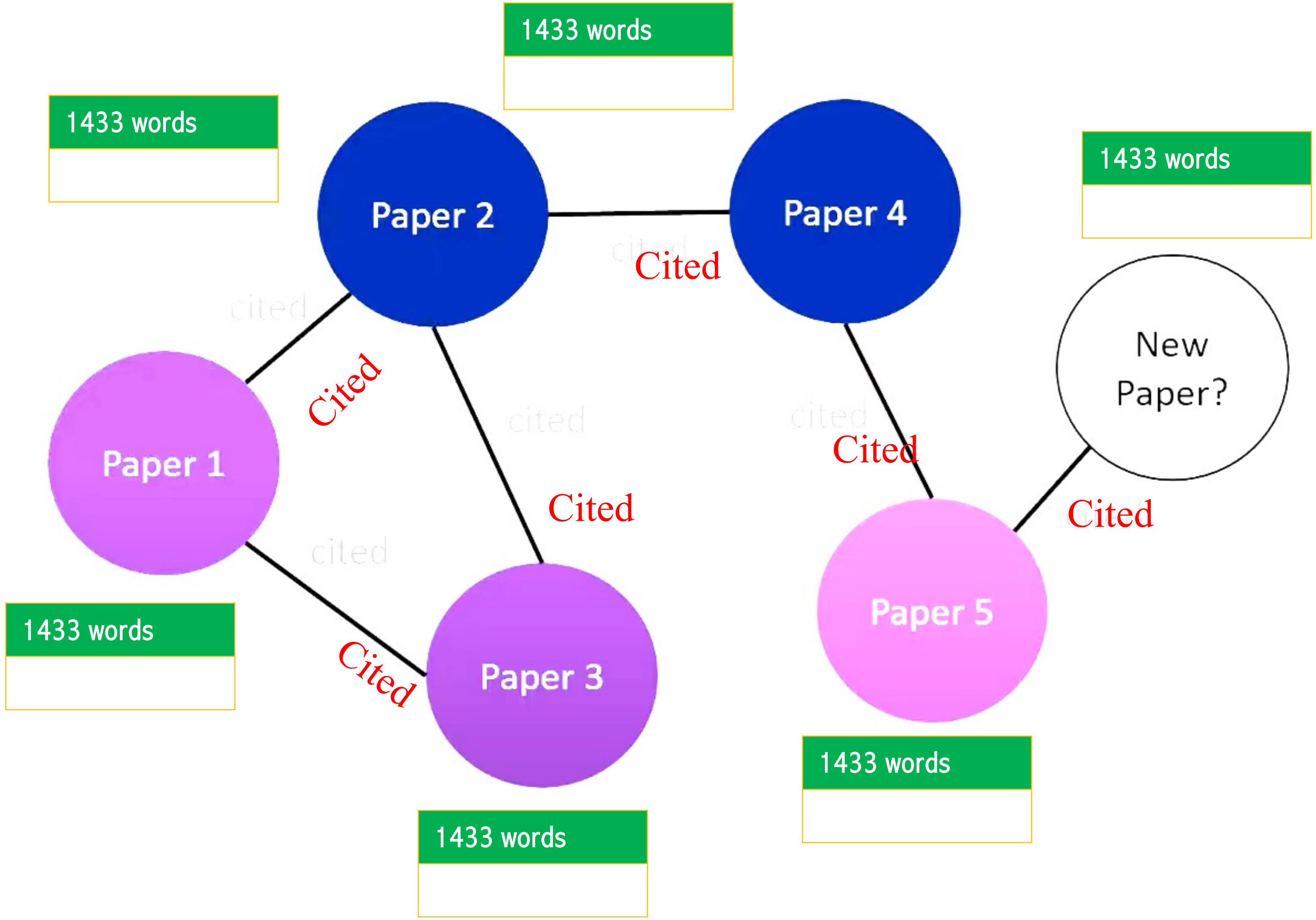
BoW Representation

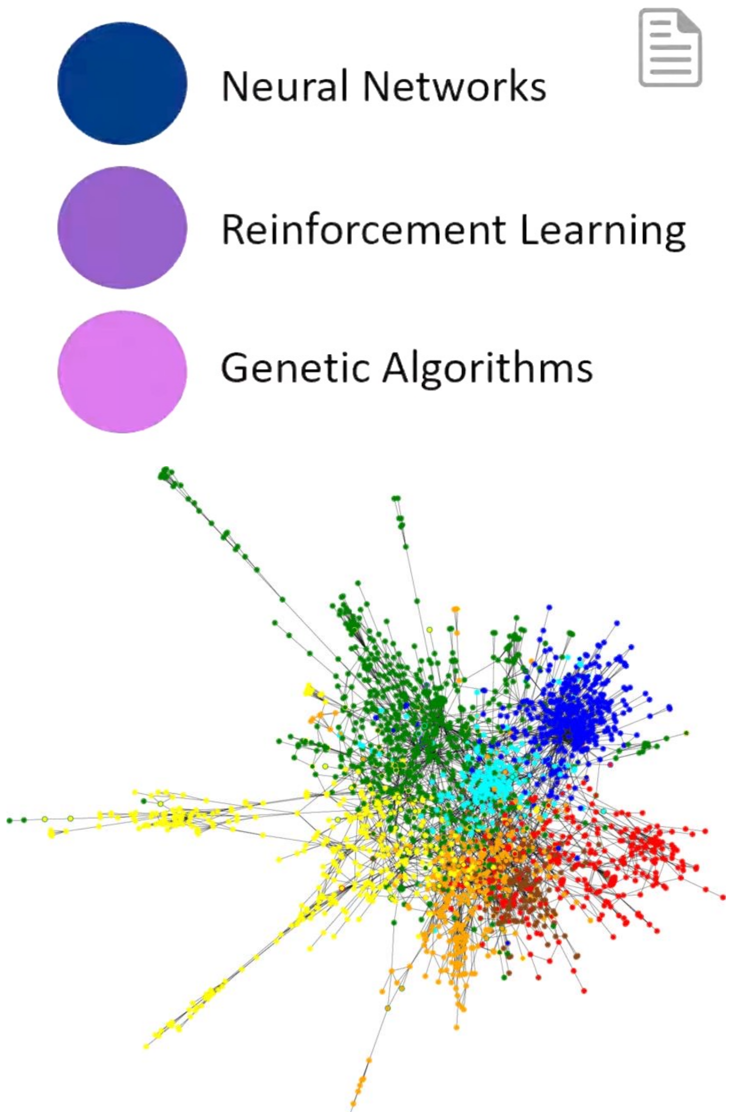
Dictionary

Neural	Network	ReLU	Reinforcement	... 1433 words
2	7	1	0	

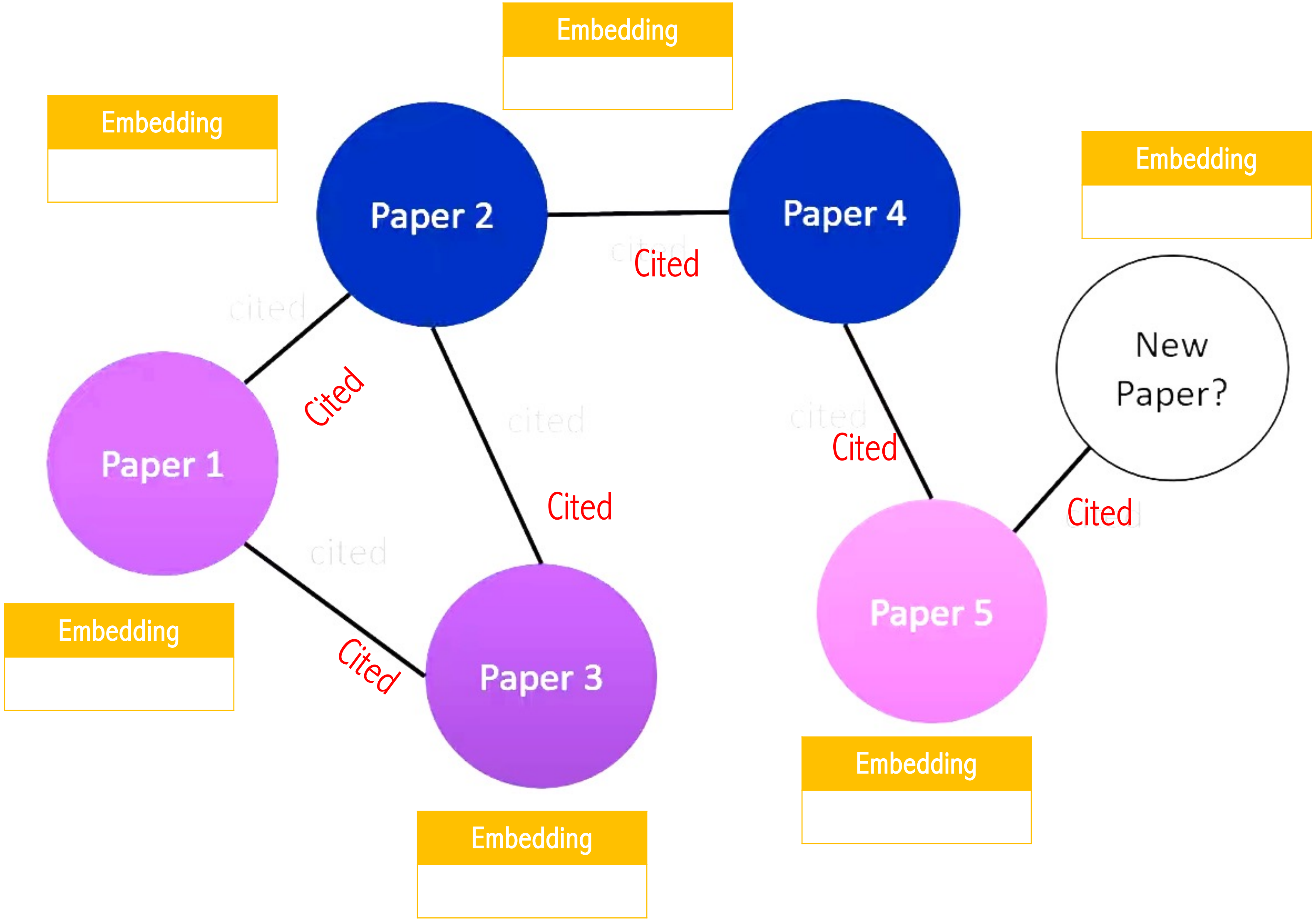
Neural	Network	ReLU	Reinforcement	... 1433 words
0.43	0.88	0.03	0.00	

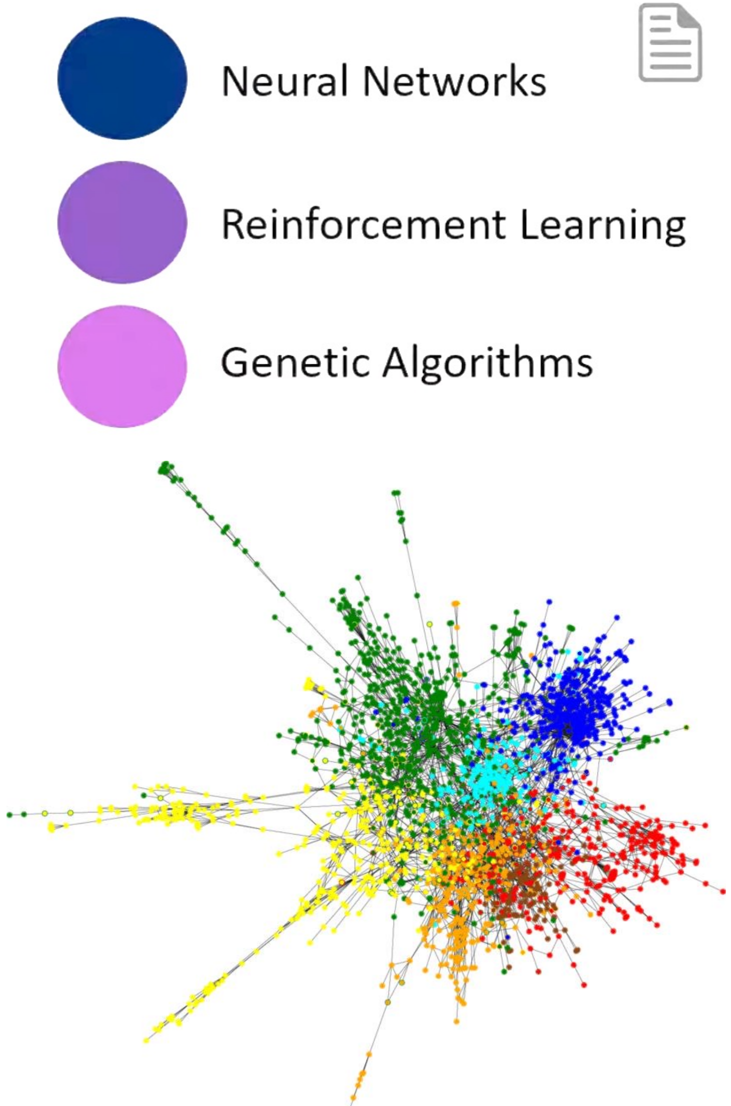
Cora Citation Dataset



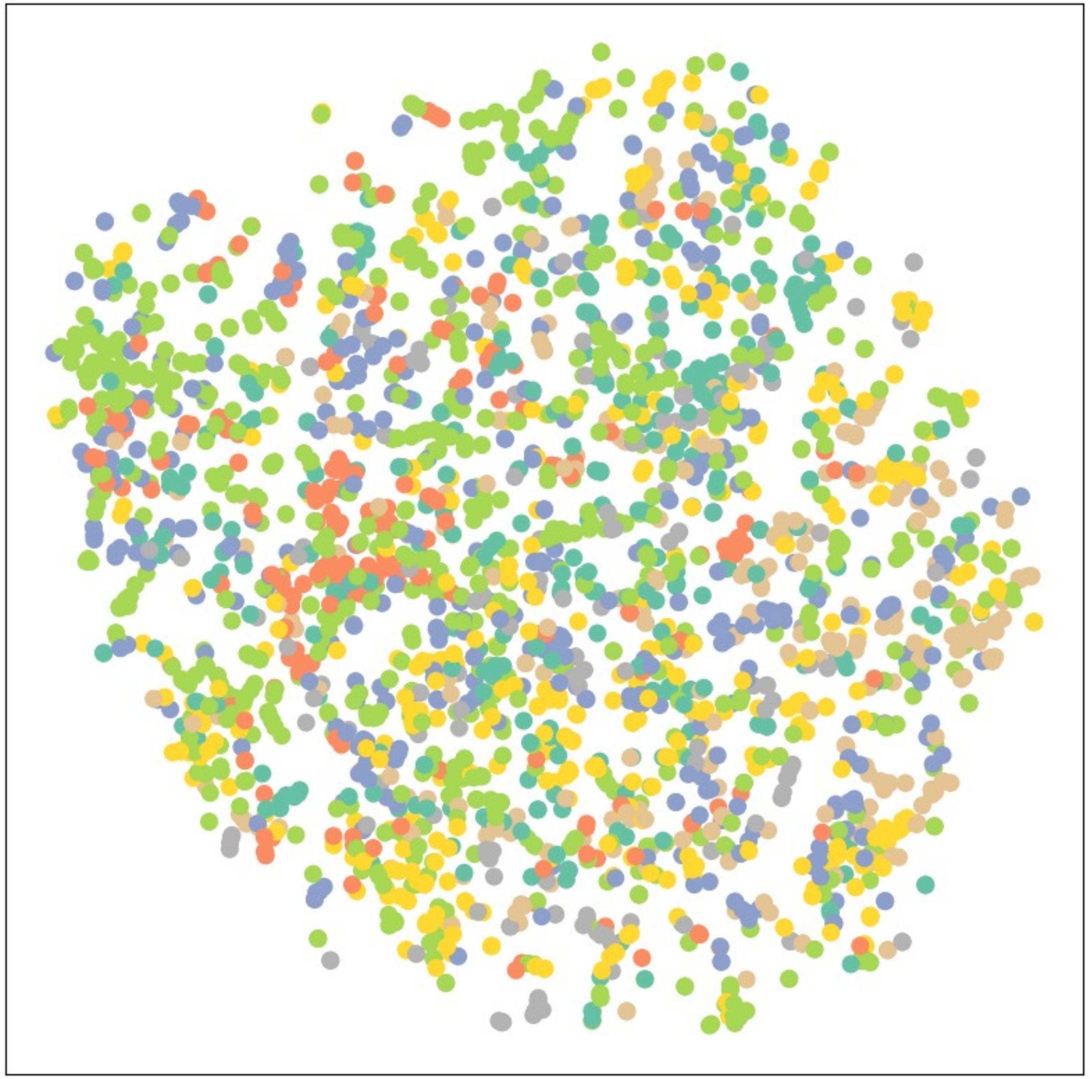


Cora Citation Dataset





Visualize the node embeddings of the untrained GCN network



```
model = GCN(hidden_channels=16)
model.eval()

out = model(data.x, data.edge_index)
visualize(out, color=data.y)
```

GNN: Node Classification

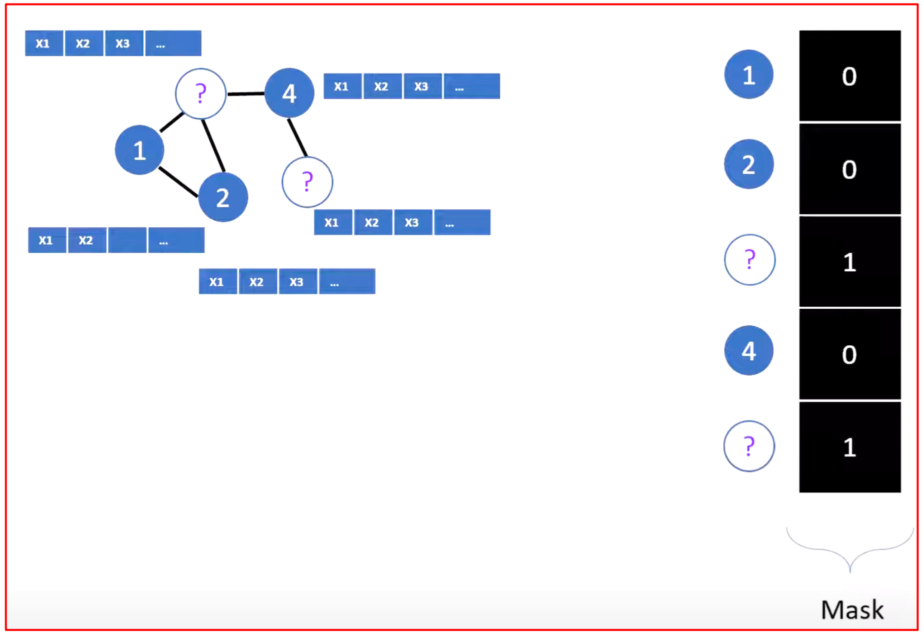
Install Pytorch Geometric

```
# Check CUDA Version
!python -c "import torch; print(torch.version.cuda)"

# Add this in a Google Colab cell to install the correct version of Pytorch Geometric.
import torch

def format_pytorch_version(version):
  return version.split('+')[0]

TORCH_version = torch.__version__
TORCH = format_pytorch_version(TORCH_version)

def format_cuda_version(version):
  return 'cu' + version.replace('.', '')

CUDA_version = torch.version.cuda
CUDA = format_cuda_version(CUDA_version)

!pip install torch-scatter     -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-sparse      -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-cluster     -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-spline-conv -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-geometric
```

GNN: Node Classification

Load Cora Dataset

```python
from torch_geometric.datasets import Planetoid
from torch_geometric.transforms import NormalizeFeatures

dataset = Planetoid(root='data/Planetoid', name='Cora', transform=NormalizeFeatures())

# Get some basic info about the dataset
print(f'Number of graphs: {len(dataset)}')
print(f'Number of features: {dataset.num_features}')
print(f'Number of classes: {dataset.num_classes}')
print(50*'=')

# There is only one graph in the dataset, use it as new data object
data = dataset[0]

# Gather some statistics about the graph.
print(data)
print(f'Number of nodes: {data.num_nodes}')
print(f'Number of edges: {data.num_edges}')
print(f'Number of training nodes: {data.train_mask.sum()}')
print(f'Training node label rate: {int(data.train_mask.sum()) / data.num_nodes:.2f}')
print(f'Is undirected: {data.is_undirected()}')
```




```
Number of graphs: 1
Number of features: 1433
Number of classes: 7
==================================================
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
Number of nodes: 2708
Number of edges: 10556
Number of training nodes: 140
Training node label rate: 0.05
Is undirected: True
```

GNN: Node Classification MLP

Define MLP

Here, we first reduce the 1433-dimensional feature vector to a low-dimensional embedding (hidden_channels=16), while the second linear layer acts as a classifier that should map each low-dimensional node embedding to one of the 7 classes.

```
import torch
from torch.nn import Linear
import torch.nn.functional as F


class MLP(torch.nn.Module):
    def __init__(self, hidden_channels):
        super().__init__()
        torch.manual_seed(12345)
        self.lin1 = Linear(dataset.num_features, hidden_channels)
        self.lin2 = Linear(hidden_channels, dataset.num_classes)

    def forward(self, x):
        x = self.lin1(x)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin2(x)
        return x

model = MLP(hidden_channels=16)
print(model)
```

```
test_acc = test()
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.5900
```

GNN: Node Classification MLP

How to Train

Here, we first reduce the 1433-dimensional feature vector to a low-dimensional embedding (hidden_channels=16), while the second linear layer acts as a classifier that should map each low-dimensional node embedding to one of the 7 classes.

```
from IPython.display import Javascript  # Restrict height of output cell.
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 300})'''))

model = MLP(hidden_channels=16)
criterion = torch.nn.CrossEntropyLoss()  # Define loss criterion.
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)  # Define optimizer.

def train():
      model.train()
      optimizer.zero_grad()  # Clear gradients.
      out = model(data.x)  # Perform a single forward pass.
      loss = criterion(out[data.train_mask], data.y[data.train_mask])  # Compute the loss solely based on the
      loss.backward()  # Derive gradients.
      optimizer.step()  # Update parameters based on gradients.
      return loss

def test():
      model.eval()
      out = model(data.x)
      pred = out.argmax(dim=1)  # Use the class with highest probability.
      test_correct = pred[data.test_mask] == data.y[data.test_mask]  # Check against ground-truth labels.
      test_acc = int(test_correct.sum()) / int(data.test_mask.sum())  # Derive ratio of correct predictions.
      return test_acc
```

```
test_acc = test()
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.5900
```

GNN: Node Classification GCN

Define GCN

```
class GCN(torch.nn.Module):
    def __init__(self, hidden_channels):
        super().__init__()
        torch.manual_seed(1234567)
        self.conv1 = GCNConv(dataset.num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, dataset.num_classes)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return x

model = GCN(hidden_channels=16)
print(model)
```

```
GCN(
  (conv1): GCNConv(1433, 16)
  (conv2): GCNConv(16, 7)
)
```

we will use on of the most simple GNN operators, the **GCN layer** (Kipf et al. (2017)), which is defined as

$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$

GNN: Node Classification GCN

Define GCN

Dropout is only applied in the training step, but not for predictions
We have 2 Message Passing Layers and one Linear output layer
We use the softmax function for the classification problem
The output of the model are 7 probabilities, one for each class

```
class GCN(torch.nn.Module):
    def __init__(self, hidden_channels):
        super(GCN, self).__init__()
        torch.manual_seed(42)

        # Initialize the layers
        self.conv1 = GCNConv(dataset.num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.out = Linear(hidden_channels, dataset.num_classes)

    def forward(self, x, edge_index):
        # First Message Passing Layer (Transformation)
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)

        # Second Message Passing Layer
        x = self.conv2(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)

        # Output layer
        x = F.softmax(self.out(x), dim=1)
        return x
```

we will use on of the most simple GNN operators, the **GCN layer** (Kipf et al. (2017)), which is defined as

$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$

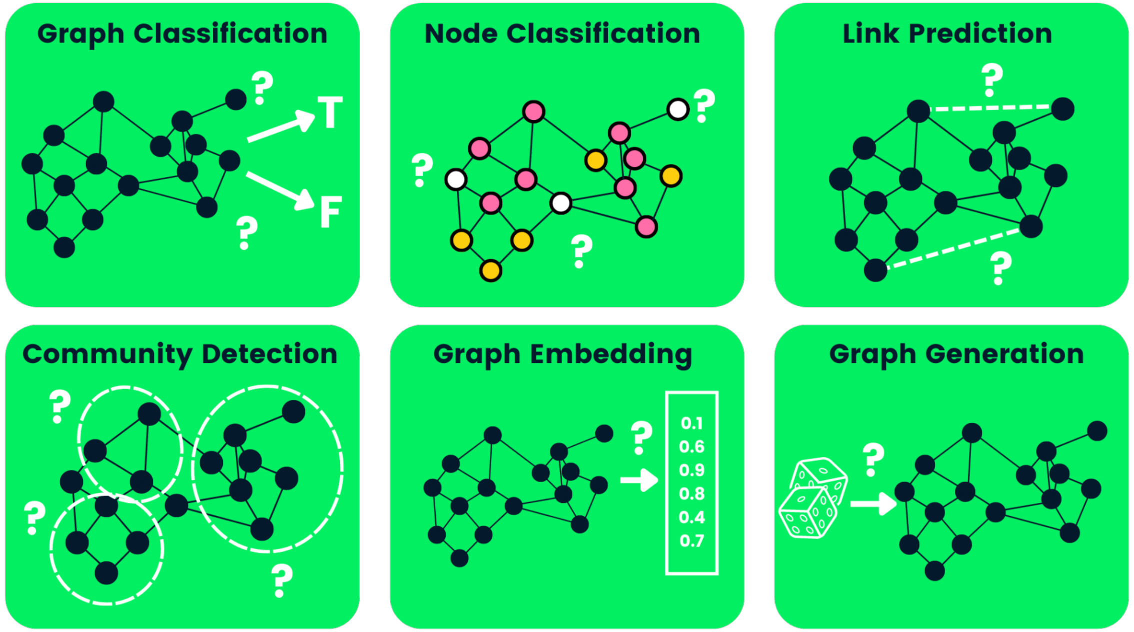
GNN: Node Classification GCN

```python
from IPython.display import Javascript  # Restrict height of output cell.
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 300})'''))

model = GCN(hidden_channels=16)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
criterion = torch.nn.CrossEntropyLoss()

def train():
      model.train()
      optimizer.zero_grad()  # Clear gradients.
      out = model(data.x, data.edge_index)  # Perform a single forward pass.
      loss = criterion(out[data.train_mask], data.y[data.train_mask])  # Compute the loss solely based on the
      loss.backward()  # Derive gradients.
      optimizer.step()  # Update parameters based on gradients.
      return loss

def test():
      model.eval()
      out = model(data.x, data.edge_index)
      pred = out.argmax(dim=1)  # Use the class with highest probability.
      test_correct = pred[data.test_mask] == data.y[data.test_mask]  # Check against ground-truth labels.
      test_acc = int(test_correct.sum()) / int(data.test_mask.sum())  # Derive ratio of correct predictions.
      return test_acc


for epoch in range(1, 101):
    loss = train()
    print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}')
```

```python
test_acc = test()
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.8150
```

Summary

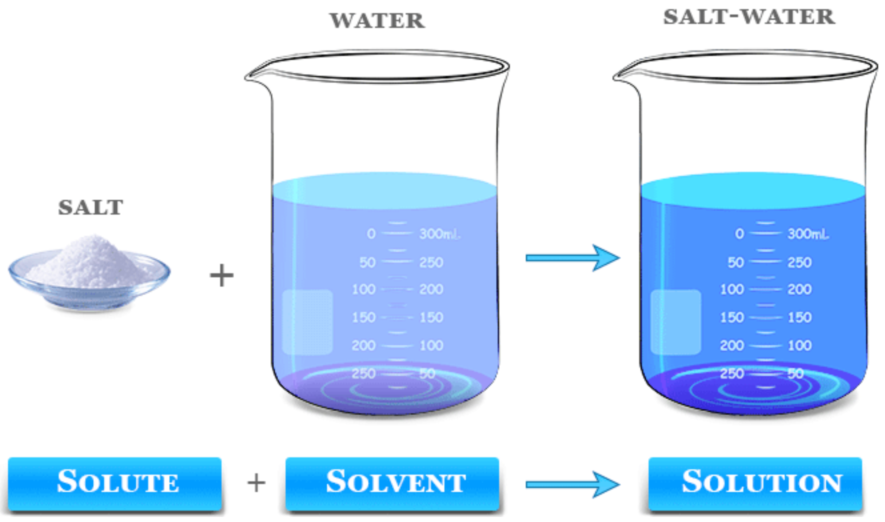


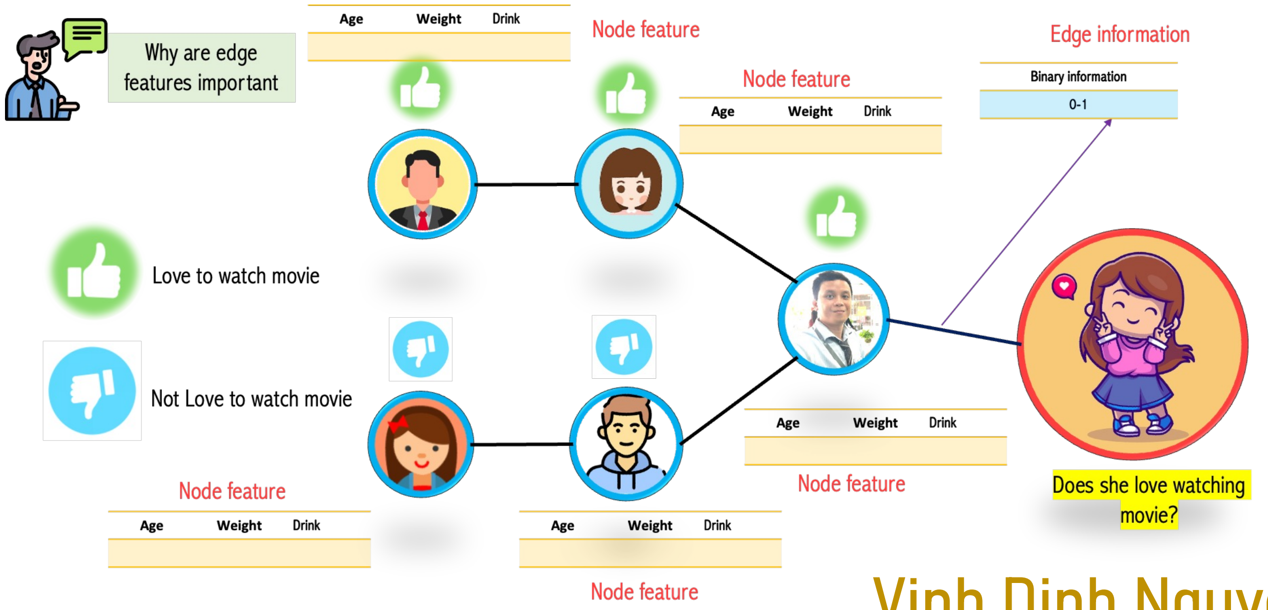
1. What is Graph Data Around Us
2. Understand Graph Neural Network
3. Understand Graph Convolutional Neural Network
4. Node Classification with Cora Citation Dataset


Thank you

Advanced Graph Neural Network

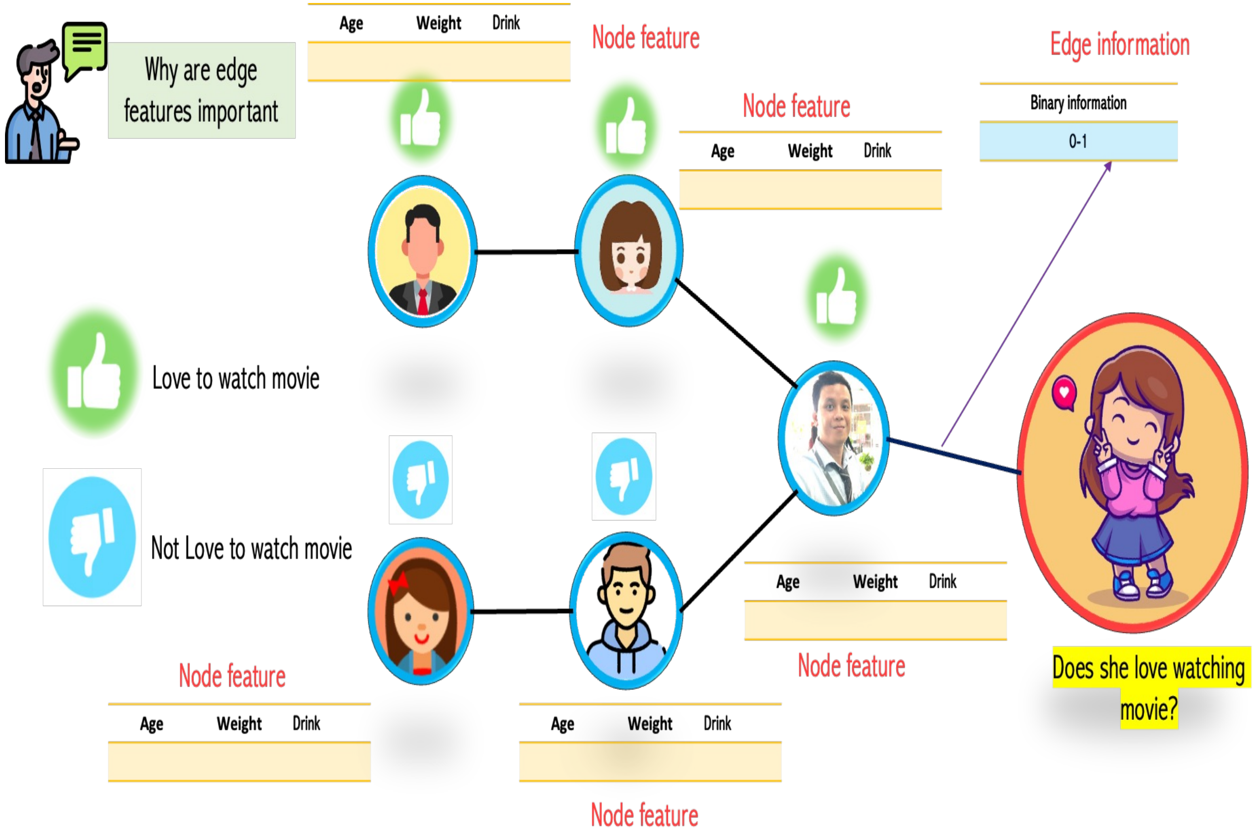
(GCN, Graph Relational, Attention & Level-Prediction)





Vinh Dinh Nguyen
PhD in Computer Science

Objective



1. How to integrate edge feature to GNN
2. Edge Weight in GNN
3. Relational GNN
4. Multidimensional Edge Feature
4. Attention in GNN
4. Graph-level prediction: Example and Code

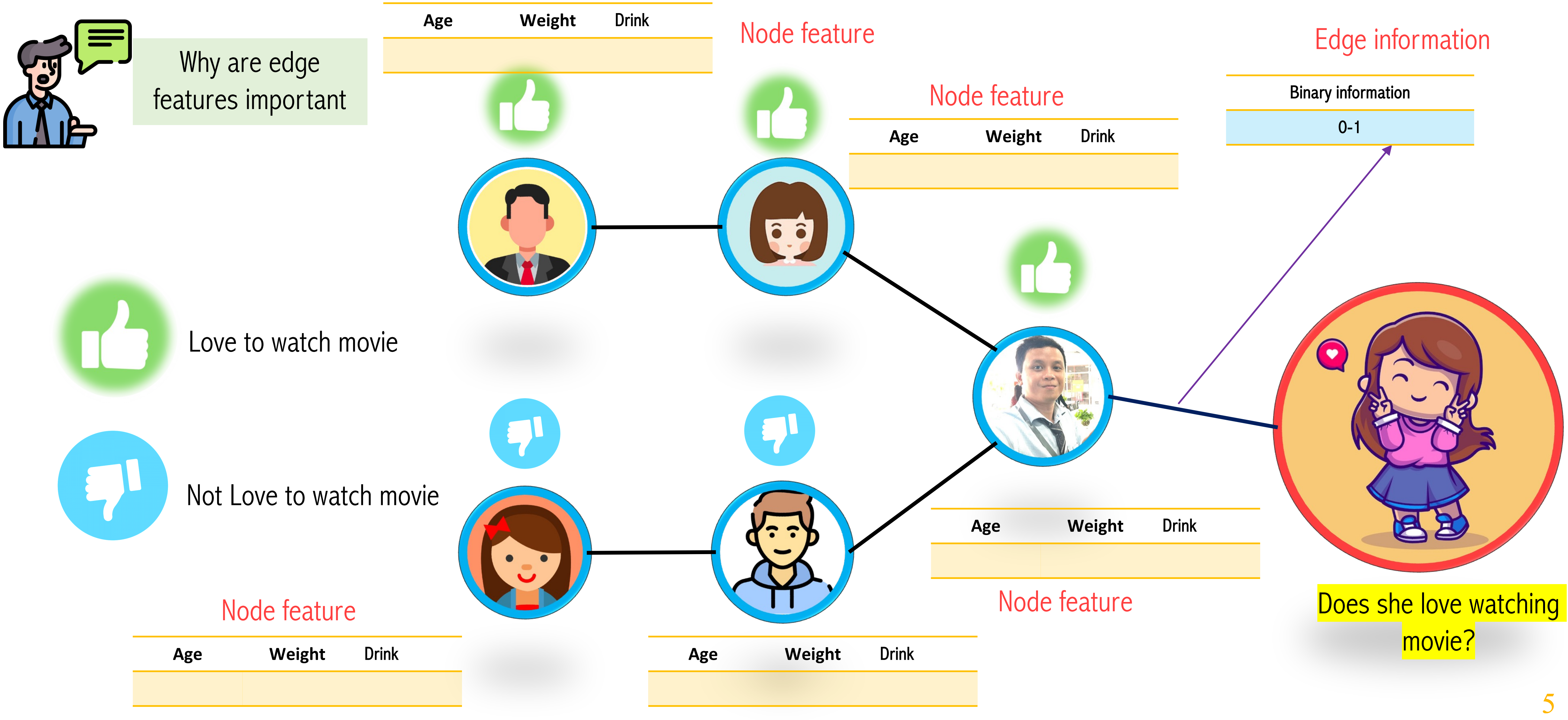
Outline

- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

Outline

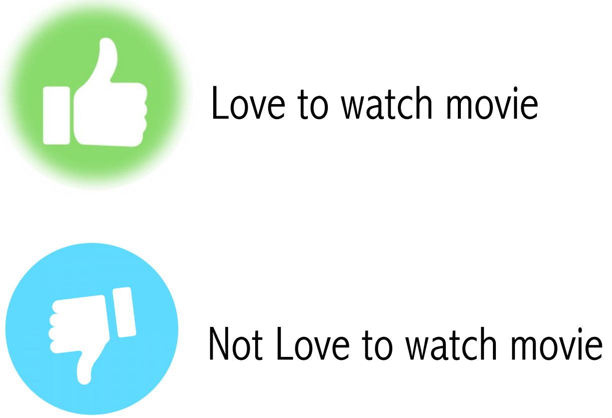
- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

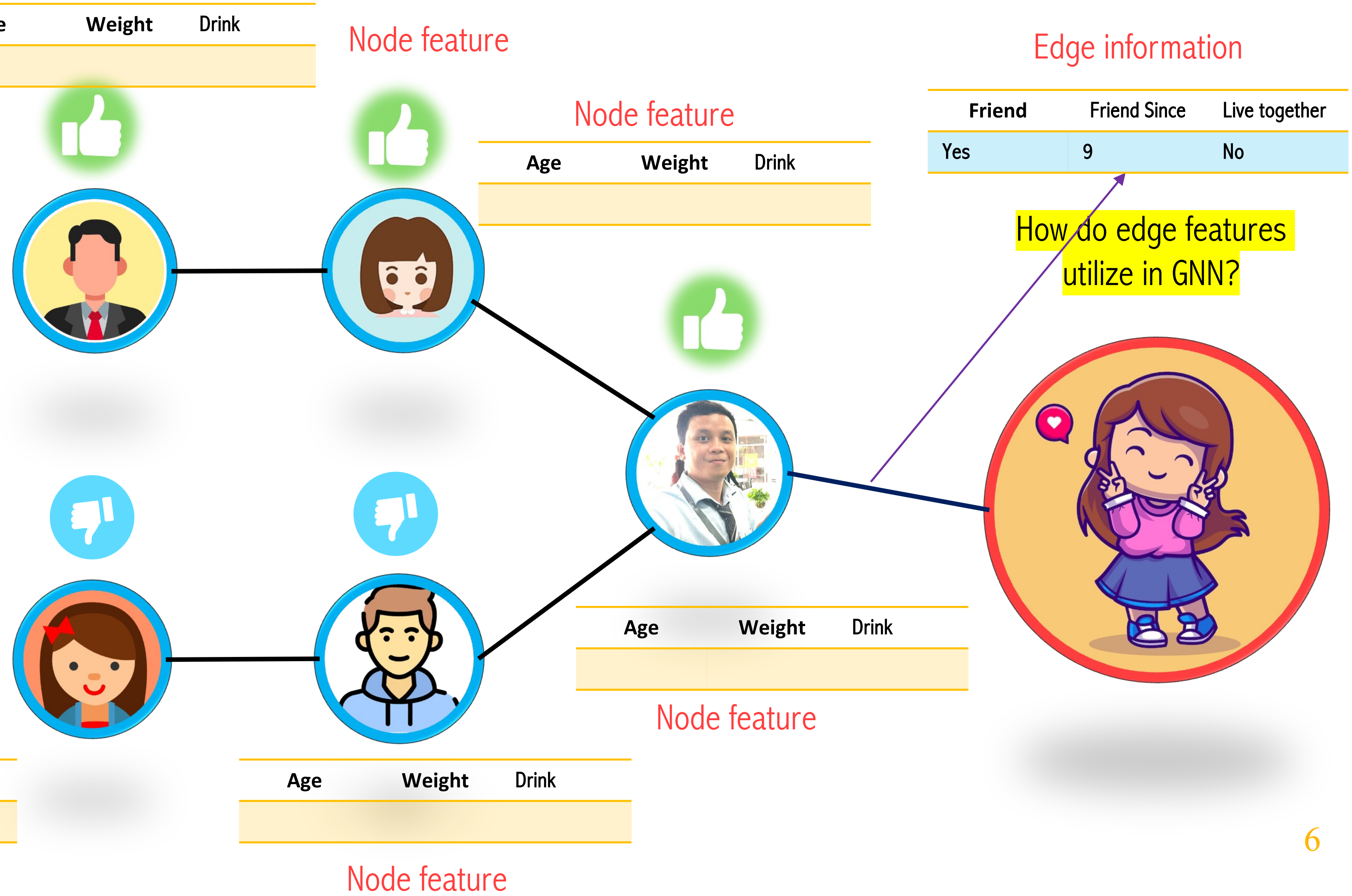
Edge Feature in GNN: Last But Not Least



Edge Feature in GNN: Last But Not Least

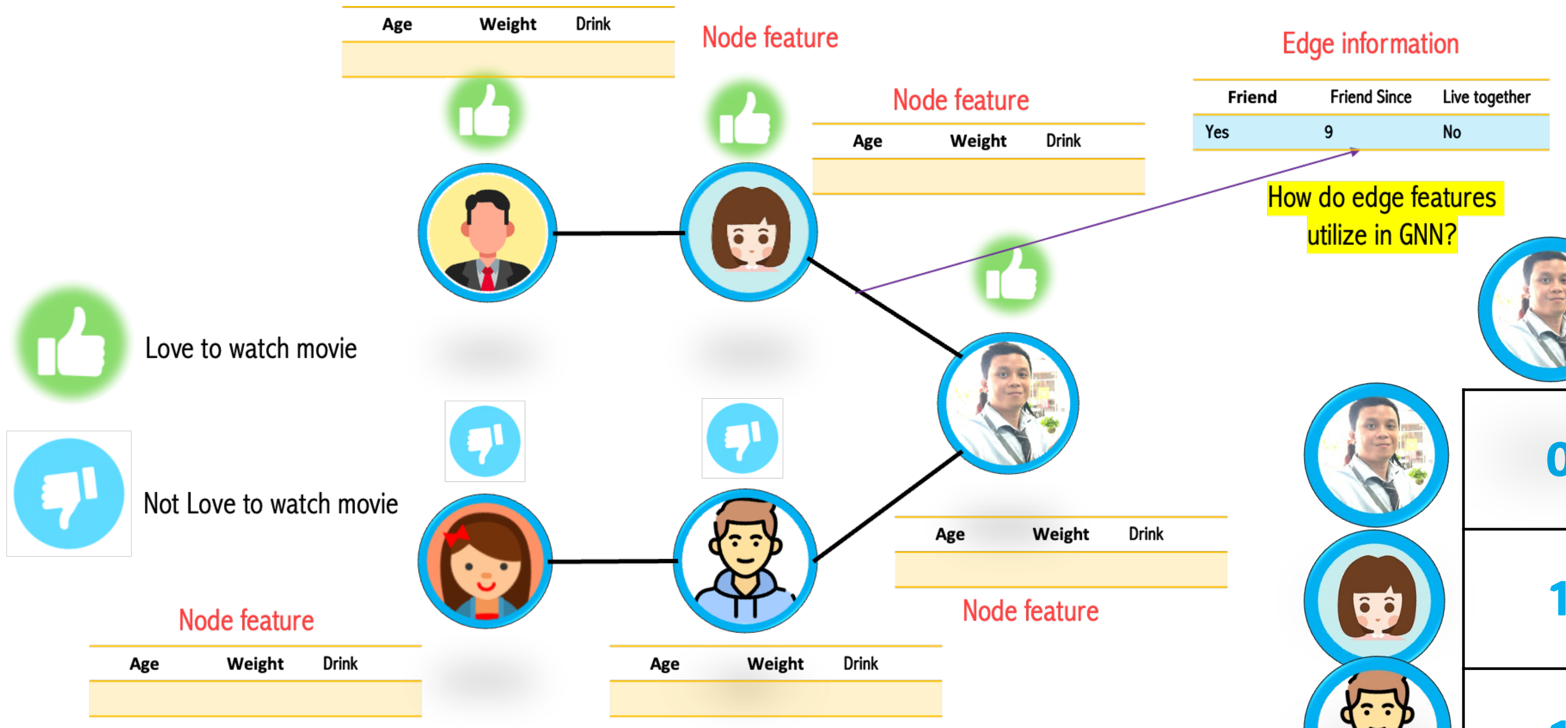
Why are edge features important





AI VIETNAM
All-in-One Course

Edge Feature in GNN

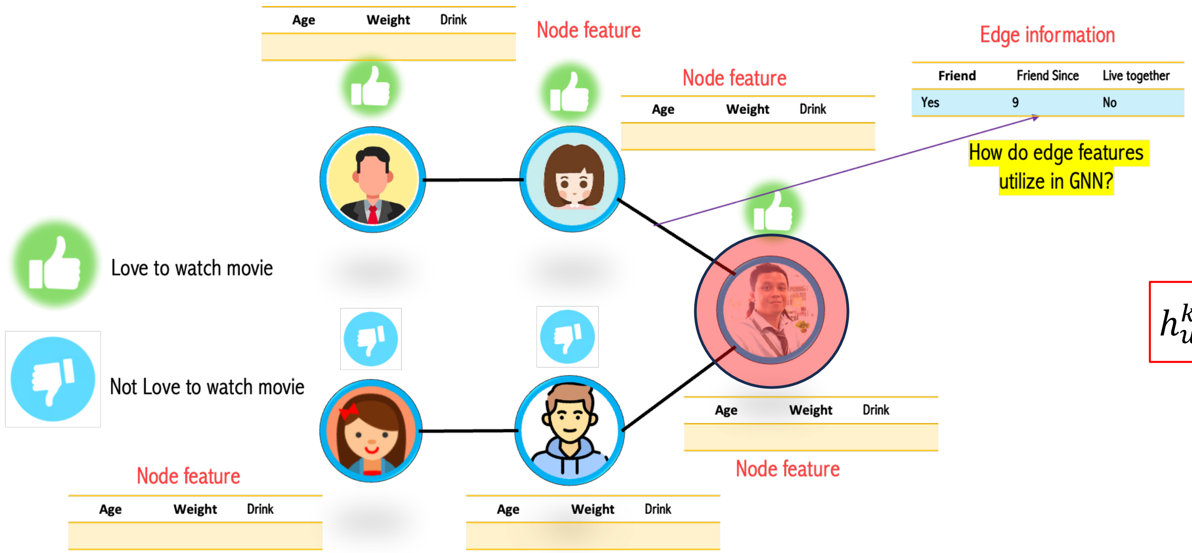


Adjacency Matrix

0	1	1	0	0
1	0	0	1	0
1	0	0	0	1
0	1	0	0	0
0	0	1	0	0

How to use Edge Feature:
Hot Research Topic

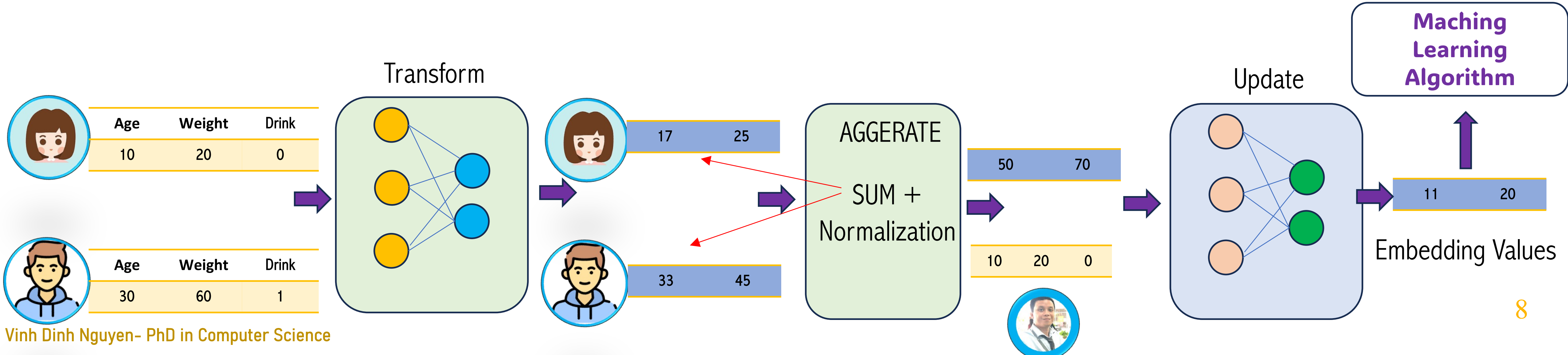
Node Embedding for Vinh Nguyen



Where is the edge information used in this process?

$$h_u^{k+1} = UPDATE^{(k)}\left(h_u^{k+1}, AGGREGATE^{(k)}\left(\{h_V^k\}, \forall v \in \mathrm{N}(u)\right)\right)$$

$$h_u^{k+1} = AGGREGATE^{(k)}\left(\{h_V^k\}, \forall v \in \mathrm{N}(u)\right)$$



Node Embedding for Vinh Nguyen

Where is the edge information used in this process?

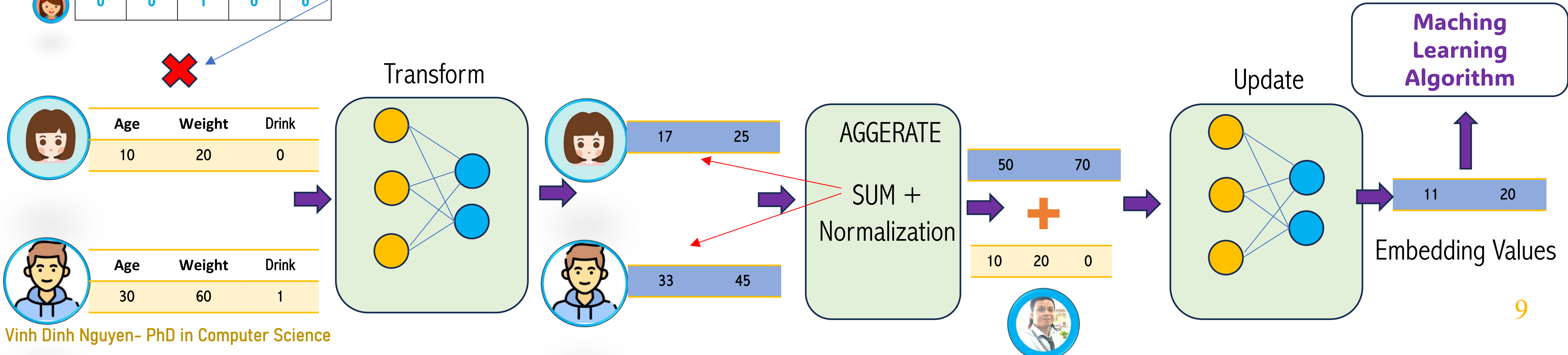
Binary approach: use directly when we select the node

Adjacency Matrix

0	1	1	0	0
1	0	0	1	0
1	0	0	0	1
0	1	0	0	0
0	0	1	0	0

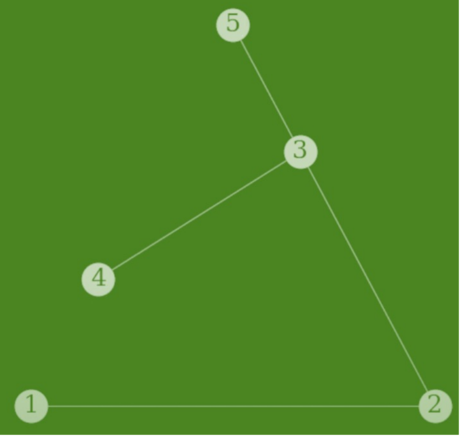
$$h_{\text{vinh}}^{k+1} = UPDATE^{(k)}\left(h_{\text{vinh}}^{k}, \text{AGGREGATE}_{j\in N(vinh)}\, TRANSFORM(h_j^k)\right)$$

$$h_{Vinh}^{k+1} = \text{AGGREGATE}_{j\in N(vinh)}\, TRANSFORM(h_j^k)$$



Example

$$
\begin{array}{c|ccccc|}
 & Node_1 & Node_2 & Node_3 & Node_4 & Node_5 \\
\hline
Node_1 & 0 & 1 & 0 & 0 & 0 \\
Node_2 & 1 & 0 & 1 & 0 & 0 \\
Node_3 & 0 & 1 & 0 & 1 & 1 \\
Node_4 & 0 & 0 & 1 & 0 & 0 \\
Node_5 & 0 & 0 & 1 & 0 & 0
\end{array}
$$



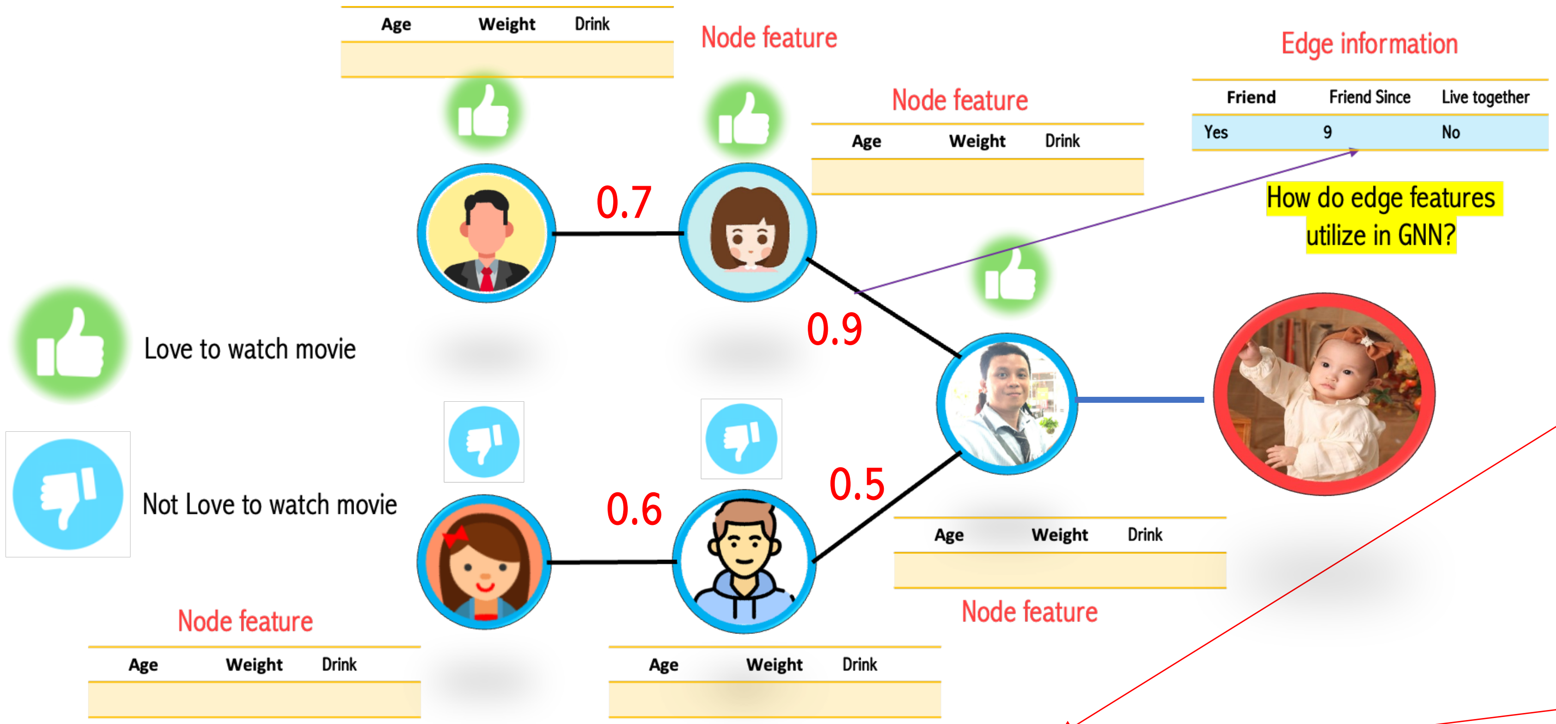
The ‘0’ of the adjacency matrix cancel the contribution of all connected nodes, resulting a sum of just the connected nodes.

$$
\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}
\begin{bmatrix} (0)(1)+(1)(2)+(0)(3)+(0)(4)+(0)(5) \\ (1)(1)+(0)(2)+(1)(3)+(0)(4)+(0)(5) \\ (0)(1)+(1)(2)+(0)(3)+(1)(4)+(1)(5) \\ (0)(1)+(0)(2)+(1)(3)+(0)(4)+(0)(5) \\ (0)(1)+(0)(2)+(1)(3)+(0)(4)+(0)(5) \end{bmatrix}
=
\begin{bmatrix} 2 \\ 4 \\ 11 \\ 3 \\ 3 \end{bmatrix}
$$

Outline

- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

Edge Weight: Common Approach



0	1	1	0	0
1	0	0	1	0
1	0	0	0	1
0	1	0	0	0
0	0	1	0	0

Adjacency Matrix: Describe the node connection

0	0.9	0.5	0	0
0.9	0	0	0.7	0
0.5	0	0	0	0.6
0	0.7	0	0	0
0	0	0.6	0	0

Weighted Matrix: Describe the "happy" information

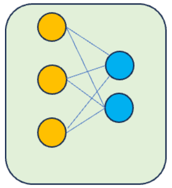
Adjacency matrix: $\tilde{A} = A + I_N$

Activation Function

New Embedding: $H^{(l+1)} = \sigma\left(\boxed{\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}}H^{(l)}W^{(l)}\right)$

Normalization $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Node feature

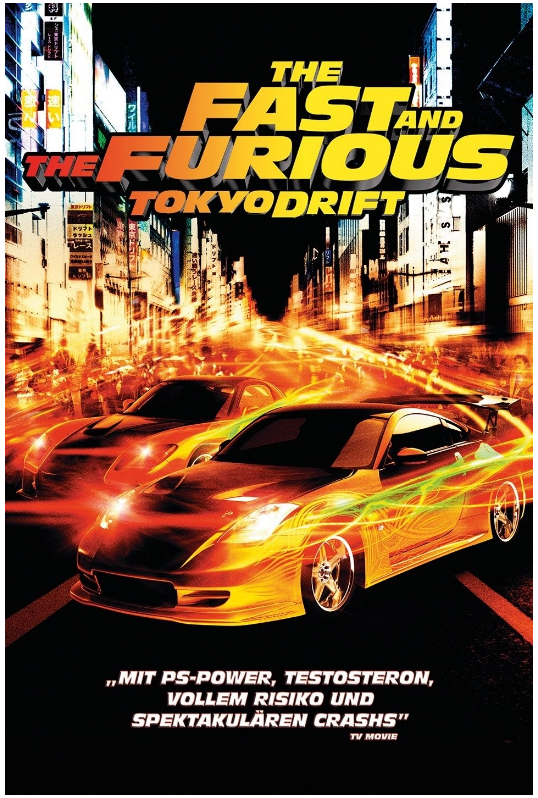


layer-specific trainable weight matrix

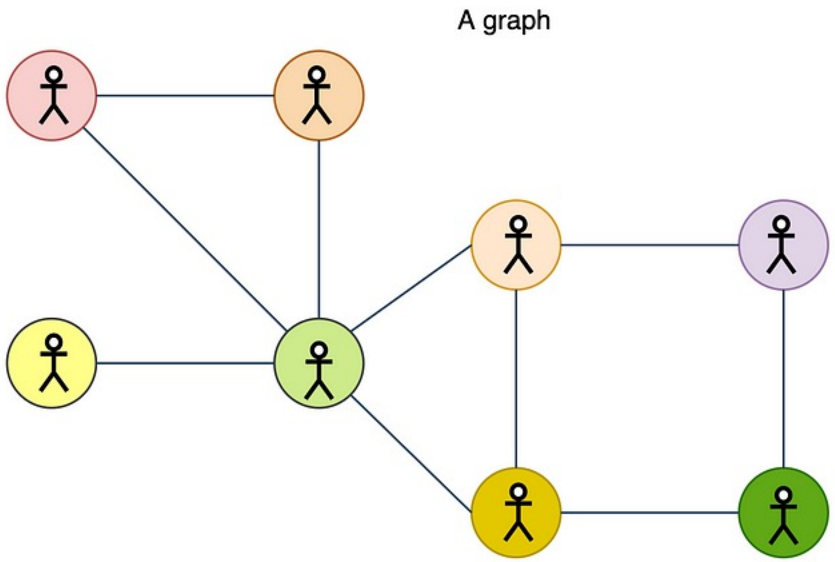
GNN: Review

You know, who you choose to be around you, let's you know who you are.

The Fast and the Furious: Tokyo Drift.

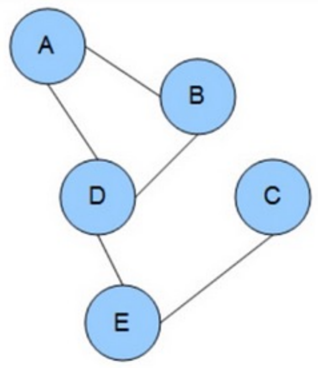


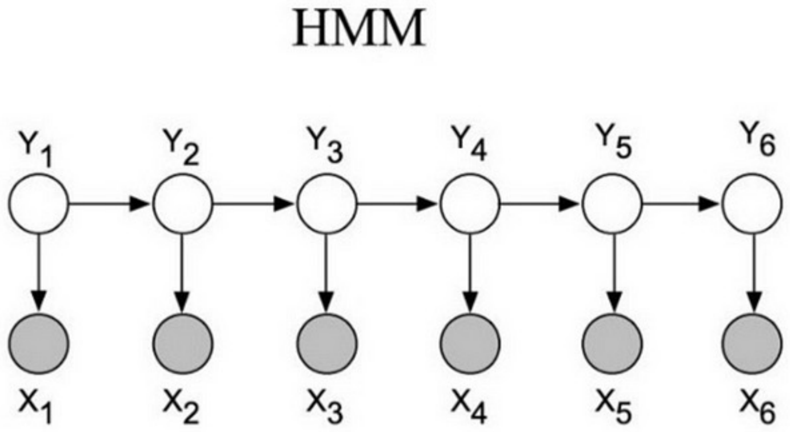
GNN: Review



In social networks, friend connections can be realized by a social graph.

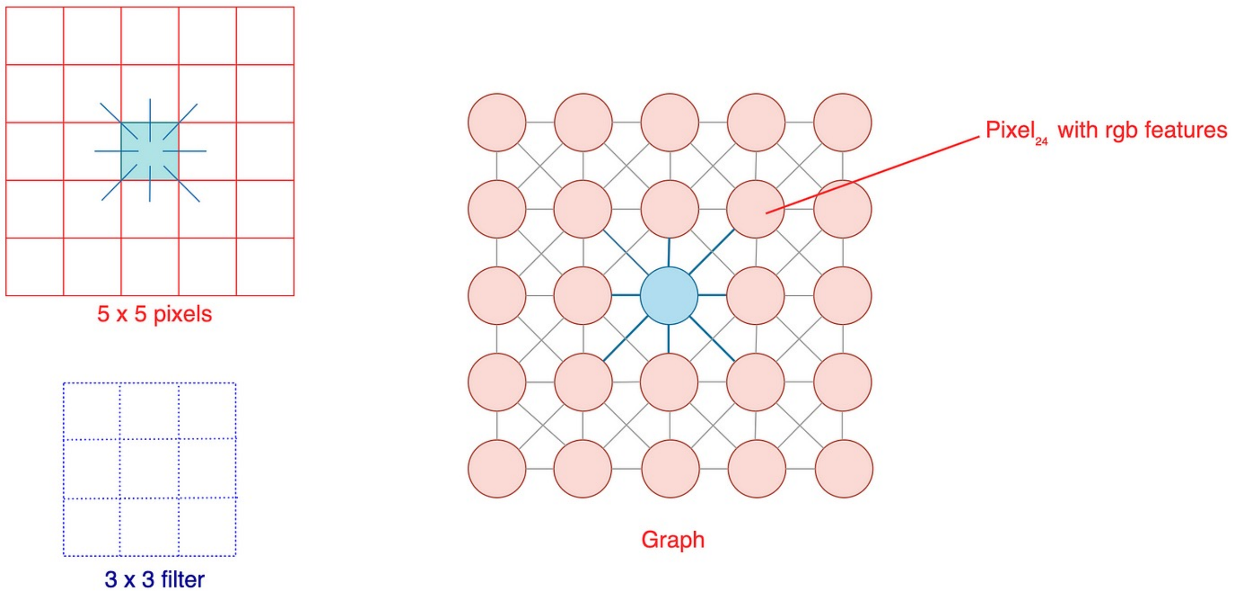
GNN: Review





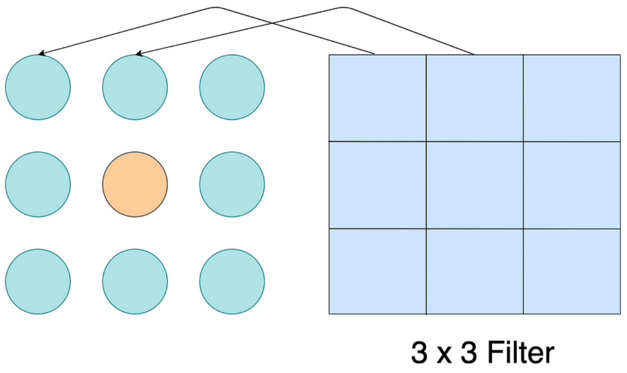
In speech recognition, the phoneme Y_i and the acoustic model x_i form an HMM (a graph for speech recognition).

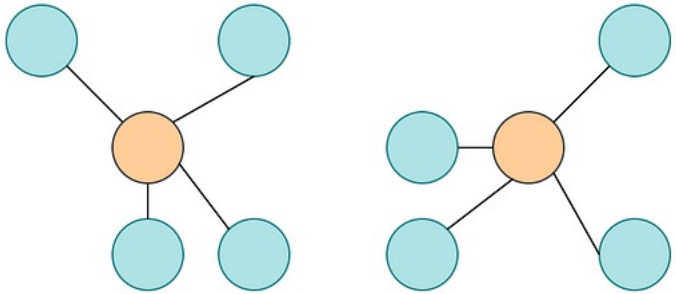
GNN: Review



Even on CNN, an input image can be modeled as a graph. For example, the graph for a 5 × 5 image. Each node represents a pixel and for the case of a 3 × 3 filter, every node is connected to its eight immediate neighbors.

GNN: Review

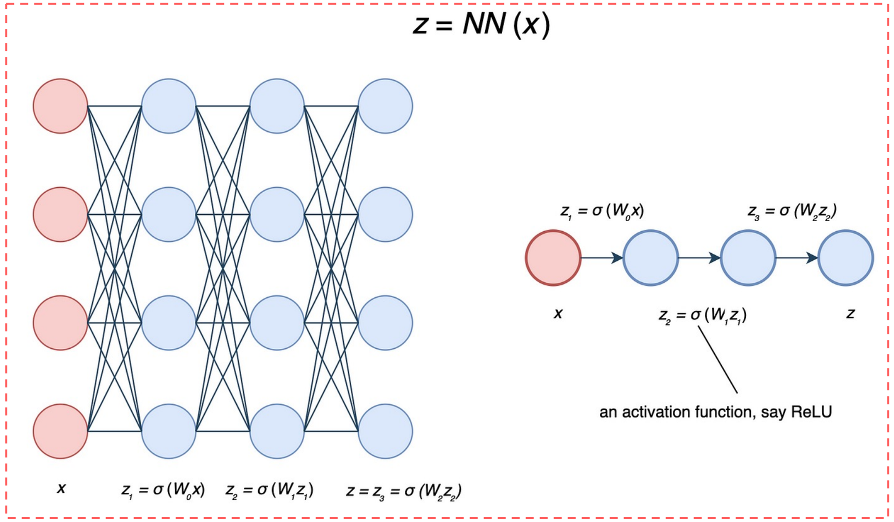


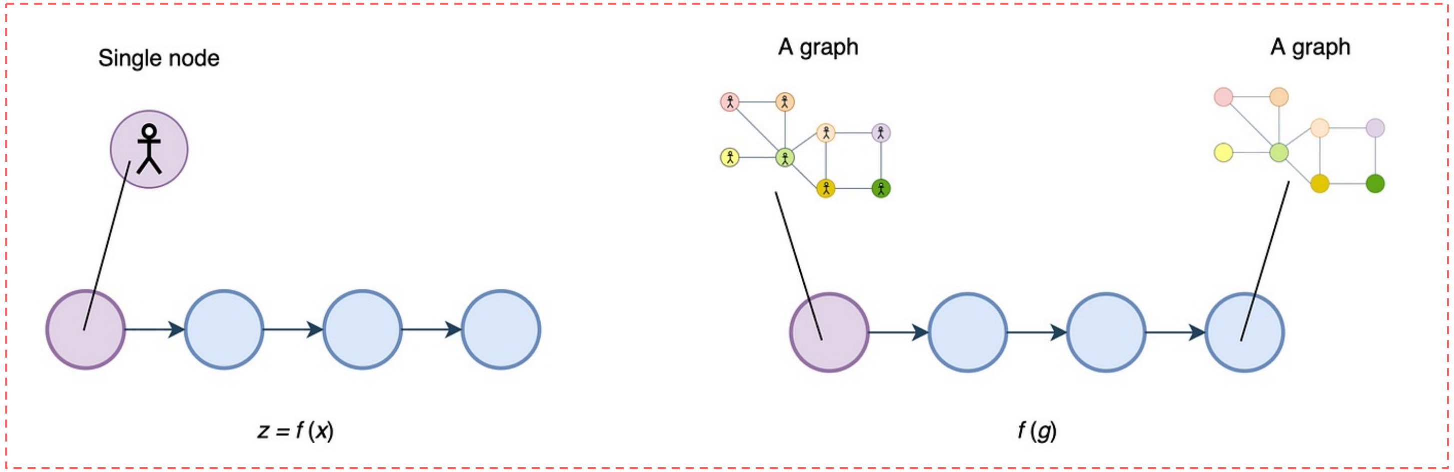


In particular, when the relationships between neighboring nodes are irregular and high dimensional, we need to define them explicitly in order to solve them efficiently. In CNN, we work in a Euclidean space. How weights are associated with the input features (pixels) is well defined.

But this is not the case for a graph. For example, the graphs above are the same even though it looks different spatially.

GNN: Review

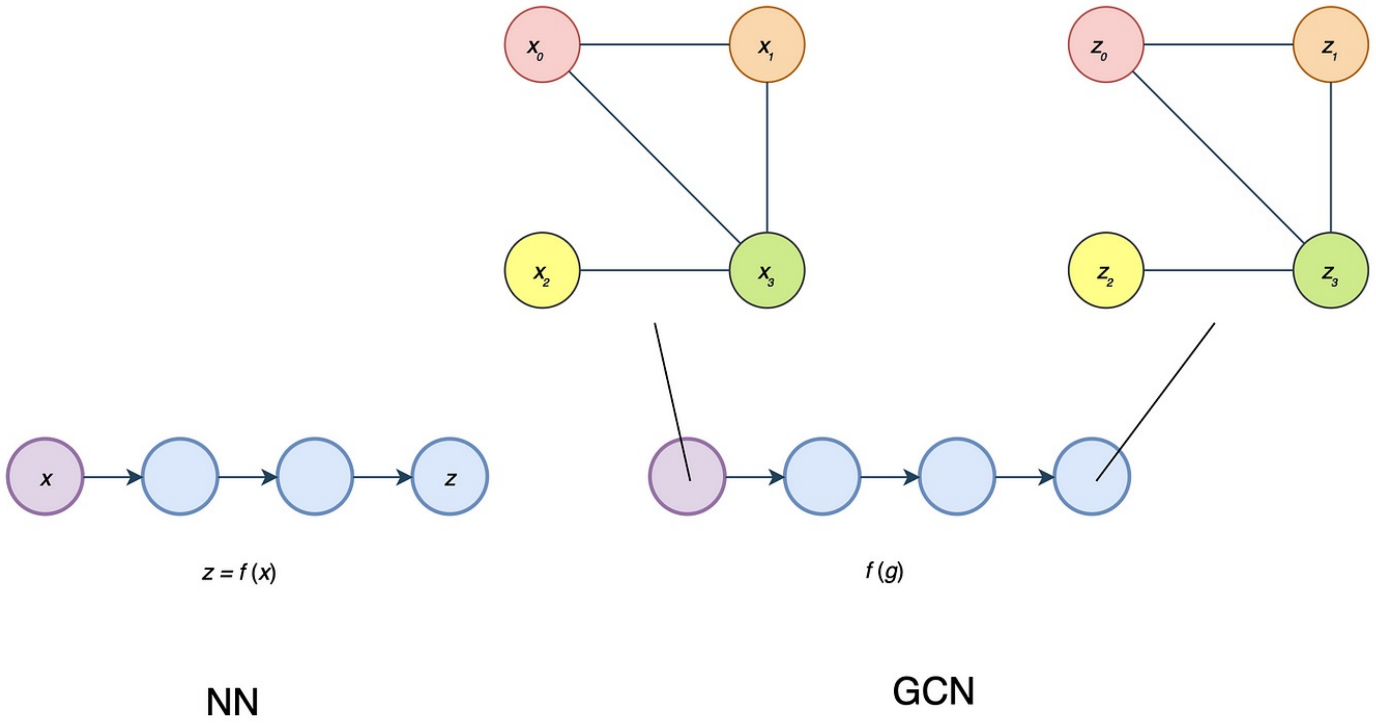




In general, neural networks (NNs) takes an input x to predict z.

This leads us to the challenge of how a NN can process a graph directly.

GNN: Review

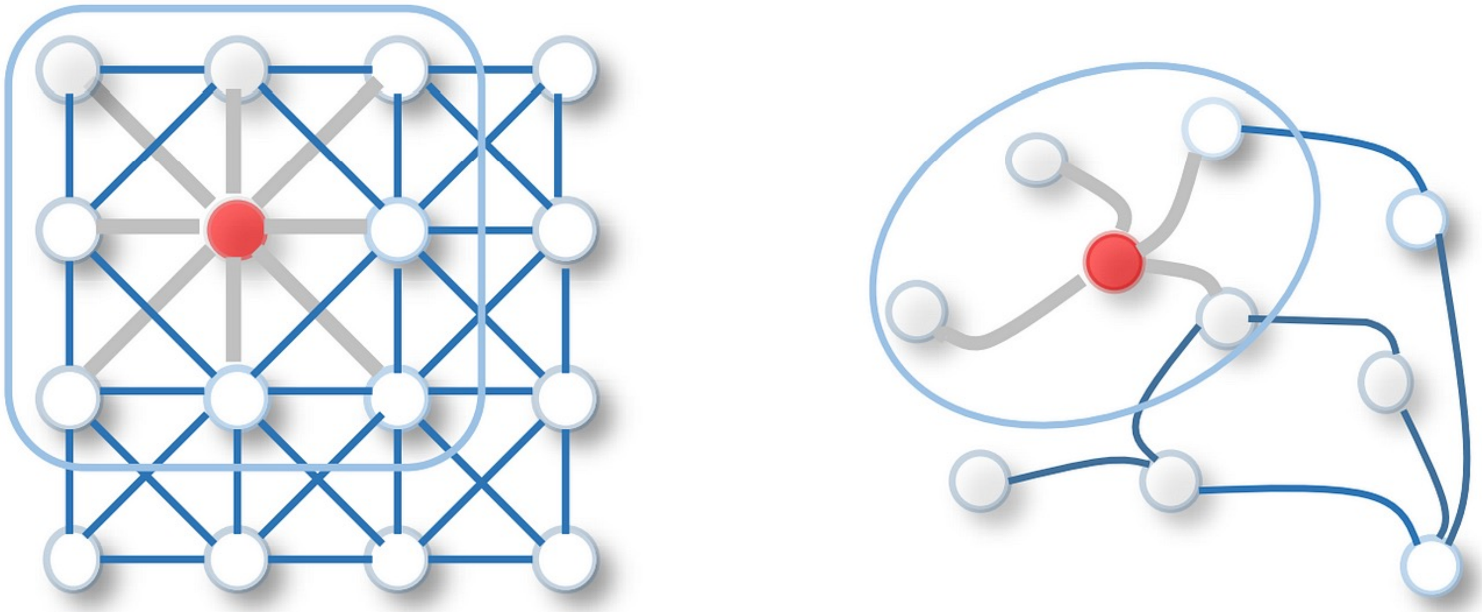


In GCN (Graph Convolutional Network), the input to the NN will be a graph. Also, instead of inferring a single z, it infers the value z_i for each node i in the graph. And to make predictions for Z_i, GCN utilizes both X_i and its neighboring nodes in the calculation.

GNN: Review

Graph Convolutional Networks (GCN)

The general idea of GCN is to apply convolution over a graph. Instead of having a 2-D array as input, GCN takes a graph as an input.



GNN: Review

Graph Convolutional Networks (GCN)

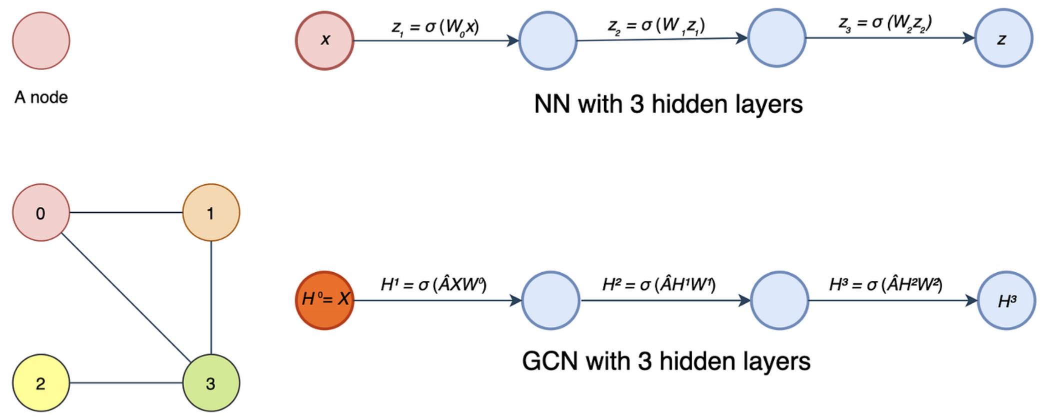
That comes to the output of the hidden layer to be $\sigma(\hat{A}H^iW^i)$. If we ignore W for a second, for each node in a hidden layer, $\hat{A}H^i$ sums up features on each node with its neighbors.

Diminishing or exploding problem in a NN

In specific, GCN wants $\hat{A}$ to be normalized to maintain the scale of the output feature vectors

One possibility is to multiple $\hat{A}$ with $\tilde{D}^{-1}$ where $\tilde{D}$ is the diagonal node degree matrix of $\hat{A}$ in measuring the degree of each node

σ is an activation function, like ReLU



$\hat{A}$

	0	1	2	3
0	1	1	0	1
1	1	1	0	1
2	0	0	1	1
3	1	1	1	1

Node 0 and 3 are connected

All diagonal elements are 1
(All nodes are self-connected)

Mathematically, $\hat{A}$ equals $A + I$

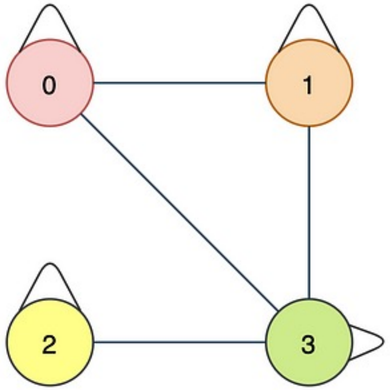
GNN: Review

$\hat{A}$

	0	1	2	3
0	1	1	0	1
1	1	1	0	1
2	0	0	1	1
3	1	1	1	1

$\hat{D}$

	0	1	2	3
0	4	0	0	0
1	0	4	0	0
2	0	0	3	0
3	0	0	0	5



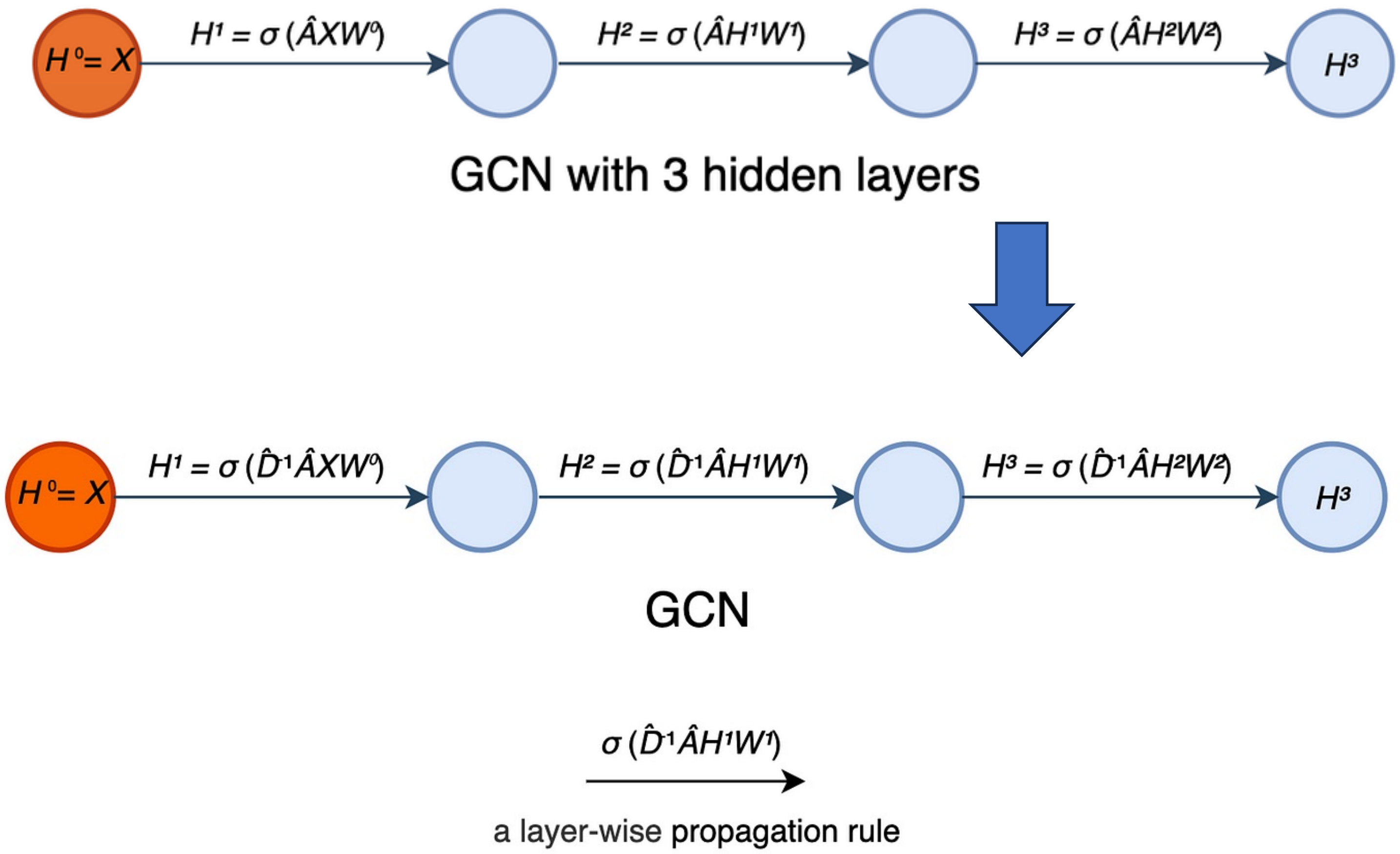
For an undirected graph, the degree of a node is counted as the number of times an edge terminates at that node. So a self-loop will count twice. In our example, node 0 has 2 edges connecting to its neighbors plus a self-loop. Its degree equals 4 (i.e. 2 + 2). For node 3, its degree equals 5 (3 + 2).

$\hat{D}^{-1}$

	0	1	2	3
0	1/4	0	0	0
1	0	1/4	0	0
2	0	0	1/3	0
3	0	0	0	1/5

GNN: Review

Graph Convolutional Networks (GCN)



The diagram summarizes the model discussed so far. In this example, it has 3 hidden layers and for each hidden layer, it computes its output as $\sigma(\tilde{D}^{-1}\hat{A}H^iW^i)$. The equation used to compute a hidden layer output from the last layer output is called the **propagation rule**.

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

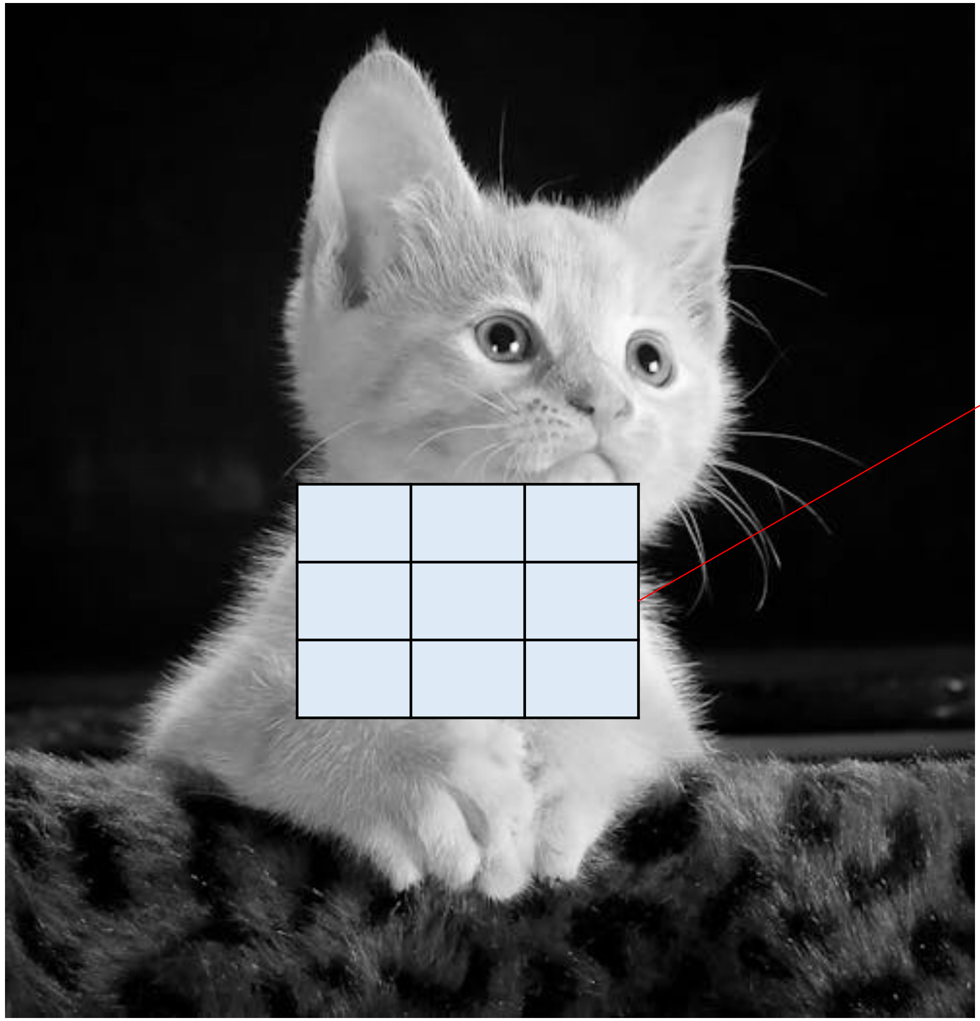
$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where

$$\tilde{A} = A + I_N$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

From CNN to GNN



Convolutional Operation

ROI

48	109	57
17	52	126
13	13	64

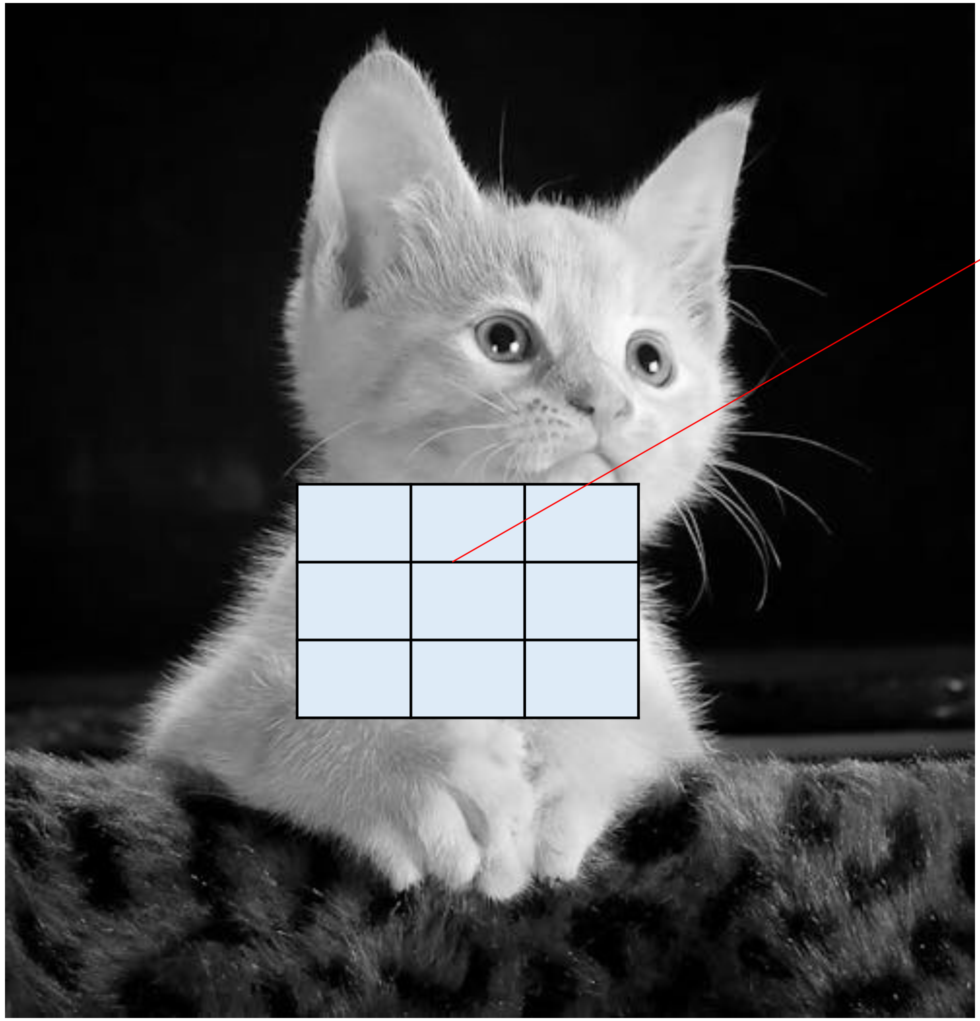
Kernel

$$\begin{bmatrix} 0.12 & 0.58 & 1.17 \\ -0.44 & 3.11 & -0.8 \\ 5.11 & -0.31 & 4.17 \end{bmatrix}$$

0.12×48 + 0.58×109 + 1.17×57 -0.44×17 + 3.11×52 -0.8×126 + 5.11×13 -0.31×13 + 4.17×64 = 635.5

	635.5	

From CNN to GNN

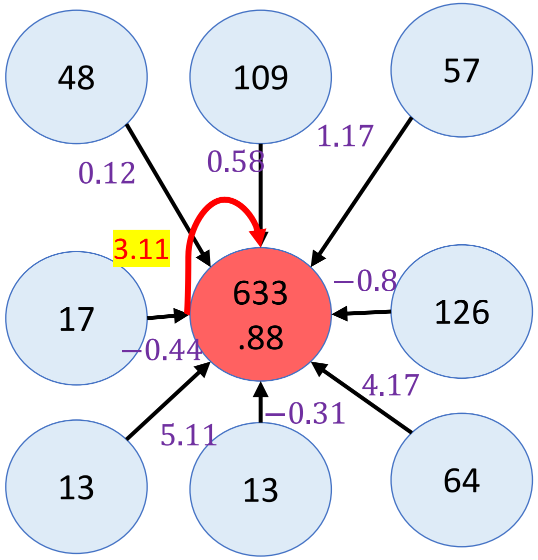


ROI

48	109	57
17	52	126
13	13	64

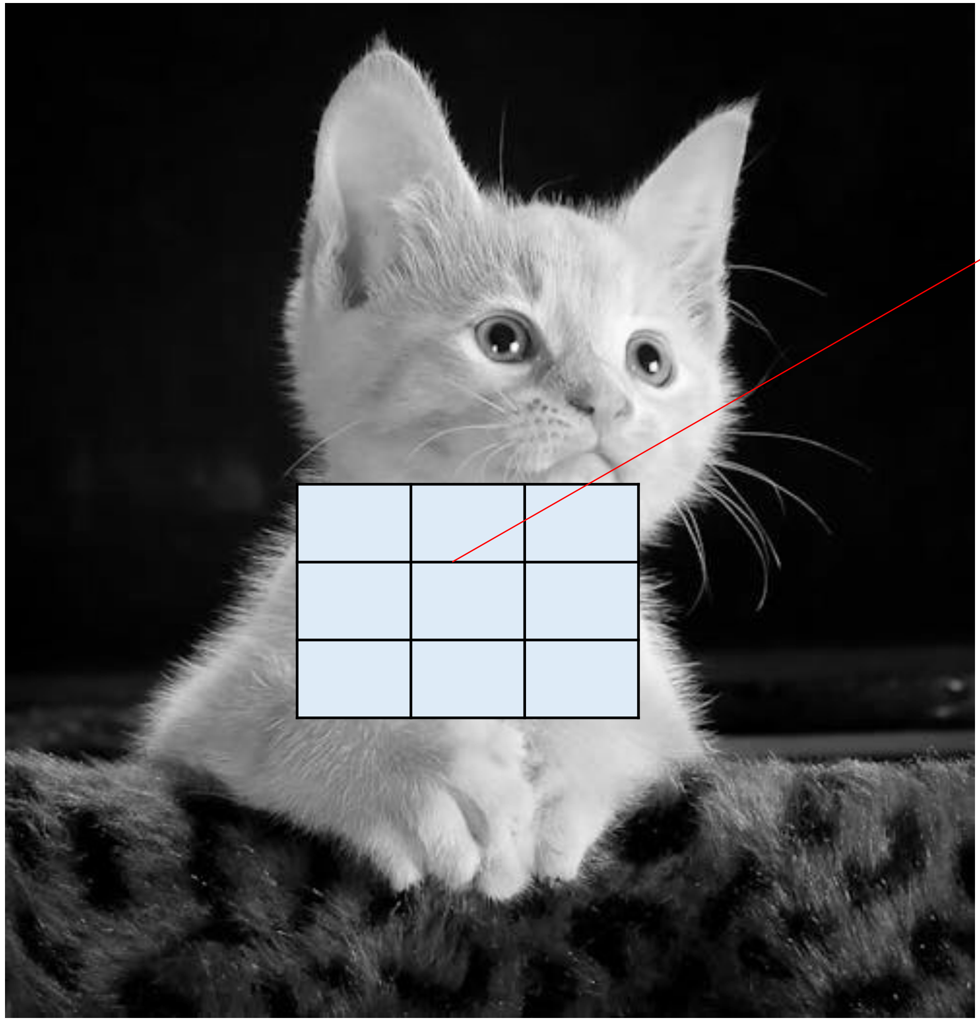
Kernel

$$\begin{bmatrix} 0.12 & 0.58 & 1.17 \\ -0.44 & 3.11 & -0.8 \\ 5.11 & -0.31 & 4.17 \end{bmatrix}$$



0.12×48 + 0.58×109 + 1.17×57 -0.44×17 - 0.8×126 + 5.11×13 -0.31×13 + 4.17×64 = 633.88

From CNN to GNN

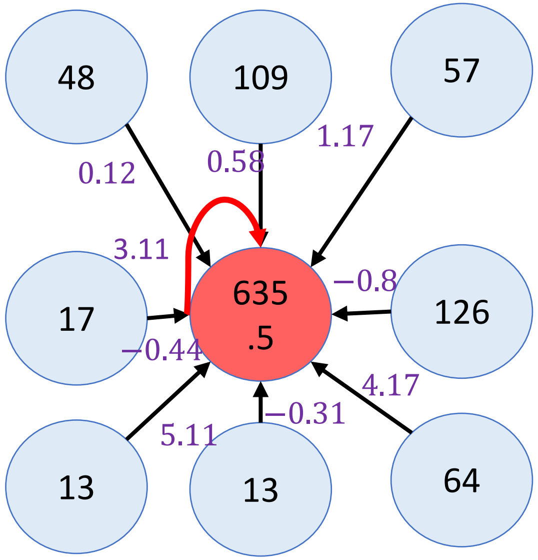


ROI

48	109	57
17	52	126
13	13	64

Kernel

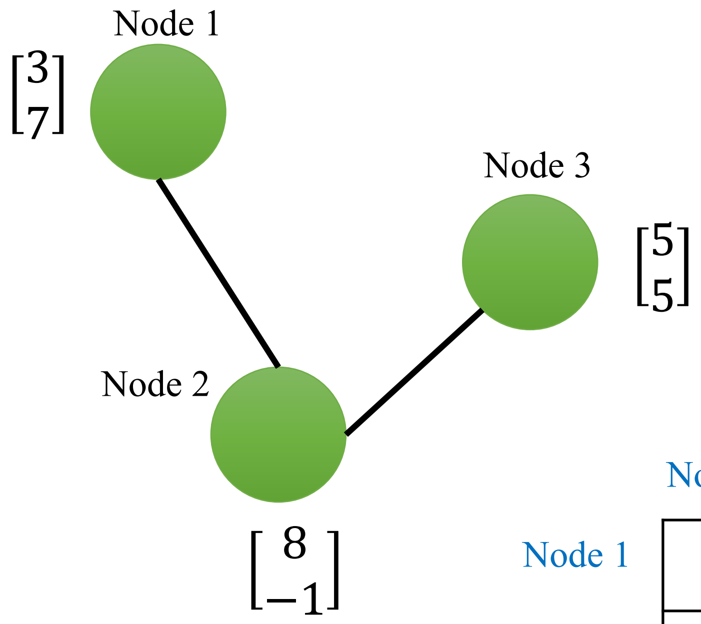
$$\begin{bmatrix} 0.12 & 0.58 & 1.17 \\ -0.44 & 3.11 & -0.8 \\ 5.11 & -0.31 & 4.17 \end{bmatrix}$$



0.12×48 + 0.58×109 + 1.17×57 -0.44×17 - 0.8×126 + 5.11×13 -0.31×13 + 4.17×64 = 633.88

0.12×48 + 0.58×109 + 1.17×57 -0.44×17 + 3.11×52 -0.8×126 + 5.11×13 -0.31×13 + 4.17×64 = 635.5

From CNN to GNN



What if information in one channel is more important than other channel?

	Node 1	Node 2	Node 3
Node 1	1	1	0
Node 2	1	1	1
Node 3	0	1	1

Adjacency matrix

×

Node 1	3	7
Node 2	8	-1
Node 3	5	5

H: node feature

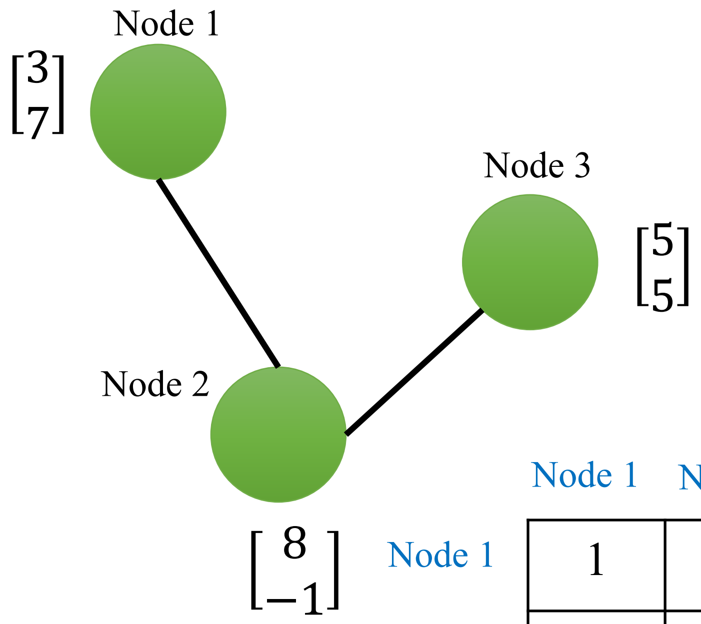
=

Node 1	N1 + N2
Node 2	N1 + N2 + N3
Node 3	N2 + N3

=

Node 1	3+8	7+-1
Node 2	3+8+5	7-1+5
Node 3	8+5	-1+5

From CNN to GNN



What if information in one channel is more important than other channel?

	Node 1	Node 2	Node 3
Node 1	1	1	0
Node 2	1	1	1
Node 3	0	1	1

Adjacency matrix

×

Node 1	3	7
Node 2	8	-1
Node 3	5	5

H: node feature

×

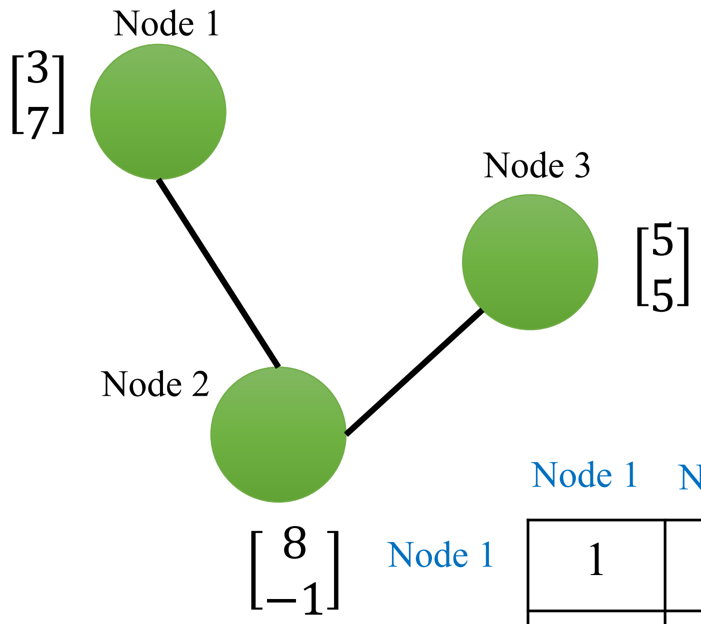
W11	W12
W21	W22

W: weight matrix

=

(3+8)W11 + (7-1)W21	(3+8)W12 + (7-1)W22
(3+8+5)W11+ (7-1+5)W21	(3+8+5)W12 + (7-1+5)W22
(8+5) W11 + (-1+5)W21	(8+5) W12 + (-1+5)W22

From CNN to GNN



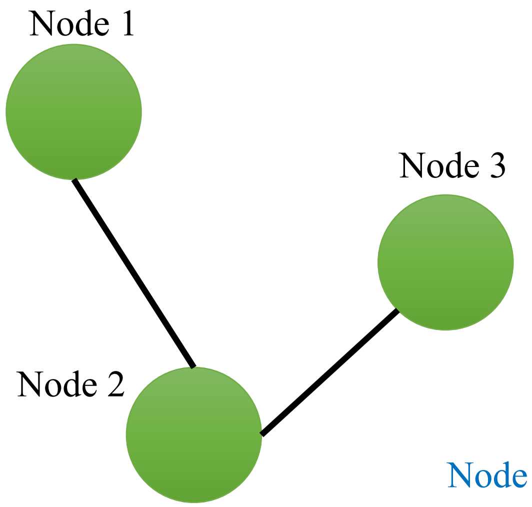
What if information in one channel is more important than other channel?

Finally, this is Graph Convolutional Neural Network

$$\begin{array}{cc} & \begin{array}{ccc}\text{Node 1}&\text{Node 2}&\text{Node 3}\end{array}\\ \begin{array}{c}\text{Node 1}\\\text{Node 2}\\\text{Node 3}\end{array} & \begin{bmatrix}1&1&0\\1&1&1\\0&1&1\end{bmatrix}\end{array} \times \begin{array}{cc}\begin{bmatrix}3&7\\8&-1\\5&5\end{bmatrix} & \begin{array}{c}\text{Node 1}\\\text{Node 2}\\\text{Node 3}\end{array}\end{array} \times \begin{bmatrix}W11\\W21\end{bmatrix} = \begin{bmatrix}(3+8)W11+(7-1)W21\\(3+8+5)W11+(7-1+5)W21\\(8+5)W11+(-1+5)W21\end{bmatrix}$$

Adjacency matrix   **H: node feature**   **W: weight matrix**

From CNN to GNN



$\begin{bmatrix}3\\7\end{bmatrix}$ $\begin{bmatrix}5\\5\end{bmatrix}$ $\begin{bmatrix}8\\-1\end{bmatrix}$

Adjacency matrix: $\tilde{A} = A + I_N$

Activation Function

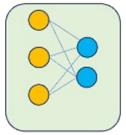
Adjacency matrix:

New Embedding: $H^{(l+1)} = \sigma\left(\boxed{\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}}H^{(l)}W^{(l)}\right)$

Normalization $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Node feature

layer-specific trainable weight matrix



	Node 1	Node 2	Node 3
Node 1	1	1	0
Node 2	1	1	1
Node 3	0	1	1

Adjacency matrix

×

3	7	Node 1
8	-1	Node 2
5	5	Node 3

H: node feature

×

W11
W21

W: weight matrix

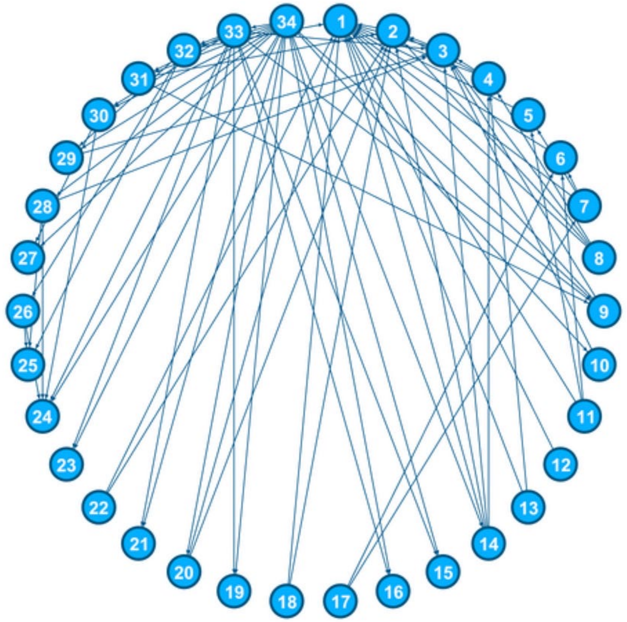
=

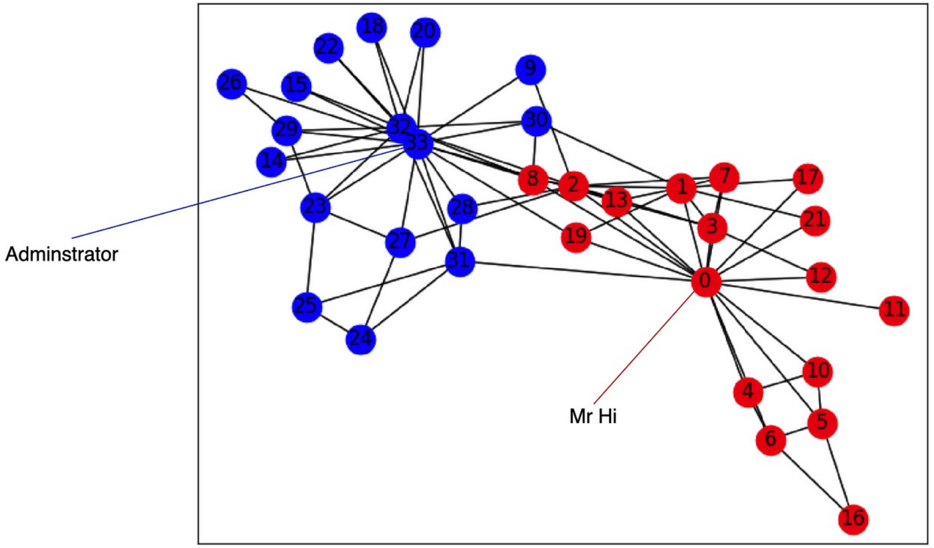
(3+8)W11 + (7-1)W21
(3+8+5)W11+ (7-1+5)W21
(8+5) W11 + (-1+5)W21

What if information in one node is more important than other node? → Attention GNN

Zachary’s karate club

There is a karate club that has two major stakeholders: the instructor (Mr. Hi) and the administrator. Unfortunately, the dispute between them causes it to split into 2 clubs. The original members will need to choose a side and pick which one to join. Their decisions will be based on how well they are connected with Mr. Hi or the administrator. This also includes how well they are connected to members that are associated with them. The diagram below is the social graph in representation connections between members.





Zachary's karate club

```
A = nx.to_numpy_array(g)
A
```

```
array([[0., 4., 5., ..., 2., 0., 0.],
       [4., 0., 6., ..., 0., 0., 0.],
       [5., 6., 0., ..., 0., 2., 0.],
       ...,
       [2., 0., 0., ..., 0., 4., 4.],
       [0., 0., 2., ..., 4., 0., 5.],
       [0., 0., 0., ..., 4., 5., 0.]])
```

```
[39] A_mod = A + np.eye(g.number_of_nodes()) # add self-connections

D_mod = np.zeros_like(A_mod)
np.fill_diagonal(D_mod, np.asarray(A_mod.sum(axis=1)).flatten())


D_mod_invroot = np.linalg.inv(sqrtm(D_mod))

A_hat = D_mod_invroot @ A_mod @ D_mod_invroot
```

$$\tilde{A} = A + I$$

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

Zachary's karate club

```
[40] X = np.eye(g.number_of_nodes())
```

```
print(X)

[[1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
```

Input Feature X

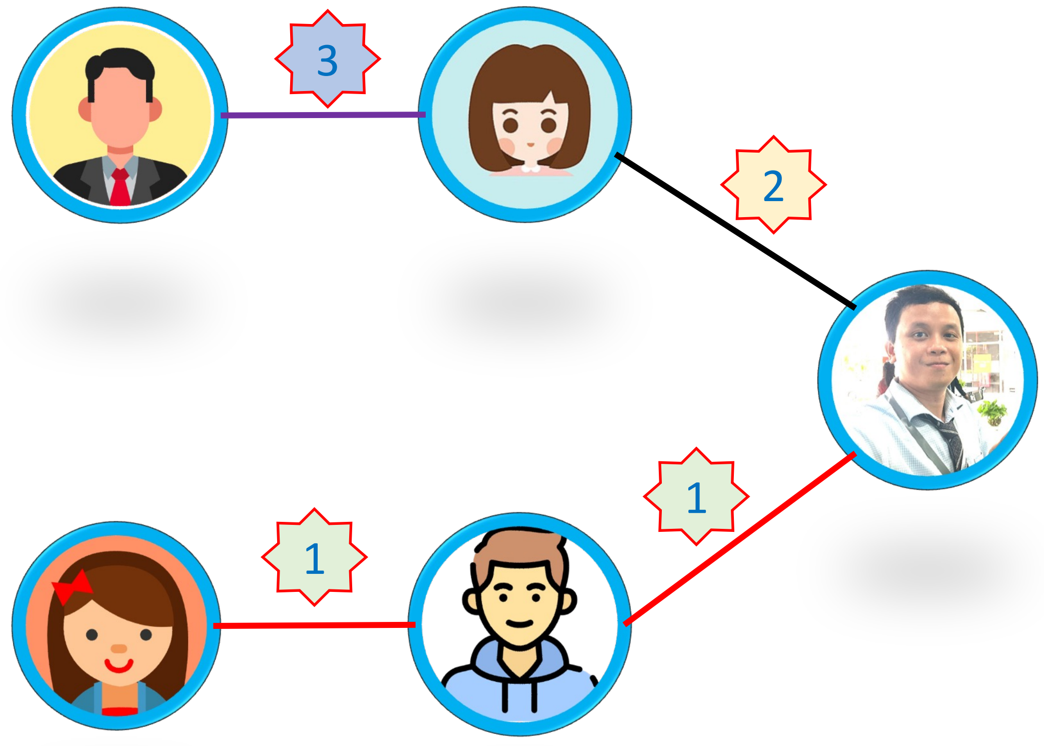
Zachary’s karate club

```python
class GCNLayer():
    def __init__(self, n_inputs, n_outputs, activation=None, name=''):
        self.n_inputs = n_inputs
        self.n_outputs = n_outputs
        self.W = glorot_init(self.n_outputs, self.n_inputs)
        self.activation = activation
        self.name = name

    def __repr__(self):
        return f"GCN: W{'_'+self.name if self.name else ''} ({self.n_inputs}, {self.n_outputs})"

    def forward(self, A, X, W=None):
        """
        Assumes A is (bs, bs) adjacency matrix and X is (bs, D),
            where bs = "batch size" and D = input feature length
        """
        self._A = A
        self._X = (A @ X).T # for calculating gradients.  (D, bs)

        if W is None:
            W = self.W

        H = W @ self._X # (h, D)*(D, bs) -> (h, bs)
        if self.activation is not None:
            H = self.activation(H)
        self._H = H # (h, bs)
        return self._H.T # (bs, h)
```

$$H^{l+1} = \sigma(W \underbrace{\hat{A} H^{l}}_{\text{Message Passing}})$$

Different Types of Edge Connections

How to include various types of Edges in GNN

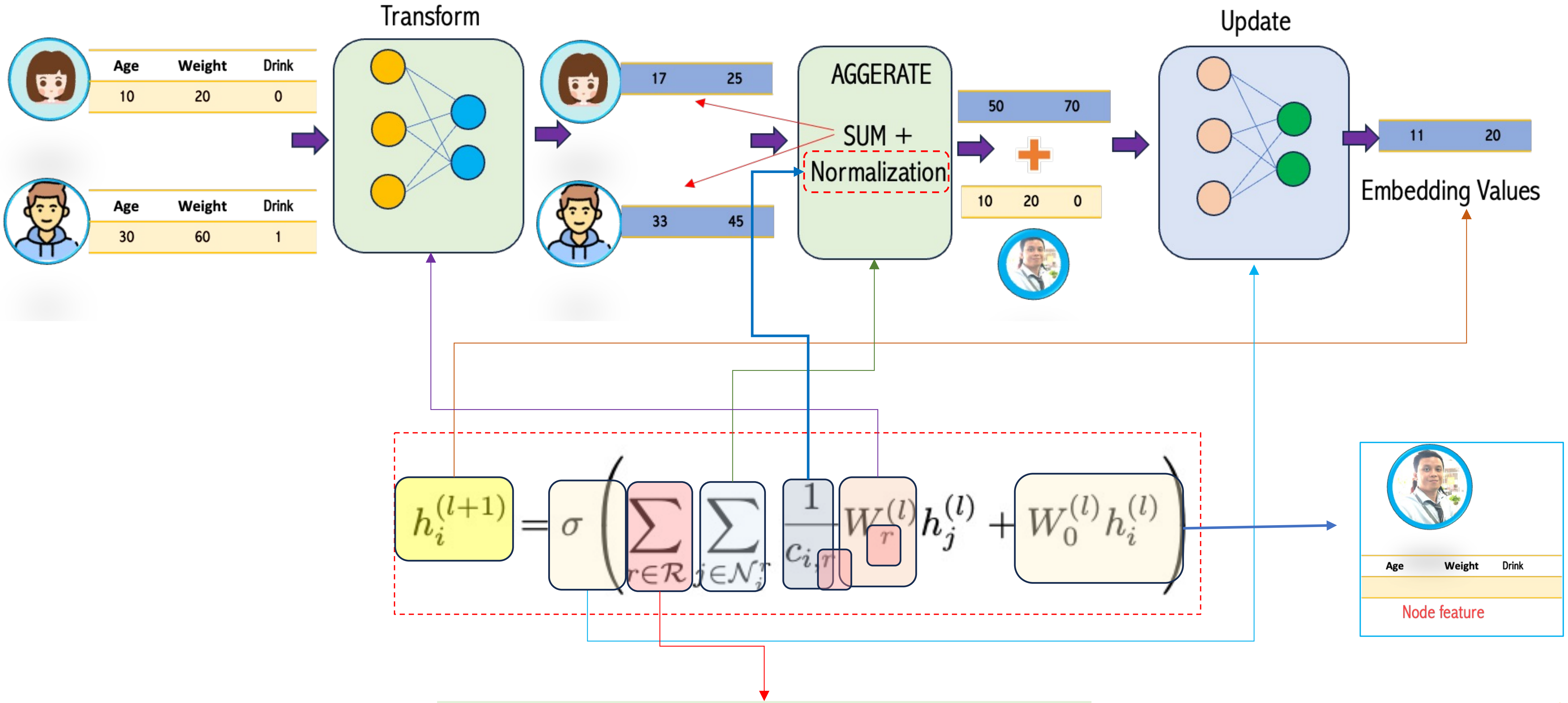


Relational Graph Convolutional Neural Network

1 Friend   2 Family   3 Colleagues

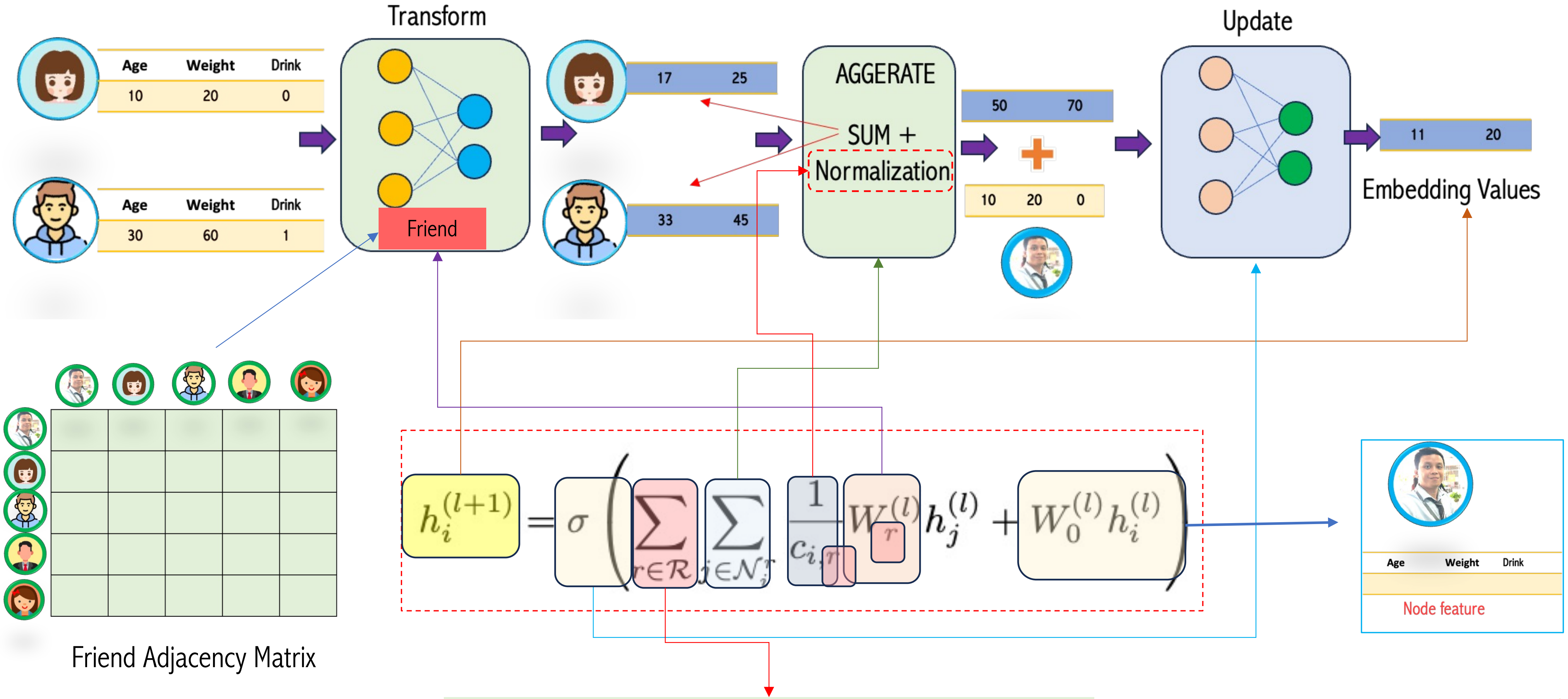
Outline

Relational GNN



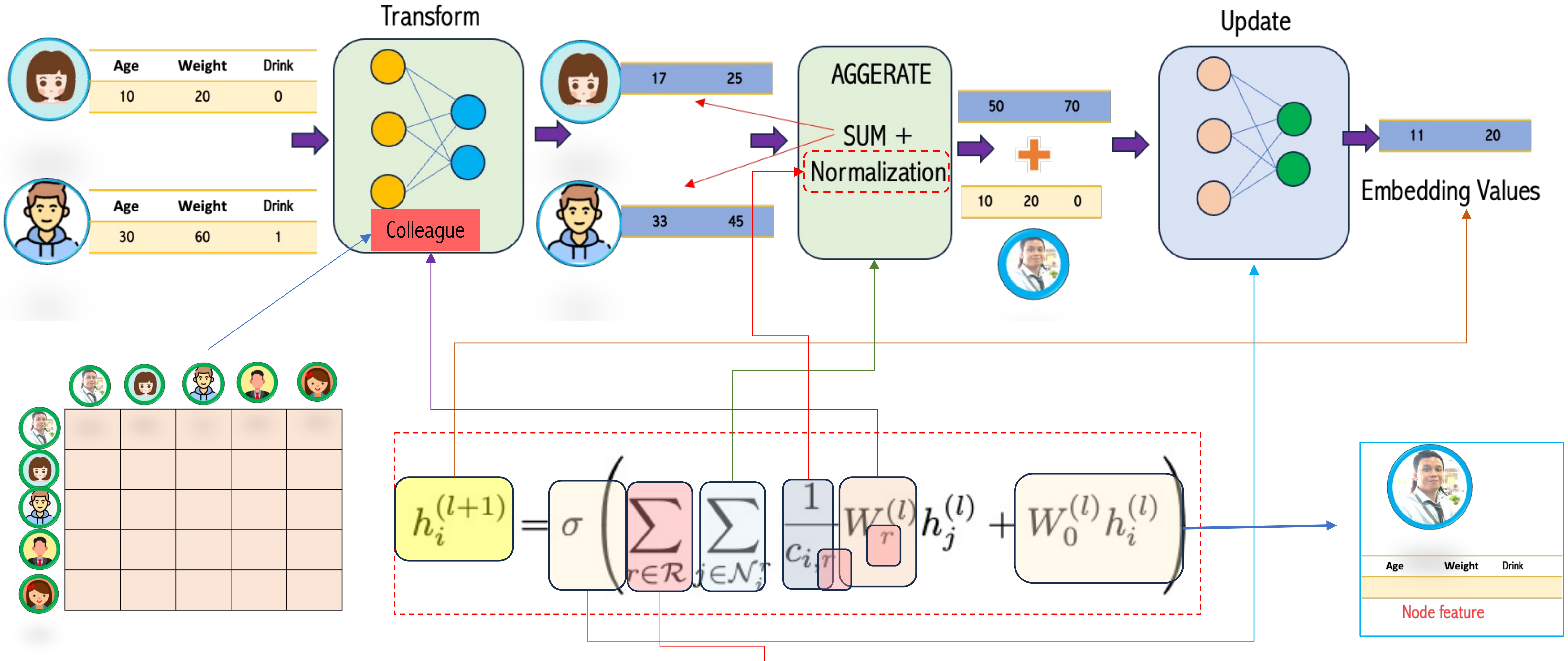
Various types of relationship: Edge conditions GNN

Relational GNN



Various types of relationship: Edge conditions GNN

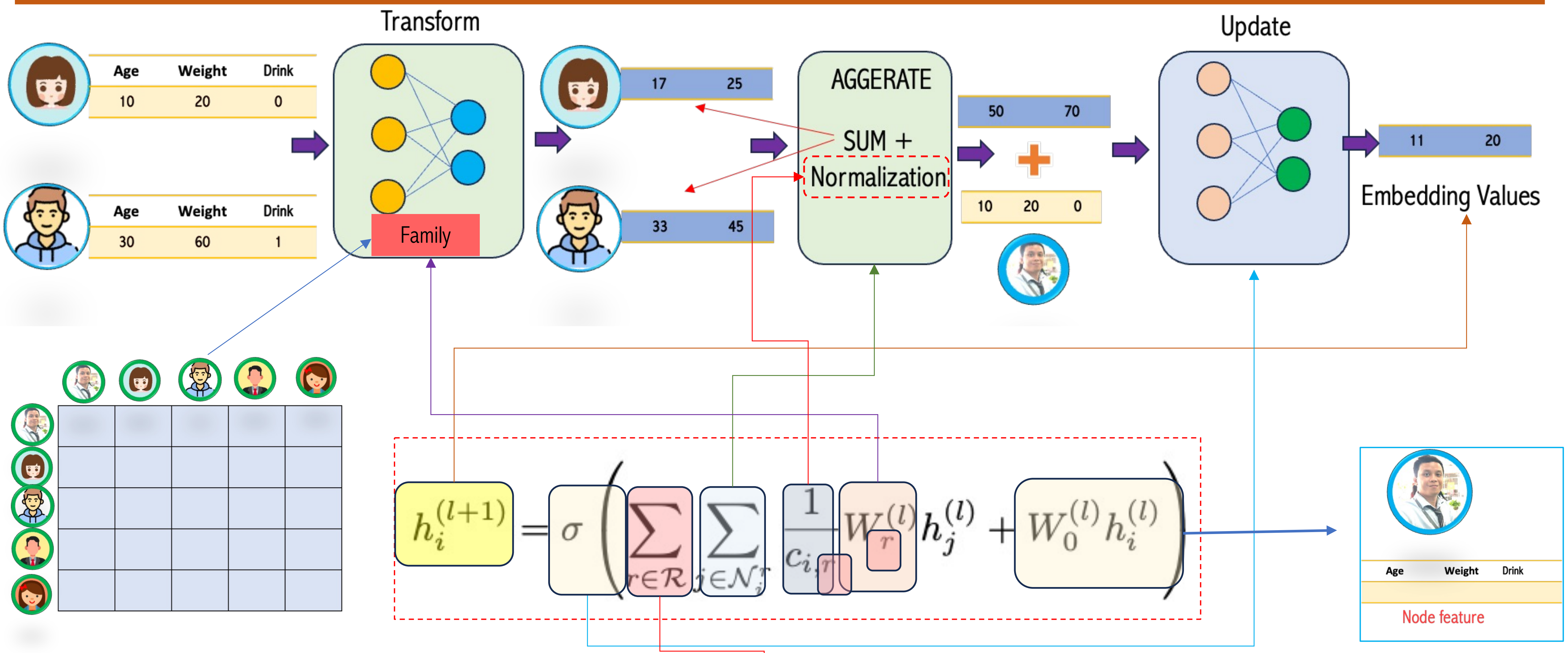
Relational GNN



$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

Various types of relationship: Edge conditions GNN

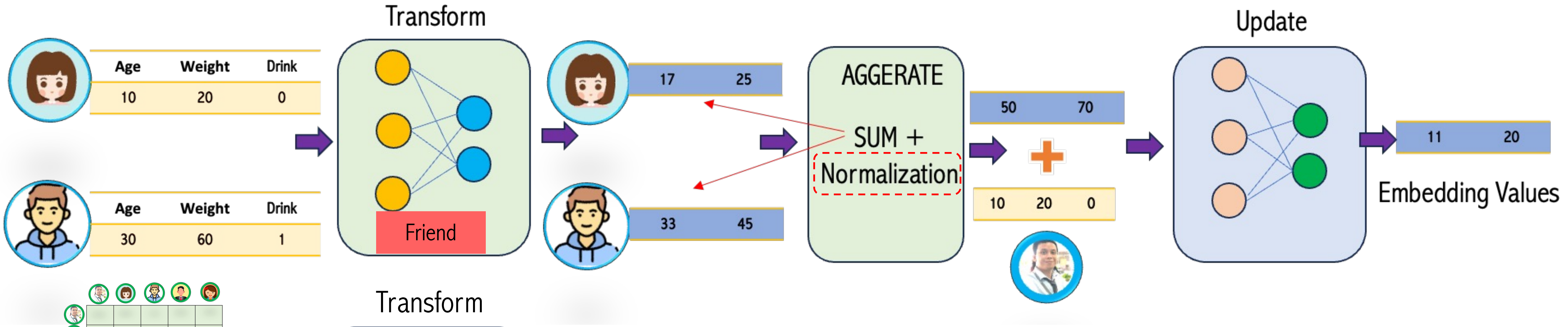
Relational GNN

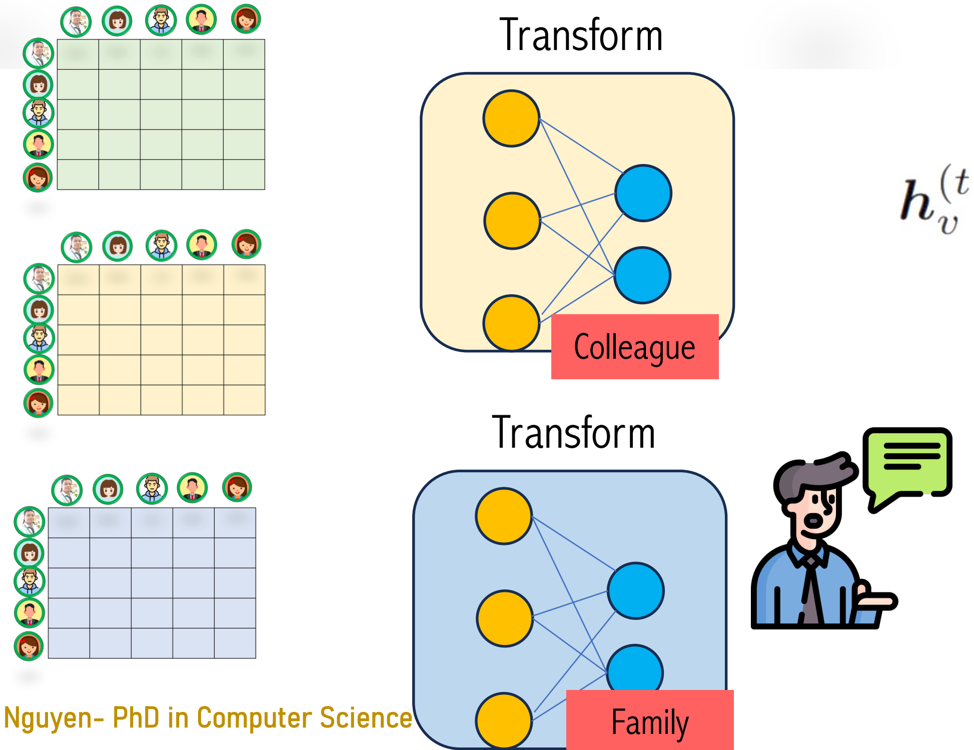


$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

Various types of relationship: Edge conditions GNN

GNN: Feature-wise Linear Modulation





$$h_v^{(t+1)} = l\left(\sum_{u \xrightarrow{\ell} v \in \mathcal{E}} \sigma\left(\gamma_{\ell,v}^{(t)} \odot \boldsymbol{W}_\ell h_u^{(t)} + \beta_{\ell,v}^{(t)}\right) ; \boldsymbol{\theta}_l\right)$$

How are edge features use?

GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation

Marc Brockschmidt [1]

Abstract

This paper presents a new Graph Neural Network (GNN) type using feature-wise linear modulation (FiLM). Many standard GNN variants propagate information along the edges of a graph by com-

Most neural graph learning methods can be summarised as neural message passing (Gilmer et al., 2017): nodes are initialised with some representation and then exchange information by transforming their current state (in practice with a single linear layer) and sending it as a message to

GNN Edge Feature: Variants

$$
\begin{aligned}
\text{GGNN: } A' &= GRU\left(A,\ \boldsymbol{W}_1 \cdot B + \boldsymbol{W}_2 \cdot C + \boldsymbol{W}_1 \cdot D\right) \\
\text{R-GCN: } A' &= \sigma\left(\boldsymbol{W}_{\circlearrowleft} \cdot A + \boldsymbol{W}_1 \cdot B + \boldsymbol{W}_2 \cdot C + \boldsymbol{W}_1 \cdot D\right) \\
\text{R-GAT: } A' &= \sigma\left((\boldsymbol{a}_{A'})_{A \xrightarrow{\circlearrowleft} A} \cdot \boldsymbol{W}_{\circlearrowleft} \cdot A + (\boldsymbol{a}_{A'})_{B \xrightarrow{1} A} \cdot \boldsymbol{W}_1 \cdot B + (\boldsymbol{a}_{A'})_{C \xrightarrow{2} A} \cdot \boldsymbol{W}_2 \cdot C + (\boldsymbol{a}_{A'})_{D \xrightarrow{1} A} \cdot \boldsymbol{W}_1 \cdot D\right) \\
\text{R-GIN: } A' &= \sigma\left(MLP_{\circlearrowleft}(A) + MLP_1(B) + MLP_2(C) + MLP_1(D)\right) \\
\text{GNN-MLP: } A' &= \sigma\left(MLP_{\circlearrowleft}(A \| A) + MLP_1(B \| A) + MLP_2(C \| A) + MLP_1(D \| A)\right) \\
\text{RGDCN: } A' &= \sigma\left(\boldsymbol{W}_{\circlearrowleft, A} \cdot A + \boldsymbol{W}_{1,A} \cdot B + \boldsymbol{W}_{2,A} \cdot C + \boldsymbol{W}_{1,A} \cdot D\right) \\
\text{GNN-FiLM: } A' &= \sigma\left(\boldsymbol{\beta}_{\circlearrowleft, A} + \boldsymbol{\gamma}_{\circlearrowleft, A} \odot \boldsymbol{W}_{\circlearrowleft} \cdot A + \boldsymbol{\beta}_{1,A} + \boldsymbol{\gamma}_{1,A} \odot \boldsymbol{W}_1 \cdot B + \boldsymbol{\beta}_{2,A} + \boldsymbol{\gamma}_{2,A} \odot \boldsymbol{W}_2 \cdot C + \boldsymbol{\beta}_{1,A} + \boldsymbol{\gamma}_{1,A} \odot \boldsymbol{W}_1 \cdot D\right)
\end{aligned}
$$

GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation

Marc Brockschmidt [1]

Abstract

This paper presents a new Graph Neural Network (GNN) type using feature-wise linear modulation (FiLM). Many standard GNN variants propagate

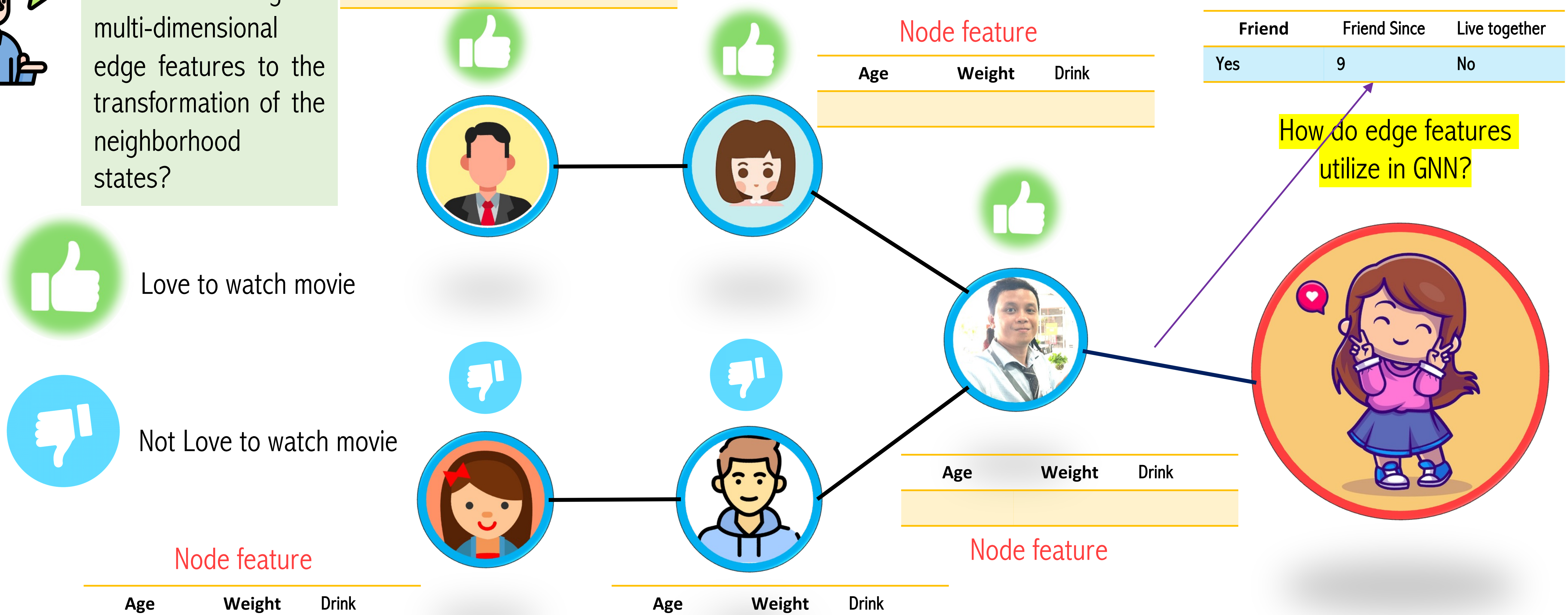
Most neural graph learning methods can be summarised as neural message passing (Gilmer et al., 2017): nodes are initialised with some representation and then exchange information by transforming their current state (in practice with a single linear layer) and sending it as a message to

Outline

- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

Multidimensional Edge Feature

How to integrate multi-dimensional edge features to the transformation of the neighborhood states?



Age	Weight	Drink

Age	Weight	Drink

Age	Weight	Drink

Age	Weight	Drink

Age	Weight	Drink

Friend	Friend Since	Live together
Yes	9	No

Multidimensional Edge Features

$$h_v = \gamma\left(x_v, \bigoplus_{w\in N(v)} \phi\left(x_v, x_w, \boxed{e_{wv}}\right)\right)$$

UPDATE

AGGREGATE

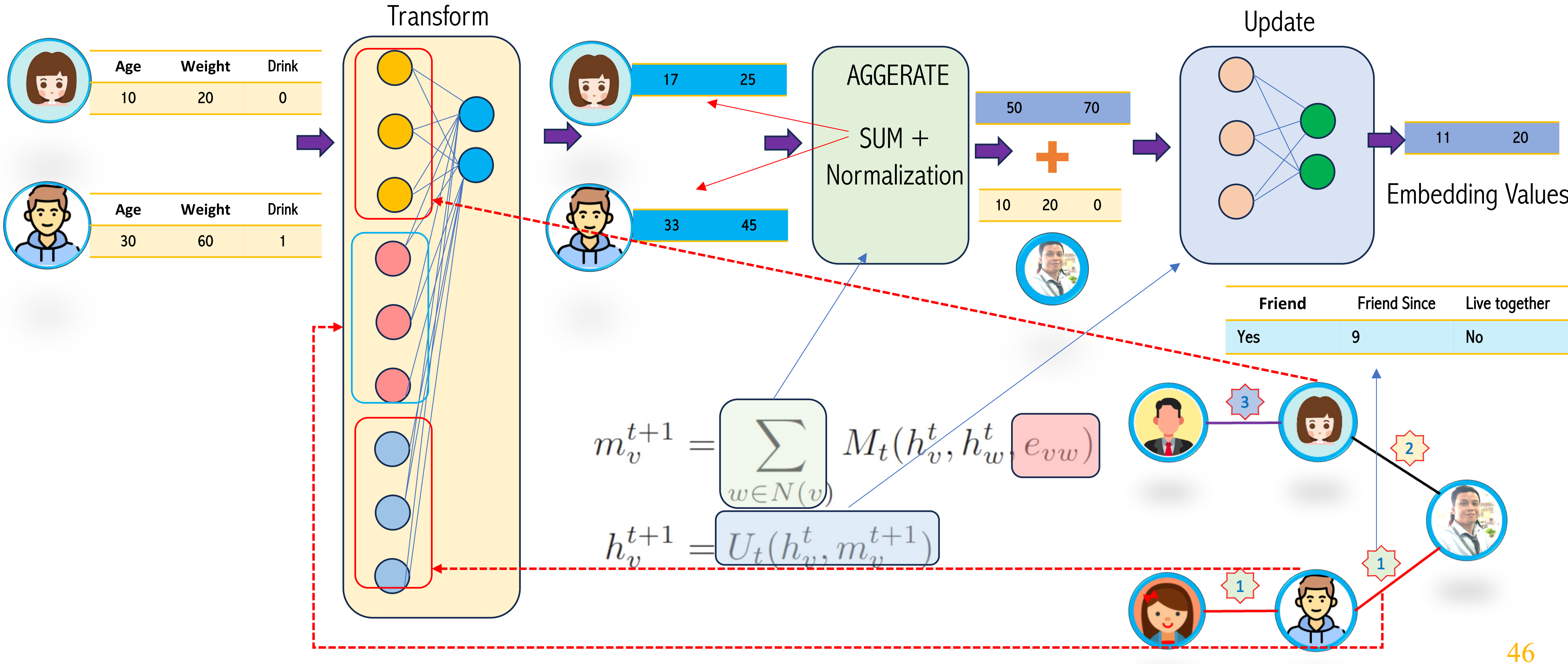
TRANSFORM

Message Passing in Neural Network

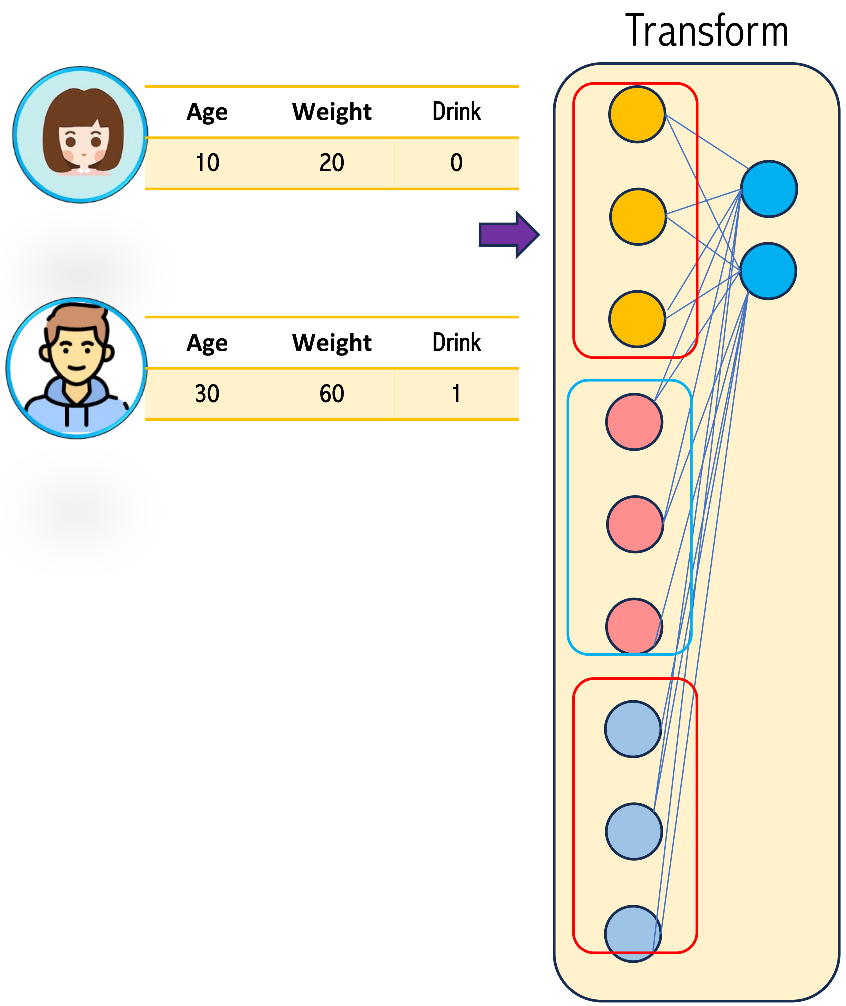
	0	[edge feature vector]	[edge feature vector]	0	0
	[edge feature vector]	0	0	[edge feature vector]	0
	[edge feature vector]	0	0	0	[edge feature vector]
	0	[edge feature vector]	0	0	0
	0	0	[edge feature vector]	0	0

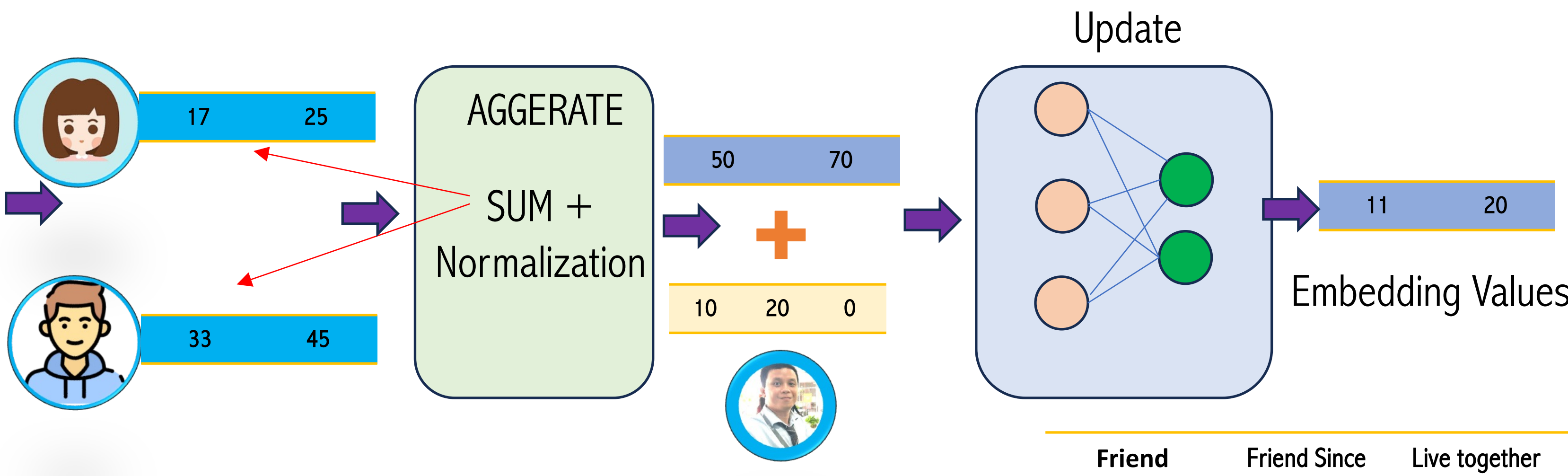
What is the size of this adjacency matrix?

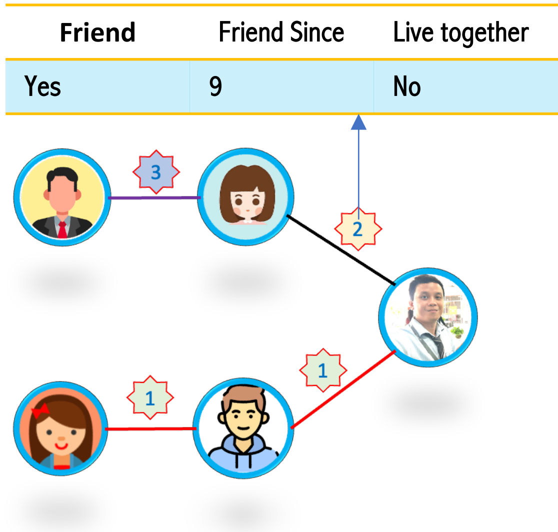
Multidimensional Edge Features: MLP



Multidimensional Edge Features: PNAConv







$$X_i^{(t+1)} = U\left(X_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(X_i^{(t)}, \boxed{E_{j\to i}}, X_j^{(t)}\right)\right)$$

Principal Neighbourhood Aggregation for Graph Nets

Gabriele Corso* University of Cambridge gc579@cam.ac.uk

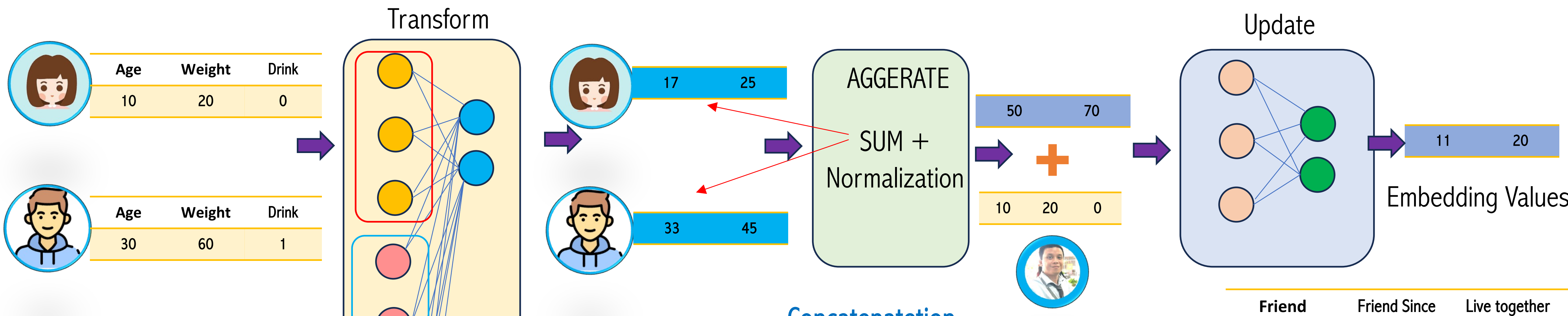
Luca Cavalleri* University of Cambridge lc737@cam.ac.uk

Dominique Beaini InVivo AI dominique@invivoai.com

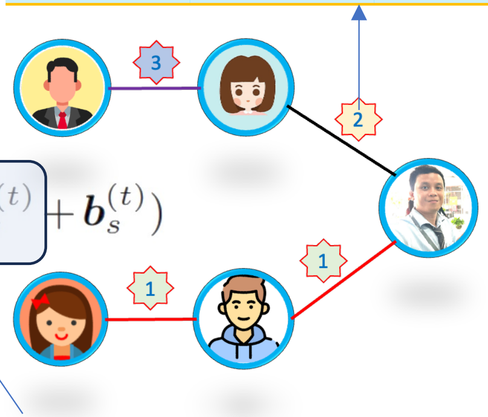
Pietro Liò University of Cambridge pietro.lio@cst.cam.ac.uk

Petar Veličković DeepMind petarv@google.com

Multidimensional Edge Features: Crystal GCN



Friend	Friend Since	Live together
Yes	9	No



Concatenatetion

$$z^{(t)}_{(i,j)_k} = \boxed{v^{(t)}_i} \oplus \boxed{v^{(t)}_j} \oplus \boxed{u_{(i,j)_k}}$$

$$v^{(t+1)}_i = v^{(t)}_i + \sum_{j,k} \sigma\left(\boxed{z^{(t)}_{(i,j)_k} W^{(t)}_f + b^{(t)}_f}\right) \odot g\left(\boxed{z^{(t)}_{(i,j)_k} W^{(t)}_s} + b^{(t)}_s\right)$$

Learnable weight matrix

Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties

Tian Xie and Jeffrey C. Grossman
Department of Materials Science and Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States
(Dated: April 10, 2018)

Edge Feature Embedding: NENN

NENN: Incorporate Node and Edge Features in Graph Neural Networks

Yulei Yang YANGYULEI18@NUDT.EDU.CN
Dongsheng Li LDS1201@163.COM
National University of Defense Technology, Chashang, China

Hierachical dual-level attention mechanism

Learn how important specific nodes and edges are for the new embedding

$$\mathbf{x}_{\mathcal{N}_i} = \sigma\Big(W_n \cdot \mathrm{MEAN}(\{\alpha_{ij}^{n}\mathbf{x}_j, \forall_j \in \mathcal{N}_i\})\Big)$$

$$\mathbf{x}_{\mathcal{E}_i} = \sigma\Big(W_e \cdot \mathrm{MEAN}(\{\alpha_{ik}^{e}\mathbf{e}_k, \forall_k \in \mathcal{E}_i\})\Big)$$

$$\mathbf{x}_i^{(l+1)} = \mathrm{CONCAT}(\mathbf{x}_{\mathcal{N}_i}^{(l)}, \mathbf{x}_{\mathcal{E}_i}^{(l)})$$

Nodel-level Attention

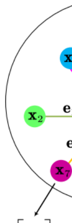
$$\mathbf{e}_{\mathcal{E}_i} = \sigma\Big(W_e \cdot \mathrm{MEAN}(\{\beta_{ik}^{e}\mathbf{e}_k, \forall_k \in \mathcal{E}_i\})\Big)$$

$$\mathbf{e}_{\mathcal{N}_i} = \sigma\Big(W_n \cdot \mathrm{MEAN}(\{\beta_{ij}^{n}\mathbf{x}_j, \forall_j \in \mathcal{N}_i\})\Big)$$

$$\mathbf{e}_i^{(l+1)} = \mathrm{CONCAT}(\mathbf{e}_{\mathcal{N}_i}^{(l)}, \mathbf{e}_{\mathcal{E}_i}^{(l)})$$

Edge-level Attention

Edge feature are used in both layers to generate new embeeding.



Edge Feature Embedding: NENN

$$\mathbf{x}_{\mathcal{N}_i} = \sigma\Big(W_n \cdot \mathrm{MEAN}(\{\alpha^n_{ij}\mathbf{x}_j, \forall_j \in \mathcal{N}_i\})\Big)$$

$$\mathbf{x}_{\mathcal{E}_i} = \sigma\Big(W_e \cdot \mathrm{MEAN}(\{\alpha^e_{ik}\mathbf{e}_k, \forall_k \in \mathcal{E}_i\})\Big)$$

$$\mathbf{x}_i^{(l+1)} = \mathrm{CONCAT}(\mathbf{x}^{(l)}_{\mathcal{N}_i}, \mathbf{x}^{(l)}_{\mathcal{E}_i})$$

Nodel-level Attention

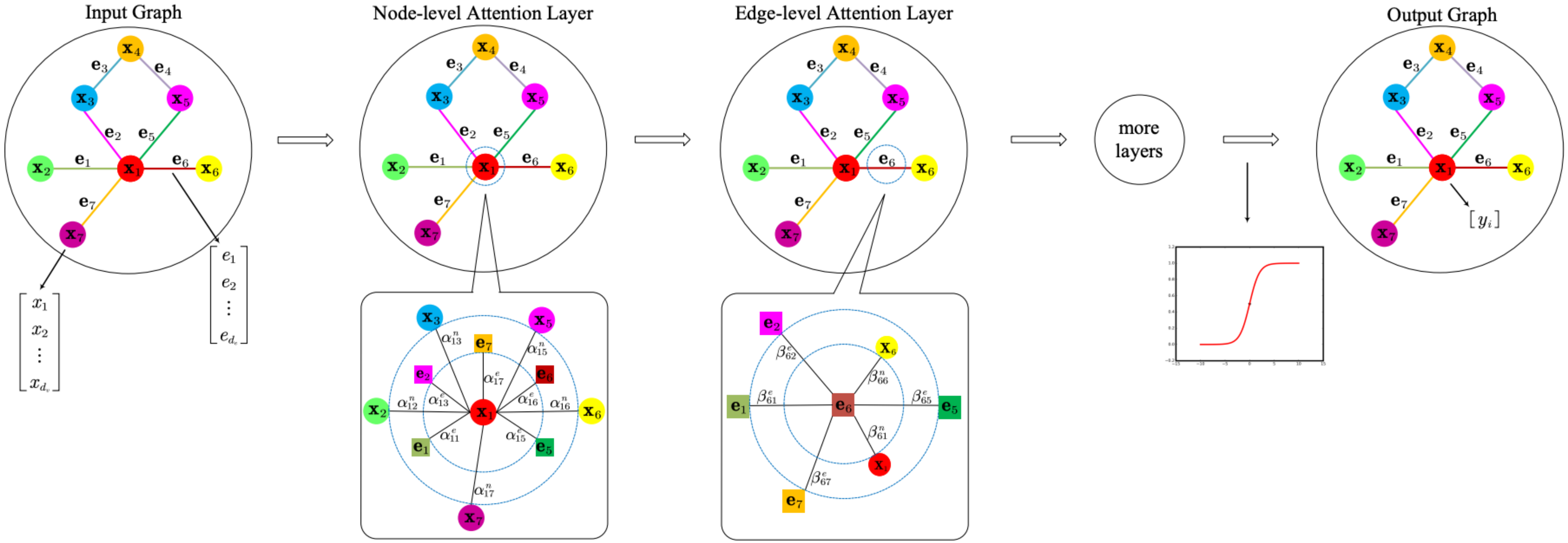
$$\mathbf{e}_{\mathcal{E}_i} = \sigma\Big(W_e \cdot \mathrm{MEAN}(\{\beta^e_{ik}\mathbf{e}_k, \forall_k \in \mathcal{E}_i\})\Big)$$

$$\mathbf{e}_{\mathcal{N}_i} = \sigma\Big(W_n \cdot \mathrm{MEAN}(\{\beta^n_{ij}\mathbf{x}_j, \forall_j \in \mathcal{N}_i\})\Big)$$

$$\mathbf{e}_i^{(l+1)} = \mathrm{CONCAT}(\mathbf{e}^{(l)}_{\mathcal{N}_i}, \mathbf{e}^{(l)}_{\mathcal{E}_i})$$

Edge-level Attention

This approach iteratively updates node and edge embeddings in order to merge both information together



Edge Feature: How to Use

PyTorch geometric

edge_weight → GNN Layer can use weight values on the adjacency matrix
edge_type → GNN Layer can use different edge types / relations
edge_attr → GNN Layer can use edge features

latest

Search docs

INSTALL PYG
- Installation

GET STARTED
- Introduction by Example
- Colab Notebooks and Video Tutorials

TUTORIALS
- Design of Graph Neural Networks
- Working with Graph Datasets
- Use-Cases & Applications
- Distributed Training

torch_geometric.utils

`scatter`	Reduces all values from the `src` tensor at the indices specified in the `index` tensor along a given dimension `dim`.
`group_argsort`	Returns the indices that sort the tensor `src` along a given dimension in ascending order by value.
`group_cat`	Concatenates the given sequence of tensors `tensors` in the given dimension `dim`.
`segment`	Reduces all values in the first dimension of the `src` tensor within the ranges specified in the `ptr`.
`index_sort`	Sorts the elements of the `inputs` tensor in ascending order.
`cumsum`	Returns the cumulative sum of elements of `x`.
`degree`	Computes the (unweighted) degree of a given one-dimensional index tensor.
`softmax`	Computes a sparsely evaluated softmax.
`lexsort`	Performs an indirect stable sort using a sequence of keys.
`sort_edge_index`	Row-wise sorts `edge_index`

- torch_geometric.sampler
- torch_geometric.datasets
- torch_geometric.transforms
- torch_geometric.utils
- torch_geometric.explain
- torch_geometric.metrics
- torch_geometric.distributed
- torch_geometric.contrib
- torch_geometric.graphgym
- torch_geometric.profile

CHEATSHEETS

GNN Cheatsheet
- Graph Neural Network Operators
- Heterogeneous Graph Neural Network Operators
- Hypergraph Neural Network Operators
- Point Cloud Neural Network Operators

Heterogeneous Graph Neural Network Operators

Name	SparseTensor	edge_weight	edge_attr	bipartite	static	lazy
RGCNConv (Paper)	✓					
FastRGCNConv	✓					
CuGraphRGCNConv (Paper)					✓	
RGATConv (Paper)	✓		✓			
FiLMConv (Paper)	✓			✓	✓	✓
HGTConv (Paper)	✓					✓
HEATConv (Paper)	✓		✓			✓
HeteroConv					✓	
HANConv (Paper)	✓					✓

Outline

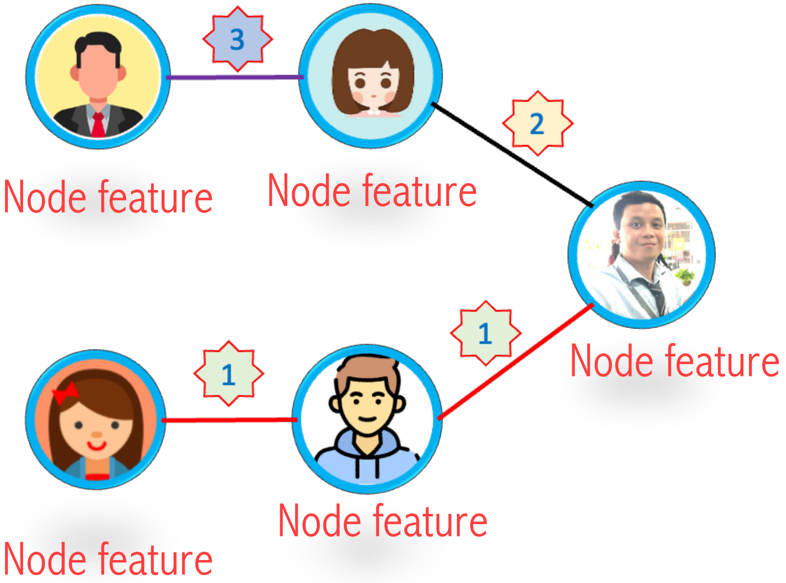
- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- **Attention in GNN**
- Example: Graph-Level Prediction
- Summary

AI VIETNAM
All-in-One Course

Attention in Graph Neural Network

1	1	1	0	0
1	1	0	1	0
1	0	1	0	1
0	1	0	1	0
0	0	1	0	1

Adjacency Matrix



Age	Weight	Drink

h1			
h2			
h3			
h4			
h5			

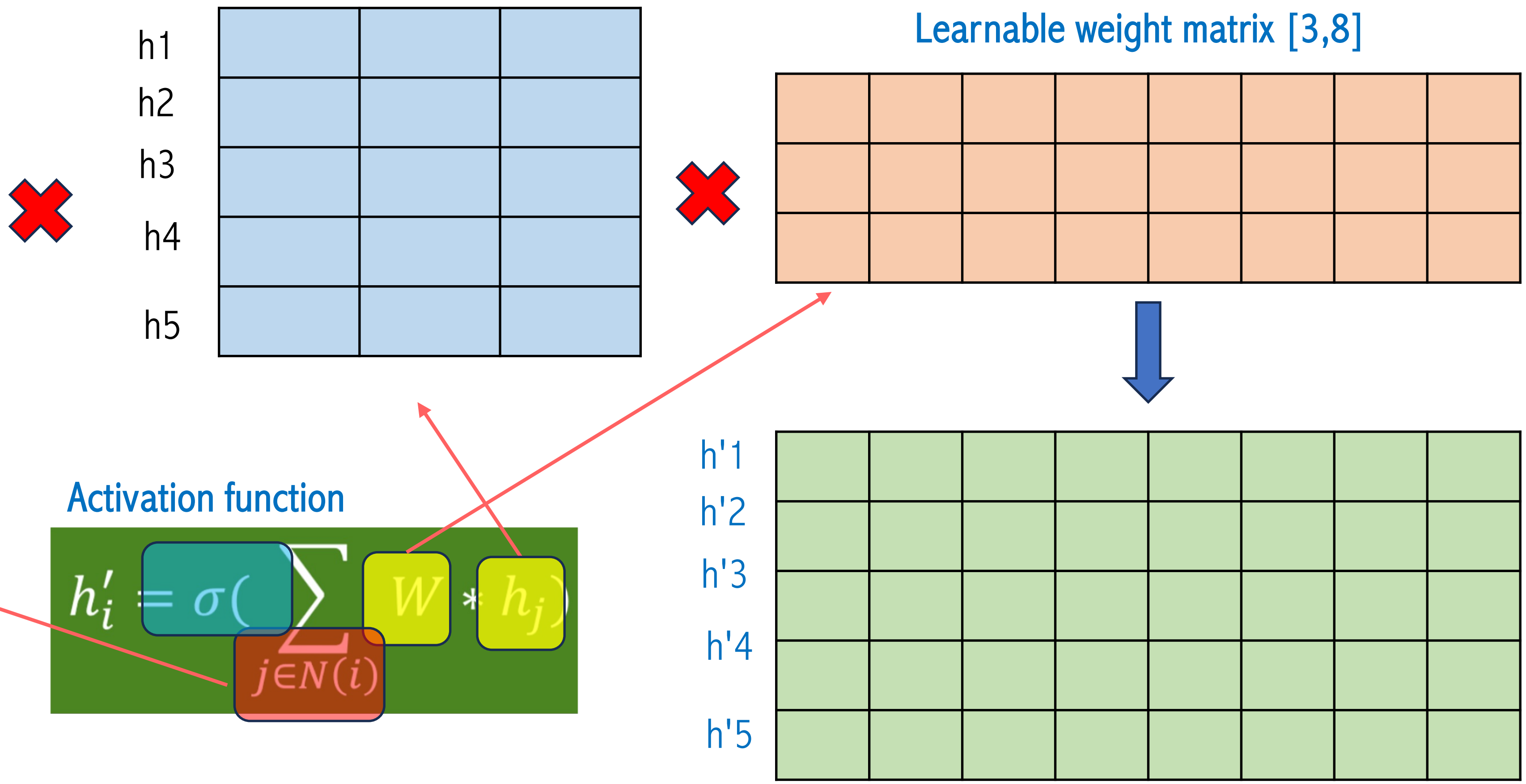
Features per node

How to use this feature matrix in GNN

GNN: Look at the 1st Layer

1	1	1	0	0
1	1	0	1	0
1	0	1	0	1
0	1	0	1	0
0	0	1	0	1

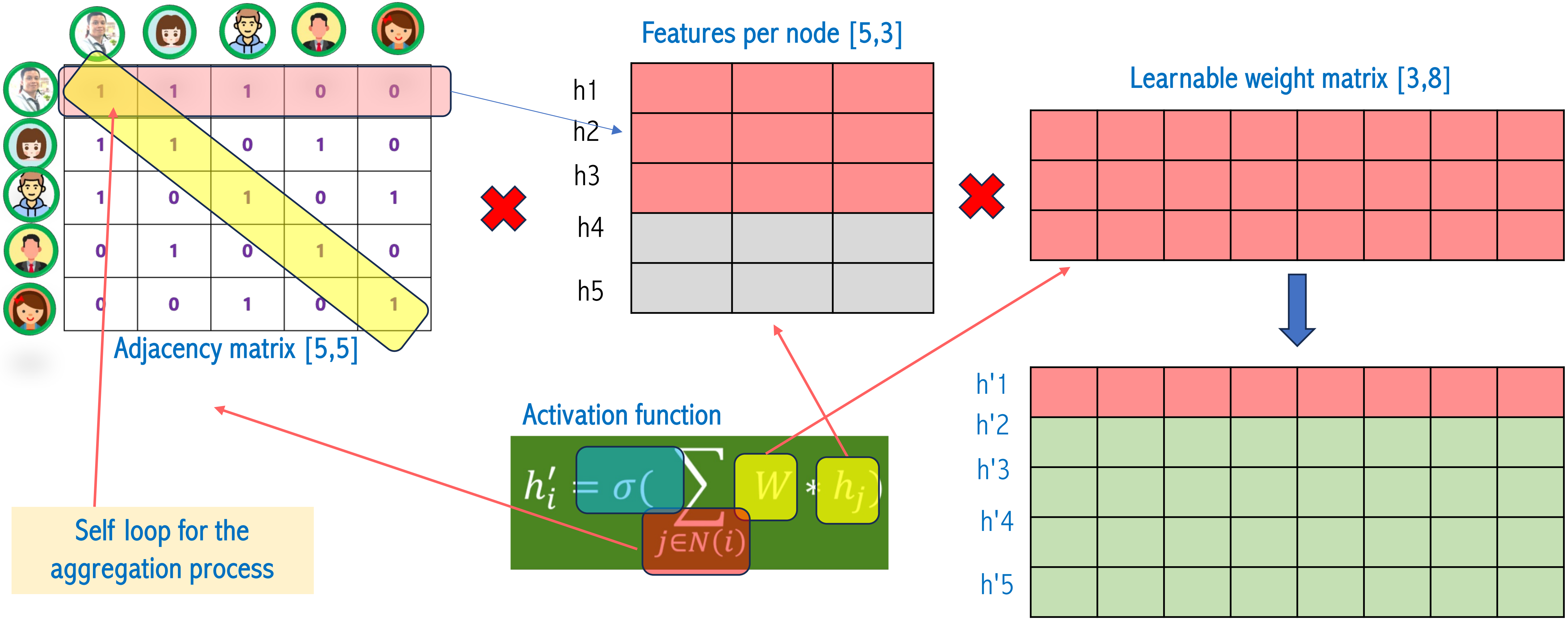
Adjacency matrix [5,5]



Embedding Features per node[1,8]: information of their own node feature and neigbor node features

AI VIETNAM
All-in-One Course

GNN: Look at the 1st Layer

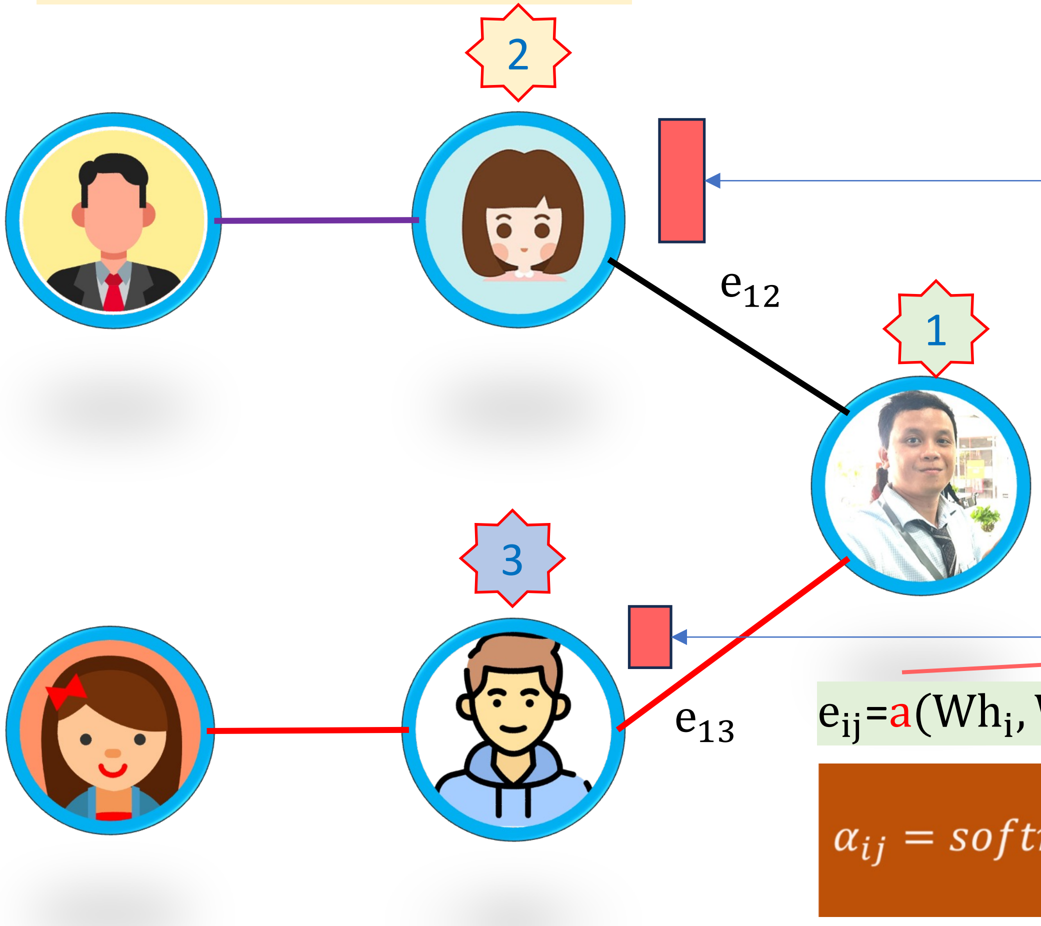


Embedding Features per node[1,8]: information of their own node feature and neigbor node features

GNN: Attention Mechanism

Learn how important node J's features are for node I: attention coefficient

For the word 'play', the word 'children' is more important than the word 'the'



The model should pay more attention to node 2

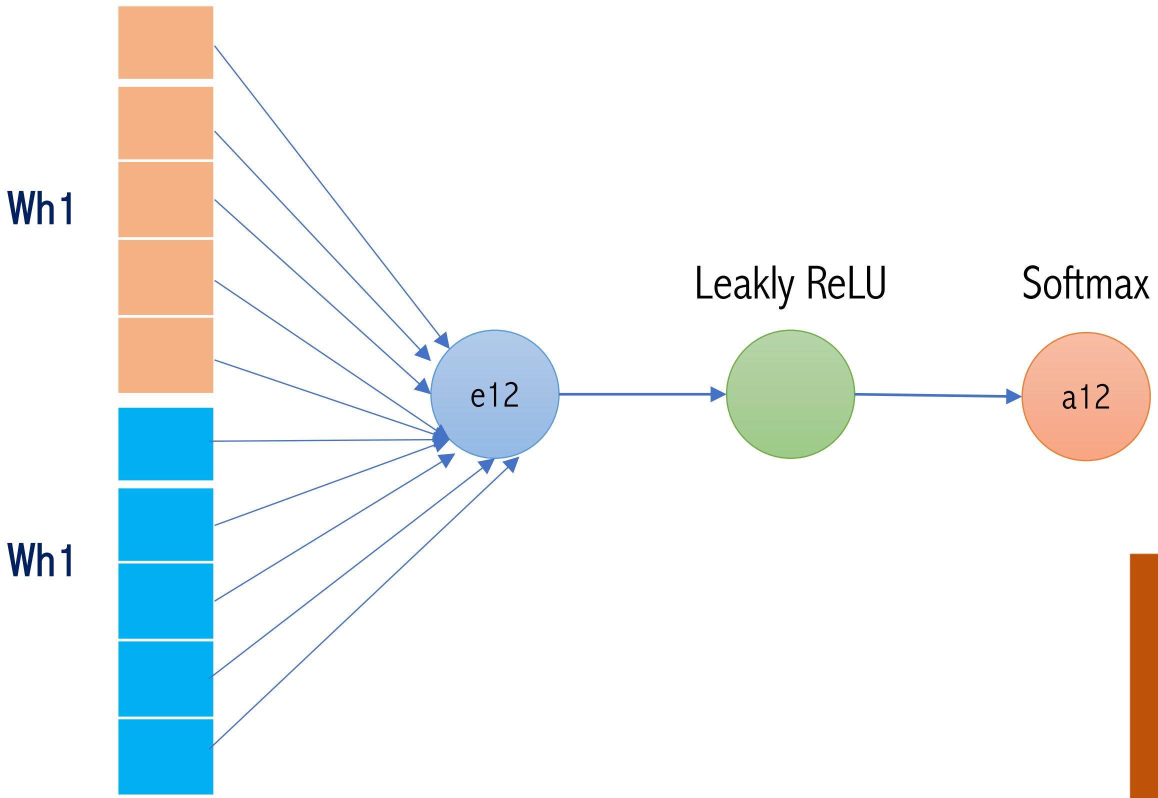
Attention mechanism

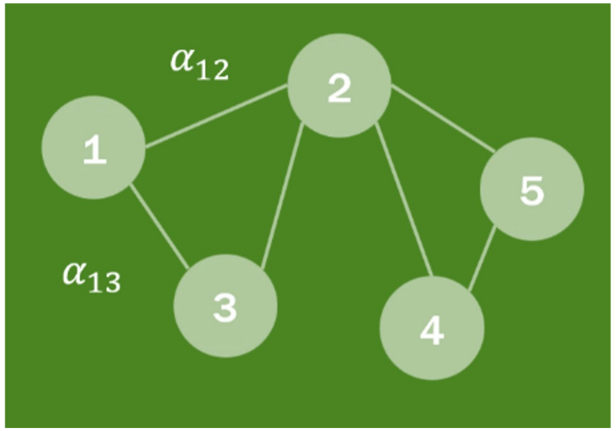
$e_{ij} = a(Wh_i, Wh_j)$

Only considering neighbor nodes

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \epsilon N(i)} \exp(e_{ik})} = \frac{\exp(a(Wh_i, Wh_j))}{\sum_{k \epsilon N(i)} \exp(a(Wh_i, Wh_j))}$$

GNN: Self-Attention Mechanism





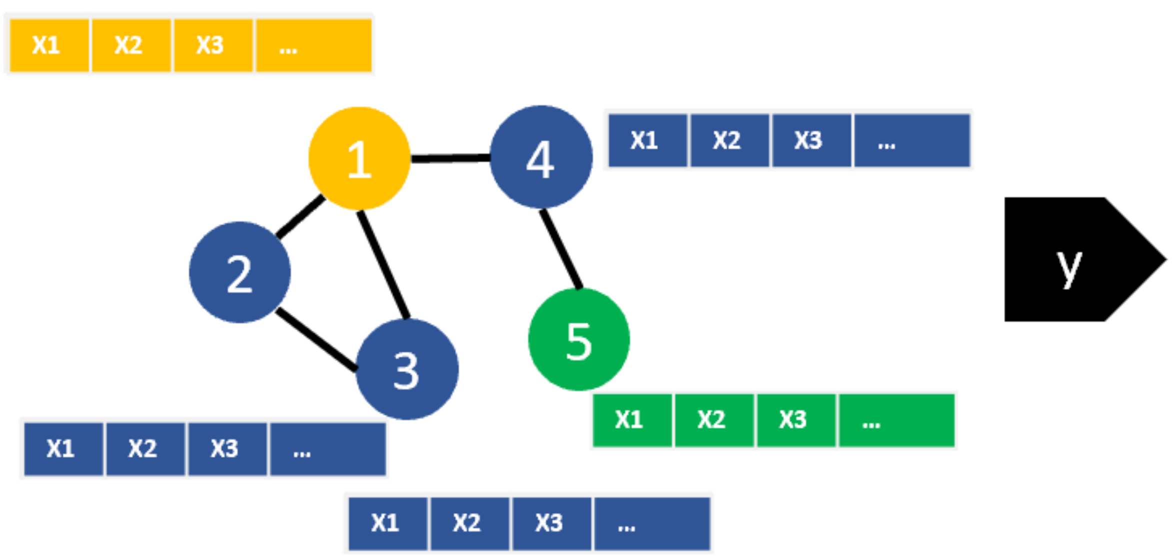
$$\alpha_{ij} = \frac{\exp\left(LeakyReLU\left(\overrightarrow{w_a^T}\,[Wh_i||Wh_j]\right)\right)}{\sum_{k\epsilon N(i)} \exp(LeakyReLU(\overrightarrow{w_a^T}\,[Wh_i||Wh_j]))}$$

QUIZ TIME

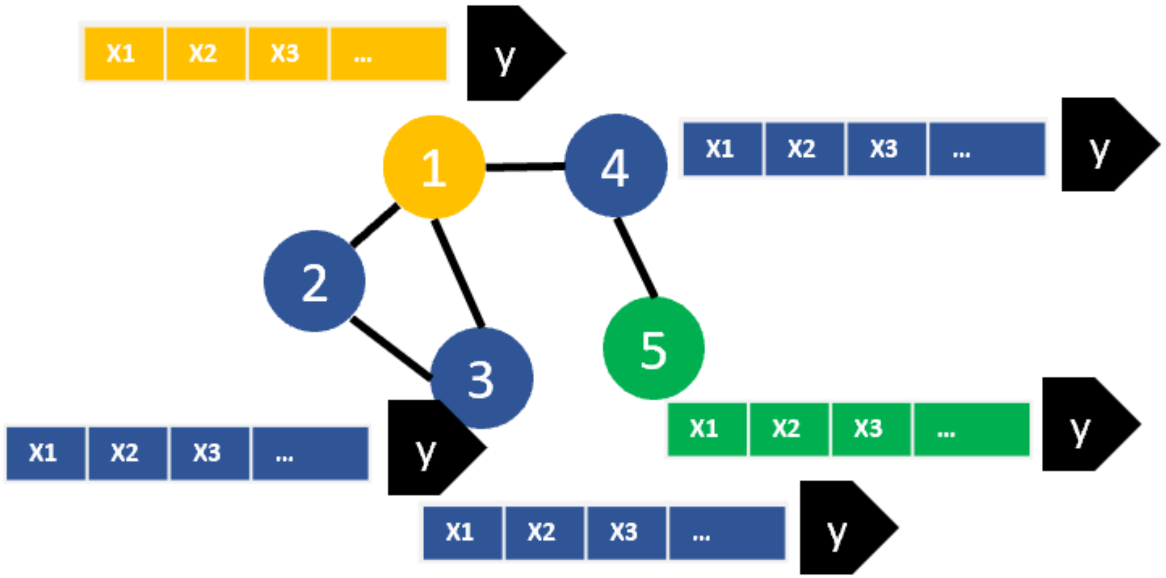
Outline

- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

GNN: Graph Prediction



Graph-level prediction



Node-level prediction

GNN: Graph Prediction

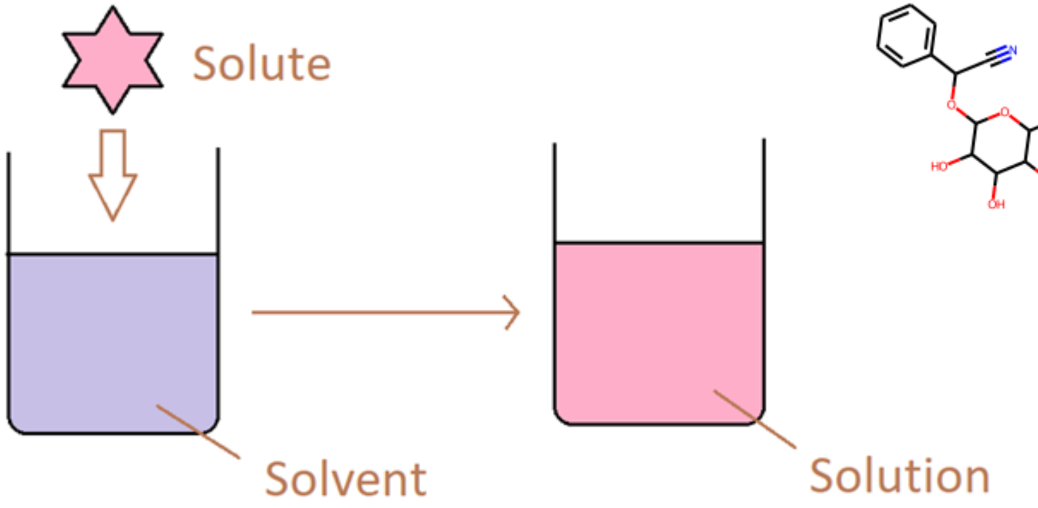
Installing Pytorch Geometric and RDKit

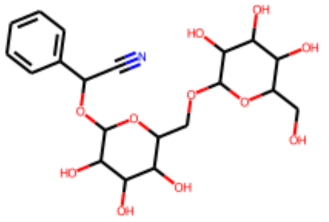
Background info on the Dataset

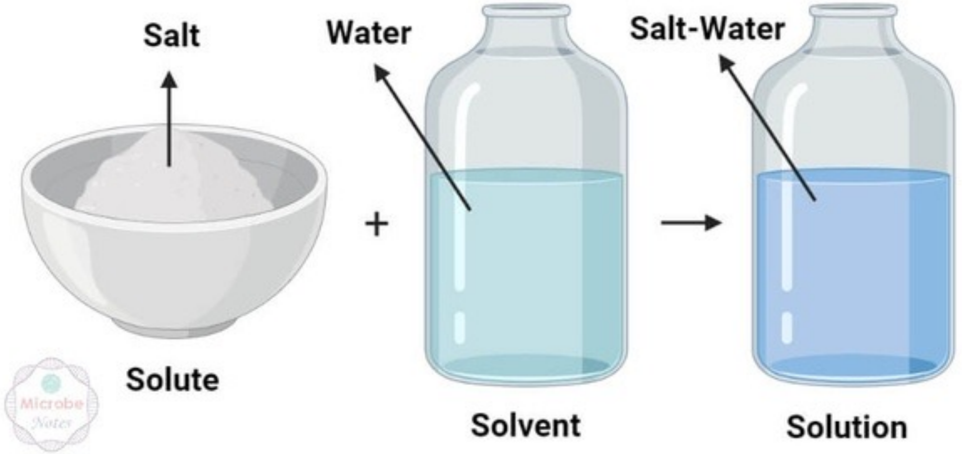
Looking into the Dataset

Visualizing molecules

Implementing the Graph Neural Network







Vinh Dinh Nguyen- PhD in Computer Science

GNN: Graph Prediction

- **Installing Pytorch Geometric and RDKit**
- Background info on the Dataset
- Looking into the Dataset
- Visualizing molecules
- Implementing the Graph Neural Network

```
!pip install torch-scatter      -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-sparse       -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-cluster      -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-spline-conv -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-geometric
```

```
!pip install rdkit
import rdkit
from torch_geometric.datasets import MoleculeNet
```

GNN: Graph Prediction

Installing Pytorch Geometric and RDKit

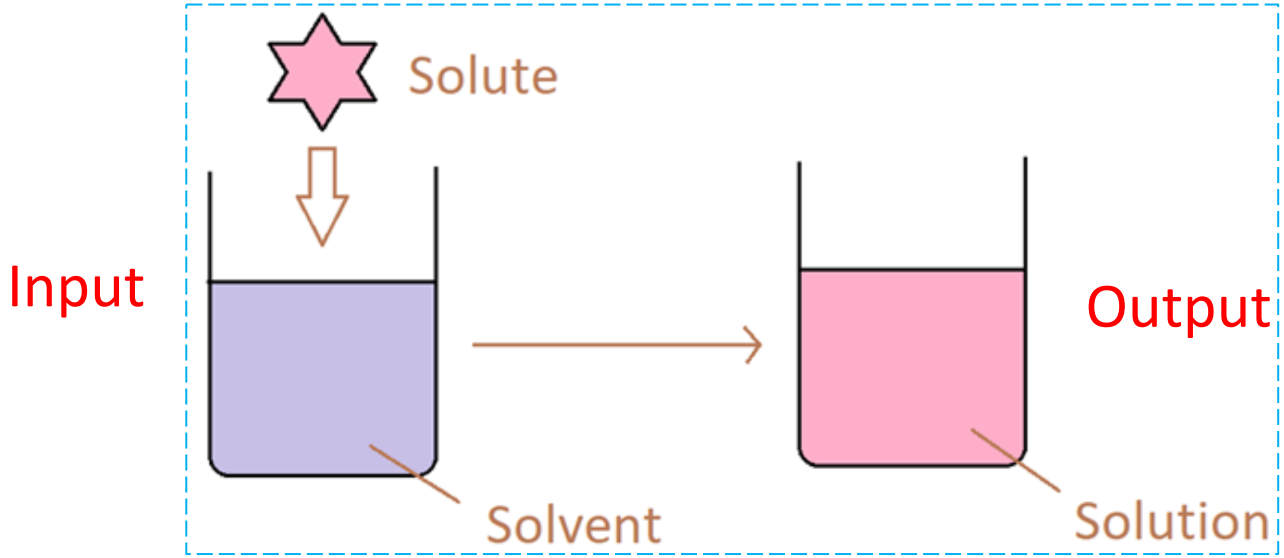
Background info on the Dataset

Looking into the Dataset

Visualizing molecules

Implementing the Graph Neural Network

How are different molecules dissolving in water?



ESOL is a small dataset consisting of water solubility data for 1128 compounds. The dataset has been used to train models that estimate solubility directly from chemical structures (as encoded in SMILES strings). Note that these structures don't include 3D coordinates, since solubility is a property of a molecule and not of its particular conformers.

GNN: Graph Prediction

Installing Pytorch Geometric and RDKit

Background info on the Dataset

Looking into the Dataset

Visualizing molecules

Implementing the Graph Neural Network

```
# Investigating the dataset
print("Dataset type: ", type(data))
print("Dataset features: ", data.num_features)
print("Dataset target: ", data.num_classes)
print("Dataset length: ", data.len)
print("Dataset sample: ", data[0])
print("Sample  nodes: ", data[0].num_nodes)
print("Sample  edges: ", data[0].num_edges)

# edge_index = graph connections
# smiles = molecule with its atoms
# x = node features (32 nodes have each 9 features)
# y = labels (dimension)
```

```
Dataset type:  <class 'torch_geometric.datasets.molecule_net.MoleculeNet'>
Dataset features:  9
Dataset target:  734
Dataset length:  <bound method InMemoryDataset.len of ESOL(1128)>
Dataset sample:  Data(x=[32, 9], edge_index=[2, 68], edge_attr=[68, 3], smiles='OCC3OC(OCC2OC(OC(C#N)
Sample  nodes:  32
Sample  edges:  68
```

GNN: Graph Prediction

Installing Pytorch Geometric and RDKit

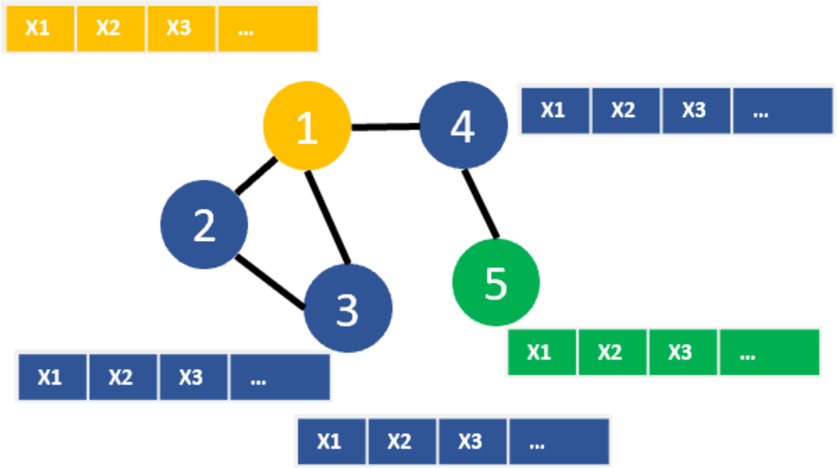
Background info on the Dataset

Looking into the Dataset

Visualizing molecules

Implementing the Graph Neural Network

```python
# Investigating the features
# Shape: [num_nodes, num_node_features]
data[0].x
```




```
tensor([[8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 2, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 0, 0, 4, 0, 1],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 0, 0, 4, 0, 0],
        [6, 0, 4, 5, 2, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 0, 0, 4, 0, 1],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 0, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 0],
        [6, 0, 2, 5, 0, 0, 2, 0, 0],
        [7, 0, 1, 5, 0, 0, 2, 0, 0],
        [6, 0, 3, 5, 0, 0, 3, 1, 1],
        [6, 0, 3, 5, 1, 0, 3, 1, 1],
        [6, 0, 3, 5, 1, 0, 3, 1, 1],
        [6, 0, 3, 5, 1, 0, 3, 1, 1],
        [6, 0, 3, 5, 1, 0, 3, 1, 1],
        [6, 0, 3, 5, 1, 0, 3, 1, 1],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 1, 0, 4, 0, 0]])
```

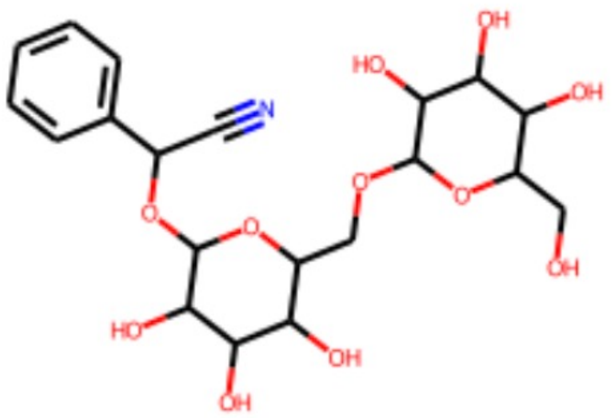
GNN: Graph Prediction

- Installing Pytorch Geometric and RDKit
- Background info on the Dataset
- Looking into the Dataset
- **Visualizing molecules**
- Implementing the Graph Neural Network

```
data[0]["smiles"]
```

```
'OCC3OC(OCC2OC(OC(C#N)c1ccccc1)C(O)C(O)C2O)C(O)C(O)C3O '
```

```
from rdkit import Chem
from rdkit.Chem.Draw import IPythonConsole
molecule = Chem.MolFromSmiles(data[0]["smiles"])
molecule
```



GNN: Graph Prediction


```
import torch
from torch.nn import Linear
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, TopKPooling, global_mean_pool
from torch_geometric.nn import global_mean_pool as gap, global_max_pool as gmp
embedding_size = 64

class GCN(torch.nn.Module):
    def __init__(self):
        # Init parent
        super(GCN, self).__init__()
        torch.manual_seed(42)

        # GCN layers
        self.initial_conv = GCNConv(data.num_features, embedding_size)
        self.conv1 = GCNConv(embedding_size, embedding_size)
        self.conv2 = GCNConv(embedding_size, embedding_size)
        self.conv3 = GCNConv(embedding_size, embedding_size)

        # Output layer
        self.out = Linear(embedding_size*2, 1)
```

GNN: Graph Prediction

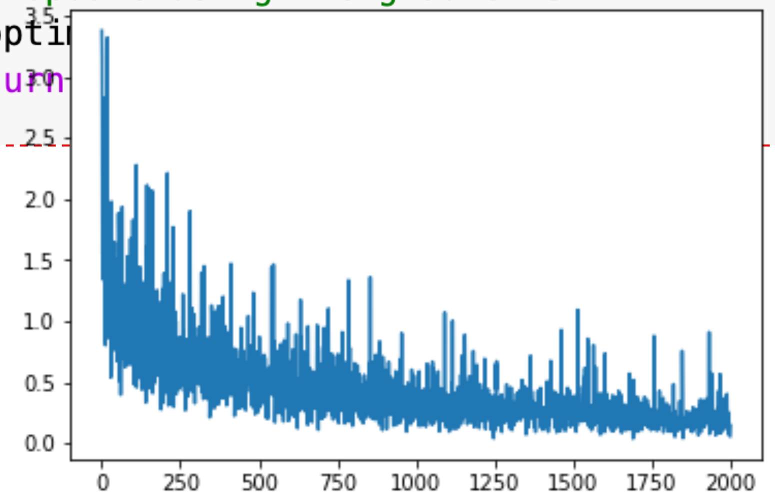

```
import torch
from torch.nn import Linear
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, TopKPooling, global_mean_pool
from torch_geometric.nn import global_mean_pool as gap, global_max_pool as gmp
embedding_size = 64

class GCN(torch.nn.Module):
    def __init__(self):
        # Init parent
        super(GCN, self).__init__()
        torch.manual_seed(42)

        # GCN layers
        self.initial_conv = GCNConv(data.num_features, embedding_size)
        self.conv1 = GC
        self.conv2 = GC
        self.conv3 = GC

        # Output layer
        self.out = Line
```

```
GCN(
  (initial_conv): GCNConv(9, 64)
  (conv1): GCNConv(64, 64)
  (conv2): GCNConv(64, 64)
  (conv3): GCNConv(64, 64)
  (out): Linear(in_features=128, out_features=1, bias=True)
)
Number of parameters:  13249
```

GNN: Graph Prediction

- Installing Pytorch Geometric and RDKit
- Background info on the Dataset
- Looking into the Dataset
- Visualizing molecules
- **Implementing the Graph Neural Network**

```
def train(data):
    # Enumerate over the data
    for batch in loader:
      # Use GPU
      batch.to(device)
      # Reset gradients
      optimizer.zero_grad()
      # Passing the node features and the connection info
      pred, embedding = model(batch.x.float(), batch
      # Calculating the loss and gradients
      loss = loss_fn(pred, batch.y)
      loss.backward()
      # Update using the gradients
      opti
    return
```




```
Starting training...
Epoch 0 | Train Loss 3.377596378326416
Epoch 100 | Train Loss 0.9617947340011597
Epoch 200 | Train Loss 1.0771363973617554
Epoch 300 | Train Loss 0.6295697093009949
Epoch 400 | Train Loss 0.37517455220222473
Epoch 500 | Train Loss 0.465716689825058
Epoch 600 | Train Loss 0.5129485726356506
Epoch 700 | Train Loss 0.21677978336811066
Epoch 800 | Train Loss 0.33871856331825256
Epoch 900 | Train Loss 0.3640660345554352
Epoch 1000 | Train Loss 0.20501013100147247
Epoch 1100 | Train Loss 0.18023353815078735
Epoch 1200 | Train Loss 0.2812242805957794
Epoch 1300 | Train Loss 0.18207958340644836
Epoch 1400 | Train Loss 0.1321338415145874
Epoch 1500 | Train Loss 0.18665631115436554
Epoch 1600 | Train Loss 0.1817747726440430
Epoch 1700 | Train Loss 0.09456530958414078
Epoch 1800 | Train Loss 0.23615044355392456
Epoch 1900 | Train Loss 0.11381624639034271
```

GNN: Graph Prediction

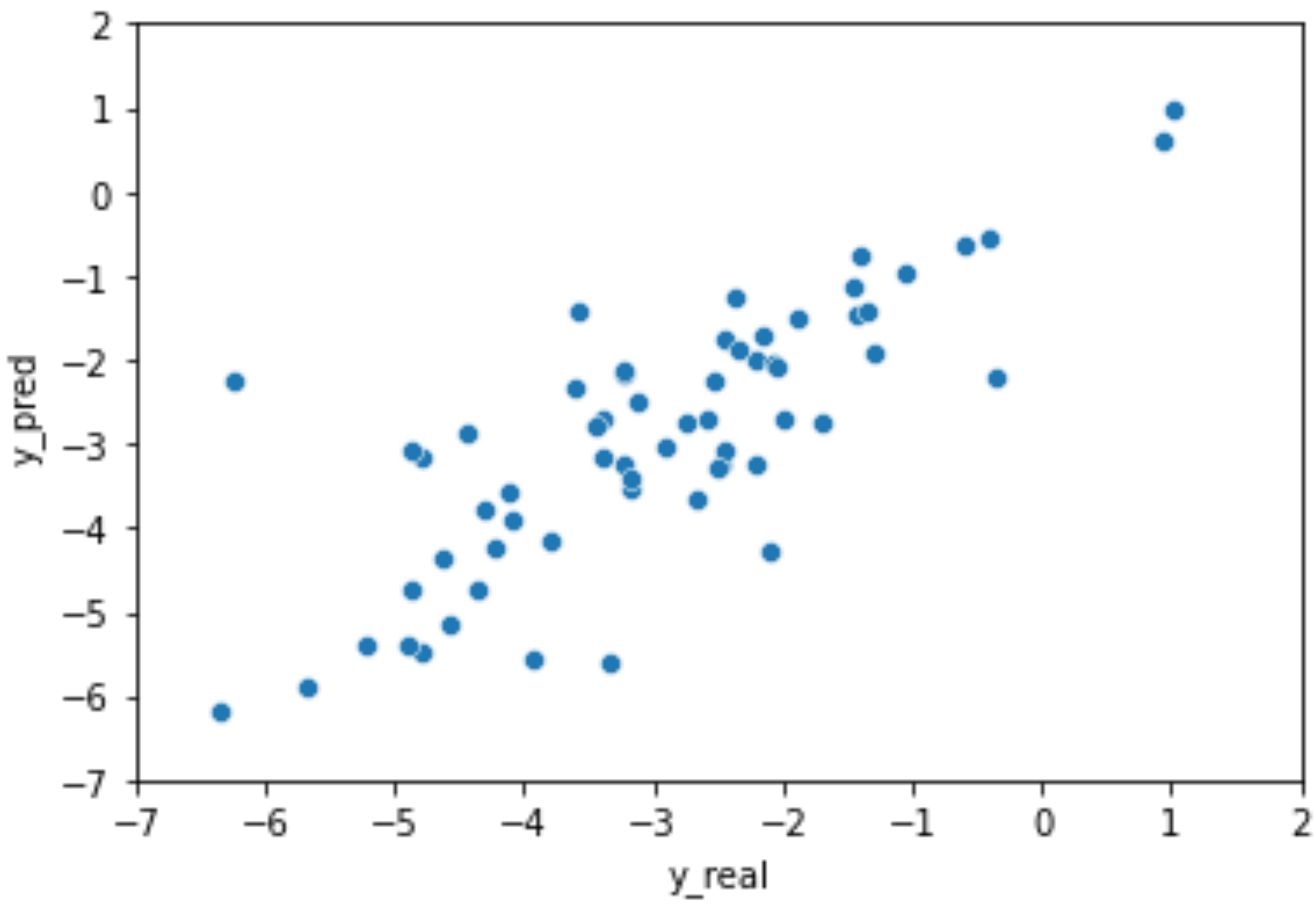
Installing Pytorch Geometric and RDKit

Background info on the Dataset

Looking into the Dataset

Visualizing molecules

Implementing the Graph Neural Network



Vision GNN: Read and Understand

Vision GNN: An Image is Worth Graph of Nodes

Kai Han[1,2*] **Yunhe Wang**[2*] **Jianyuan Guo**[2] **Yehui Tang**[2,3] **Enhua Wu**[1,4]
[1]State Key Lab of Computer Science, ISCAS & UCAS
[2]Huawei Noah's Ark Lab
[3]Peking University [4]University of Macau
{kai.han,yunhe.wang}@huawei.com, weh@ios.ac.cn

Abstract

Network architecture plays a key role in the deep learning-based computer vision system. The widely-used convolutional neural network and transformer treat the image as a grid or sequence structure, which is not flexible to capture irregular and complex objects. In this paper, we propose to represent the image as a graph structure and introduce a new *Vision GNN* (ViG) architecture to extract graph-level feature for visual tasks. We first split the image to a number of patches which are viewed as nodes, and construct a graph by connecting the nearest neighbors. Based on the graph representation of images, we build our ViG model to transform and exchange information among all the nodes. ViG consists of two basic modules: Grapher module with graph convolution for aggregating and updating graph information, and FFN module with two linear layers for node feature transformation. Both isotropic and pyramid architectures of ViG are built with different model sizes. Extensive experiments on image recognition and object detection tasks demonstrate the superiority of our ViG architecture. We hope this pioneering study of GNN on general visual tasks will provide useful inspiration and experience for future research.

The PyTorch code is available at `https://github.com/huawei-noah/Efficient-AI-Backbones` and the MindSpore code is available at `https://gitee.com/mindspore/models`.

Vision GNN: Read and Understand

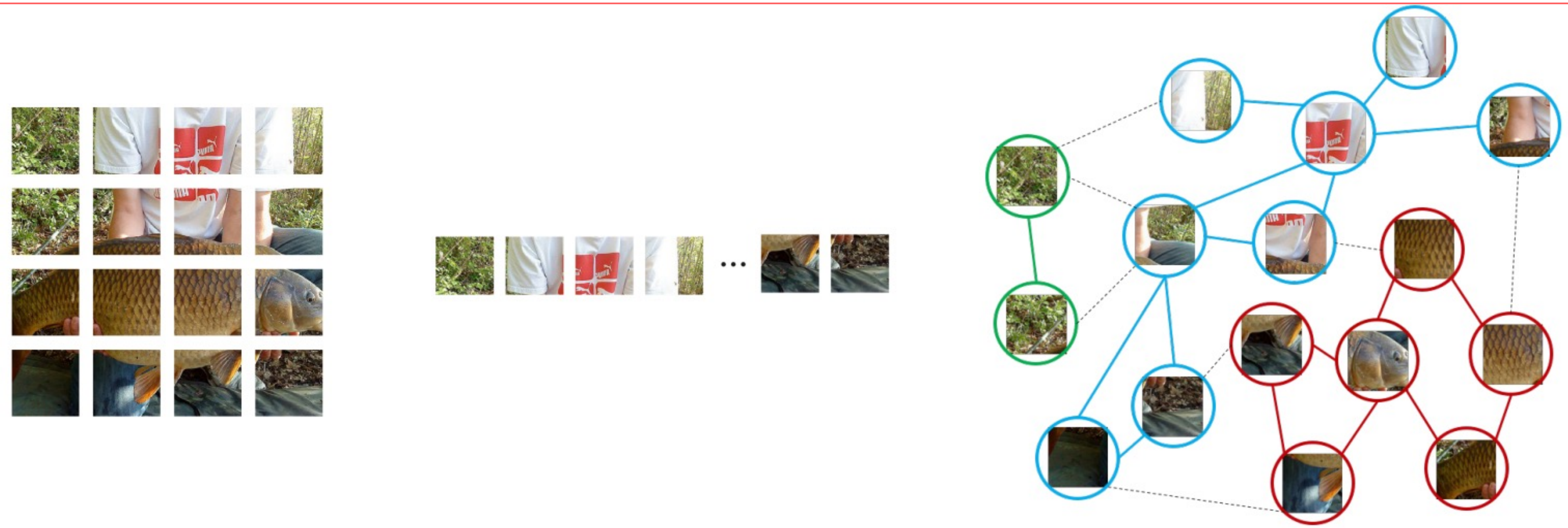
Abstract

Network architecture plays a key role in the deep learning-based computer vision system. The widely-used convolutional neural network and transformer treat the image as a grid or sequence structure, which is not flexible to capture irregular and complex objects. In this paper, we propose to represent the image as a graph structure and introduce a new *Vision GNN* (ViG) architecture to extract graph-level feature for visual tasks. We first split the image to a number of patches which are viewed as nodes, and construct a graph by connecting the nearest neighbors. Based on the graph representation of images, we build our ViG model to transform and exchange information among all the nodes. ViG consists of two basic modules: Grapher module with graph convolution for aggregating and updating graph information, and FFN module with two linear layers for node feature transformation. Both isotropic and pyramid architectures of ViG are built with different model sizes. Extensive experiments on image recognition and object detection tasks demonstrate the superiority of our ViG architecture. We hope this pioneering study of GNN on general visual tasks will provide useful inspiration and experience for future research.

The widely-used convolutional neural network and transformer treat the image as a grid or sequence structure, which is not flexible to capture irregular and complex objects

In this paper, we propose to represent the image as a graph structure and introduce a new Vision GNN (ViG) architecture to extract graph level feature for visual tasks

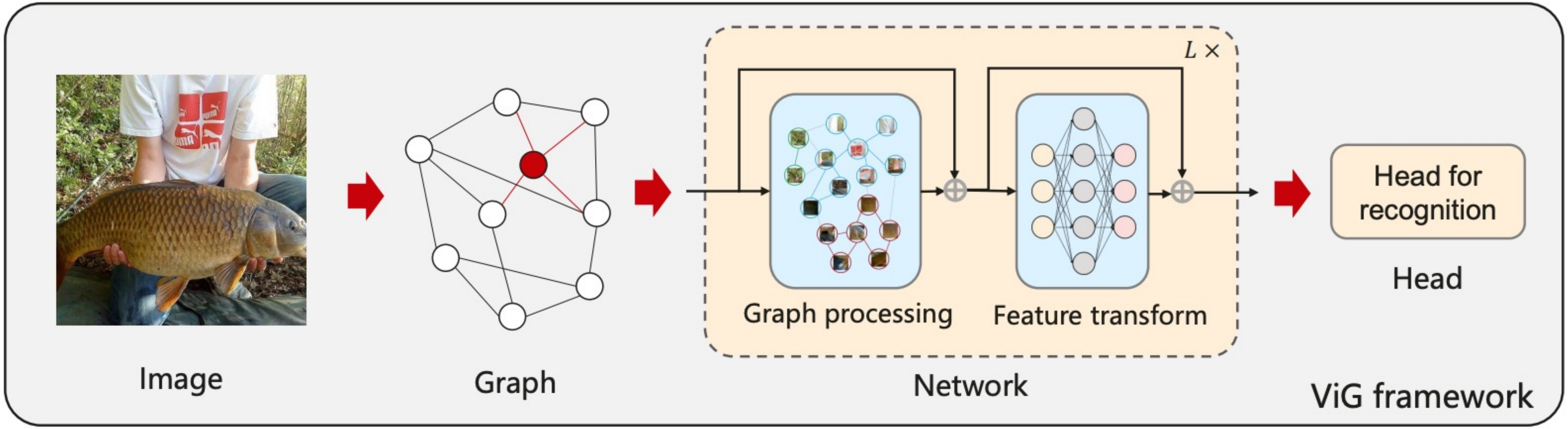
Vision GNN: Read and Understand



(a) Grid structure. (b) Sequence structure. (b) Graph structure.

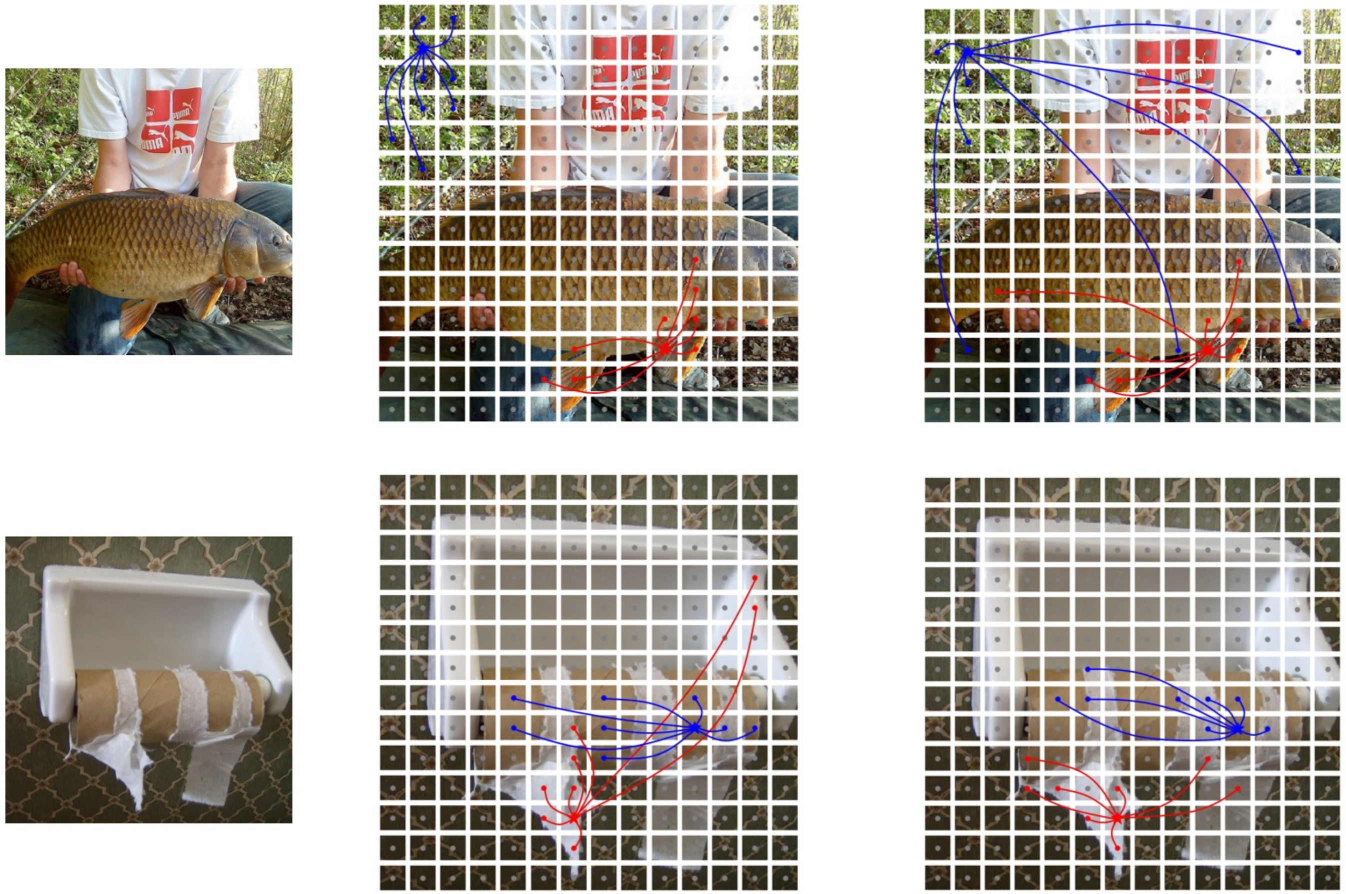
Illustration of the grid, sequence and graph representation of the image

Vision GNN: Read and Understand



The framework of the proposed ViG model

Vision GNN: Read and Understand

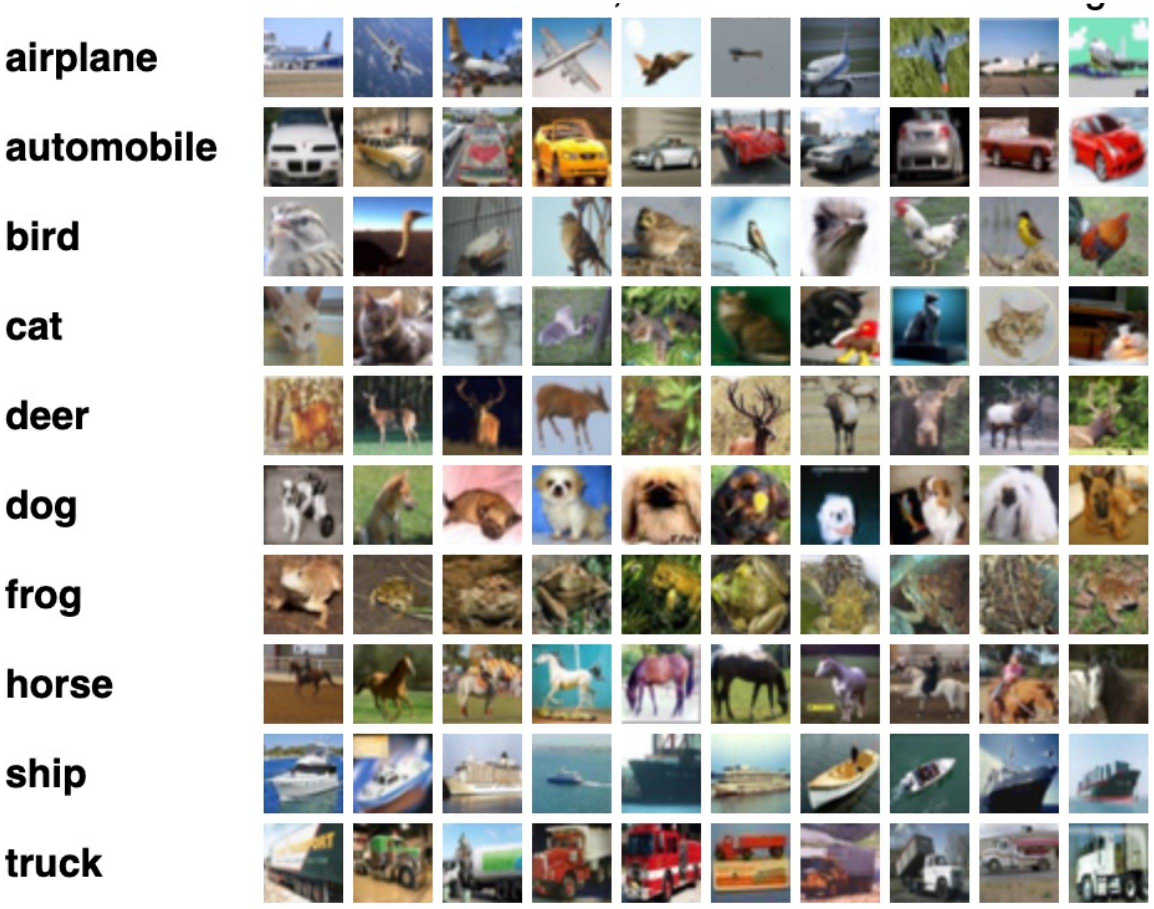


(a) Input image. (b) Graph connection in the 1st block. (c) Graph connection in the 12th block.

Visualization of the constructed graph structure. The pentagram is the center node, and the nodes with the same color are its neighbors in the graph

Example: CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.



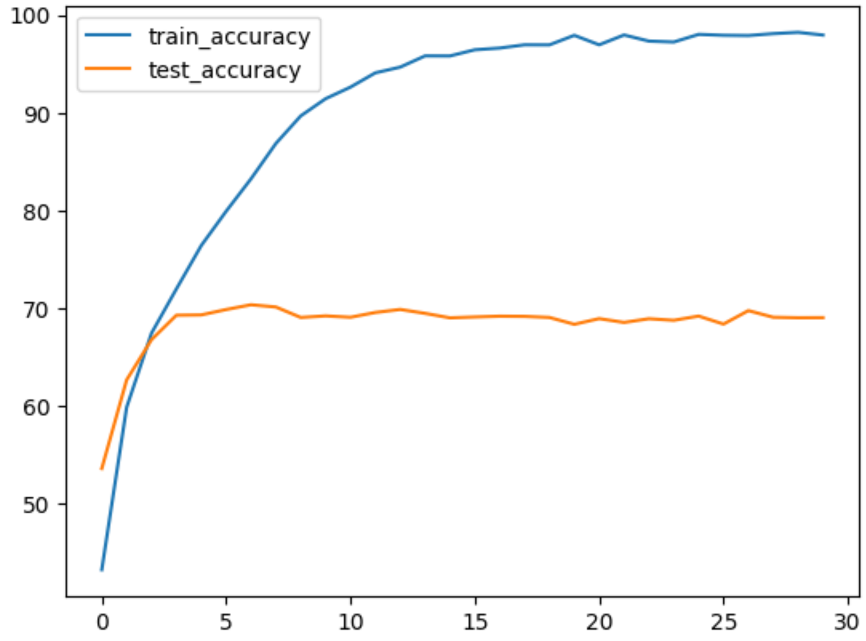
Example: CIFAR-10 Dataset

	#params	Eval Accuracy
CNN (LeNet)	5.6M	70.34
ViT	5.6M	48.29
ViG	5.7M	**74.93**

Example: CIFAR-10 Dataset

```python
class LeNet(nn.Module):
    def __init__(self, imdim=3, num_classes=10):
        super(LeNet, self).__init__()

        self.conv1 = nn.Conv2d(imdim, 64, kernel_size=5, stride=1, padding=0)
        self.mp = nn.MaxPool2d(2)
        self.relu1 = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=5, stride=1, padding=0)
        self.relu2 = nn.ReLU(inplace=True)
        self.fc1 = nn.Linear(128 * 5 * 5, 1024)
        self.relu3 = nn.ReLU(inplace=True)
        self.fc2 = nn.Linear(1024, 1024)
        self.relu4 = nn.ReLU(inplace=True)

        self.fc3 = nn.Linear(1024, 1024)
        self.fc4 = nn.Linear(1024, num_classes)

    def forward(self, x):
        in_size = x.size(0)
        out1 = self.mp(self.relu1(self.conv1(x)))
        out2 = self.mp(self.relu2(self.conv2(out1)))
        out2 = out2.view(in_size, -1)
        out3 = self.relu3(self.fc1(out2))
        out = self.relu4(self.fc2(out3))

        out = self.fc3(out)

        return self.fc4(out)
```

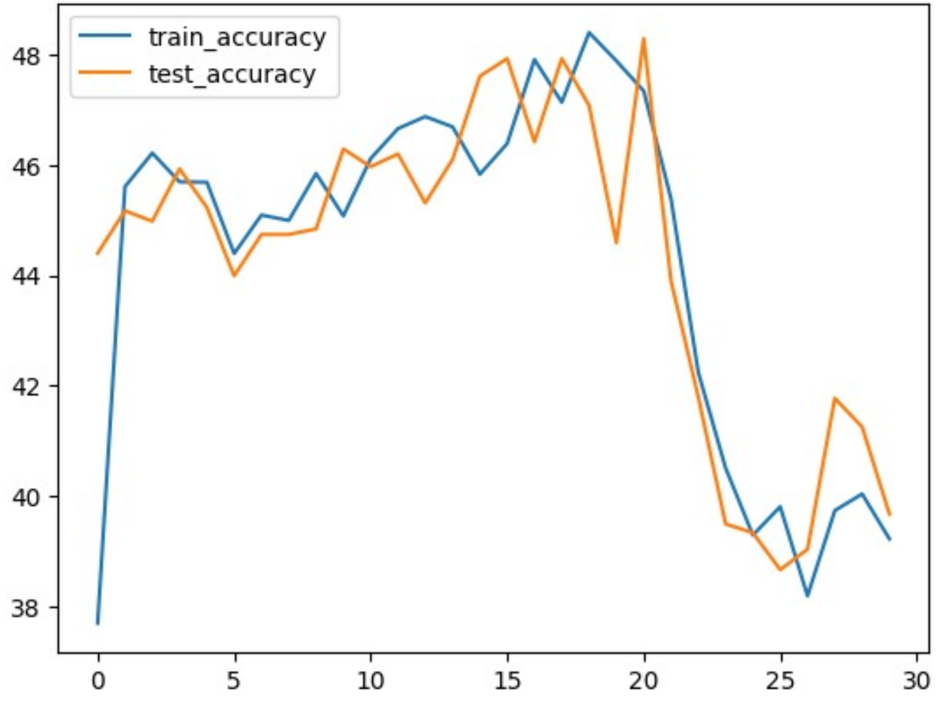
CNN for Image Classification



Example: CIFAR-10 Dataset

```
class ViT(nn.Module):
    def __init__(
        self,
        *,
        image_size,
        patch_size,
        num_classes,
        dim,
        depth,
        heads,
        mlp_dim,
        pool="cls",
        channels=3,
        dim_head=64,
        dropout=0.0,
        emb_dropout=0.0,
    ):
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)

        assert (
            image_height % patch_height == 0 and image_width % patch_width == 0
        ), "Image dimensions must be divisible by the patch size."

        num_patches = (image_height // patch_height) * (image_width // patch_width)
        patch_dim = channels * patch_height * patch_width
        assert pool in {
            "cls",
```

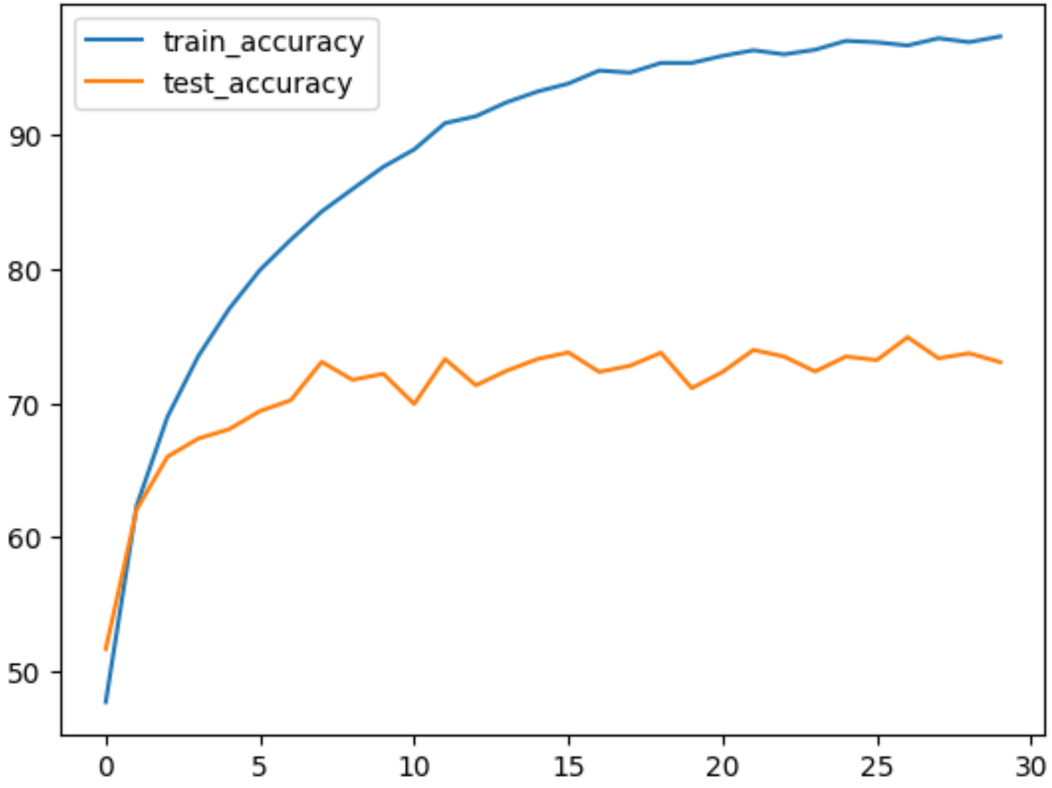
Vision Transformer for Image Classification



Example: CIFAR-10 Dataset

```python
# train
for epoch in range(max_epoch):
    model.train()
    running_loss = 0.0
    running_correct = 0  # to track number of correct predictions
    total = 0  # to track total number of samples

    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)

        loss = criterion(outputs, labels)

        running_loss += loss.item()

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        running_correct += (predicted == labels).sum().item()
```

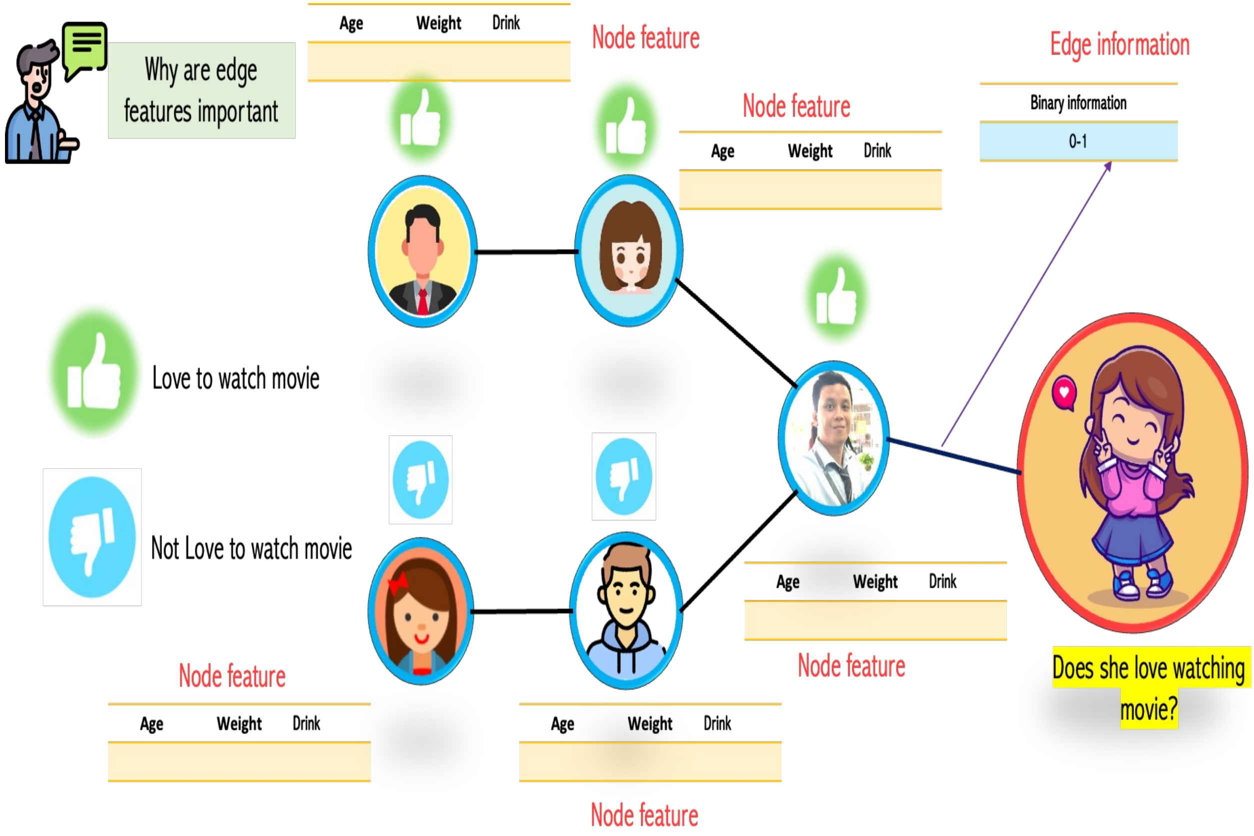
Vision GNN for Image Classification



Outline

- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

Summary



1. How to integrate edge feature to GNN
2. Edge Weight in GNN
3. Relational GNN
4. Multidimensional Edge Feature
4. Attention in GNN
4. Graph-level prediction


Thank you