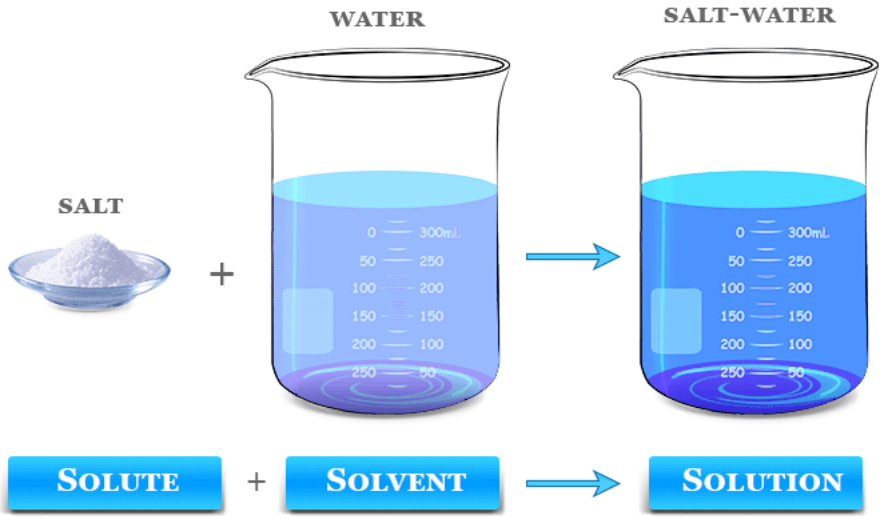
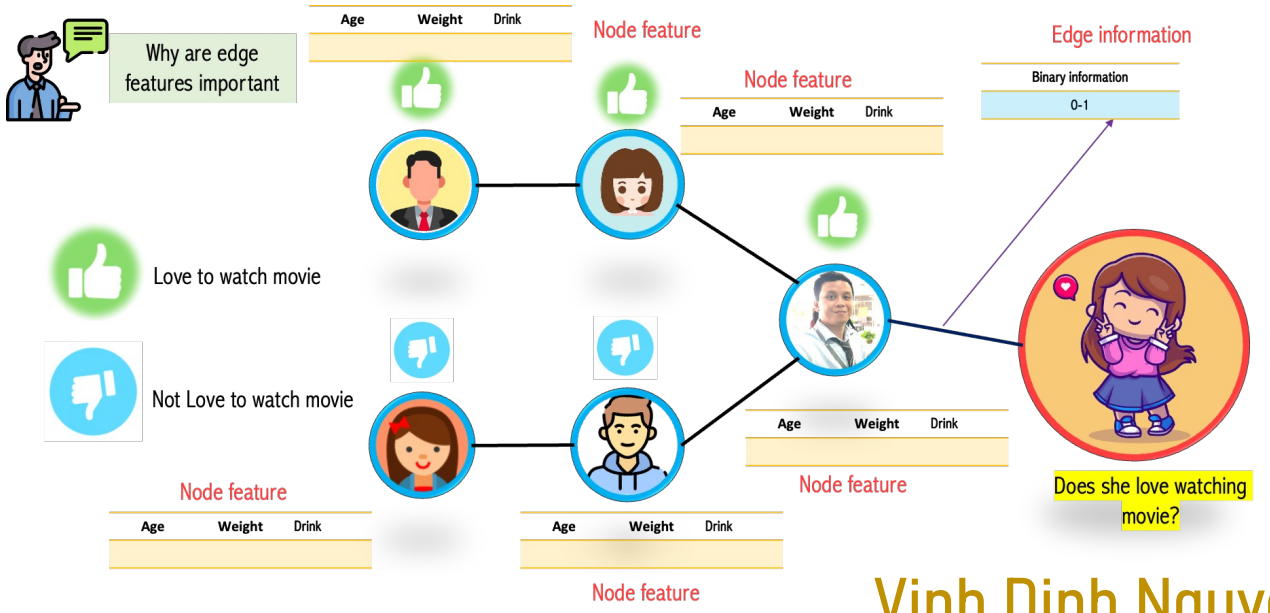
AI VIETNAM
All-in-One Course

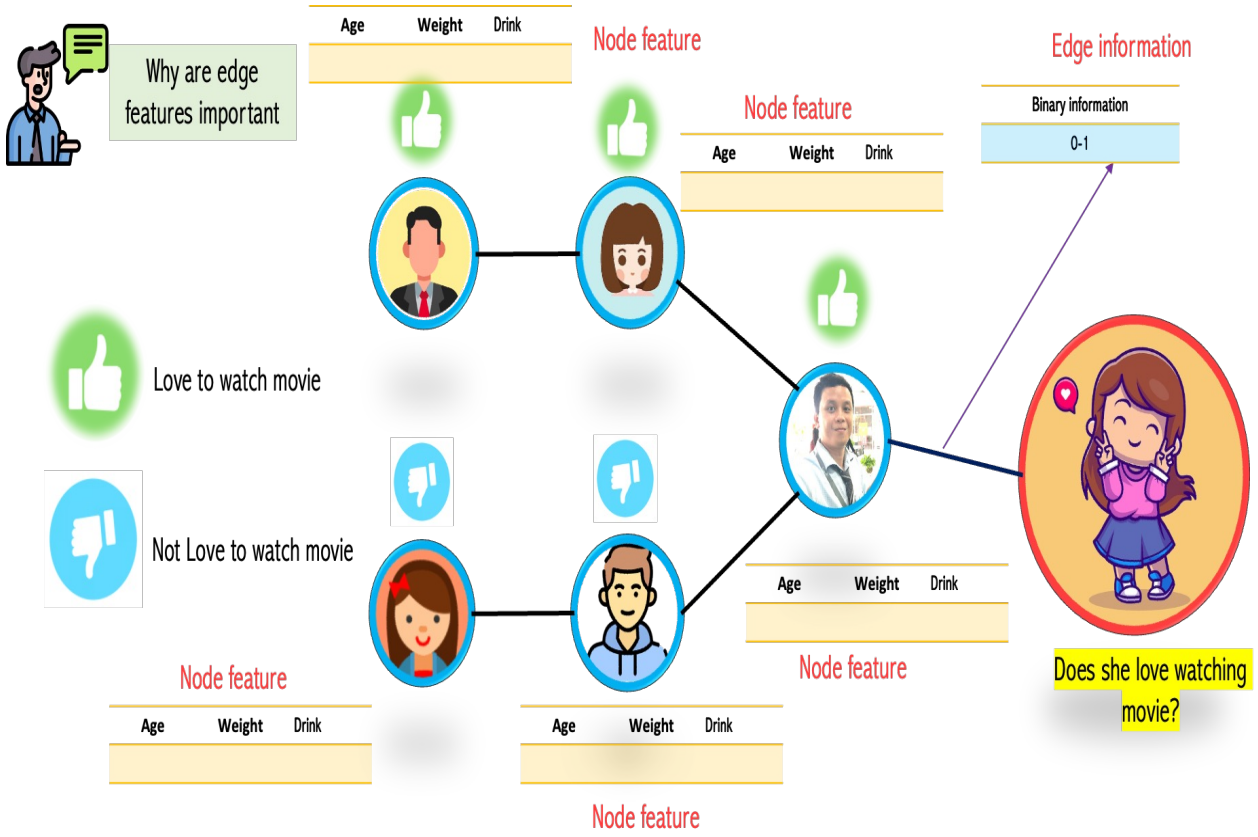
Advanced Graph Neural Network
(GCN, Graph Relational, Attention & Level-Prediction)





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

Why are edge features important

Love to watch movie

Not Love to watch movie

Node feature

Age	Weight	Drink

Node feature

Age	Weight	Drink

Edge information

Binary information
0-1

Node feature

Age	Weight	Drink

Node feature

Age	Weight	Drink

Node feature

Age	Weight	Drink

Does she love watching movie?

Edge Feature in GNN: Last But Not Least

Why are edge features important

Love to watch movie

Not Love to watch movie

Node feature

Age	Weight	Drink

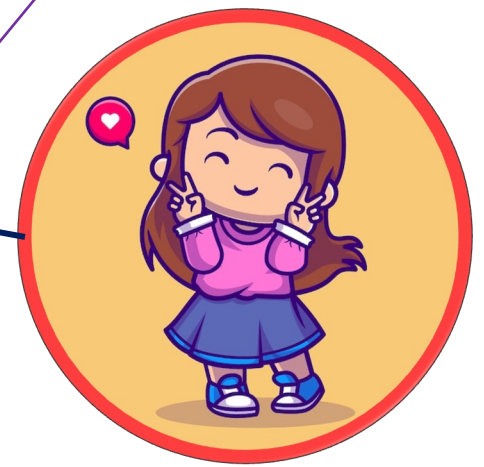
Node feature

Age	Weight	Drink

Edge information

Friend	Friend Since	Live together
Yes	9	No

How do edge features utilize in GNN?



Node feature

Age	Weight	Drink

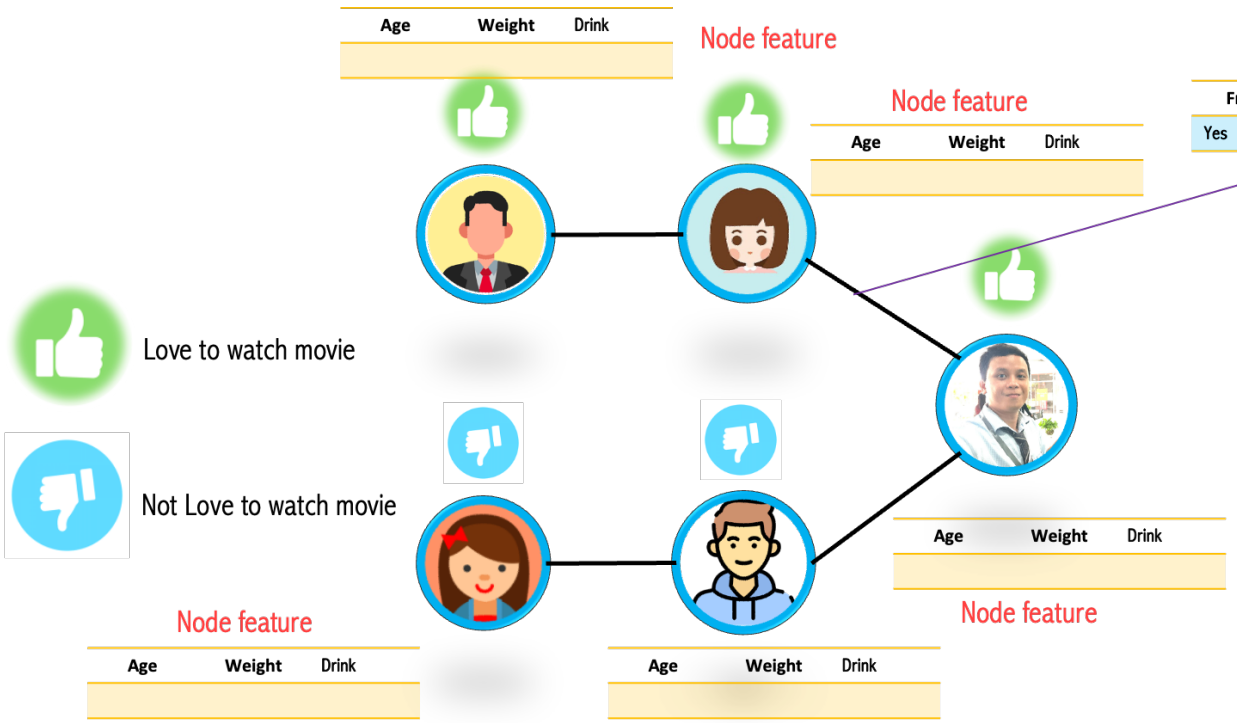
Node feature

Age	Weight	Drink

Node feature

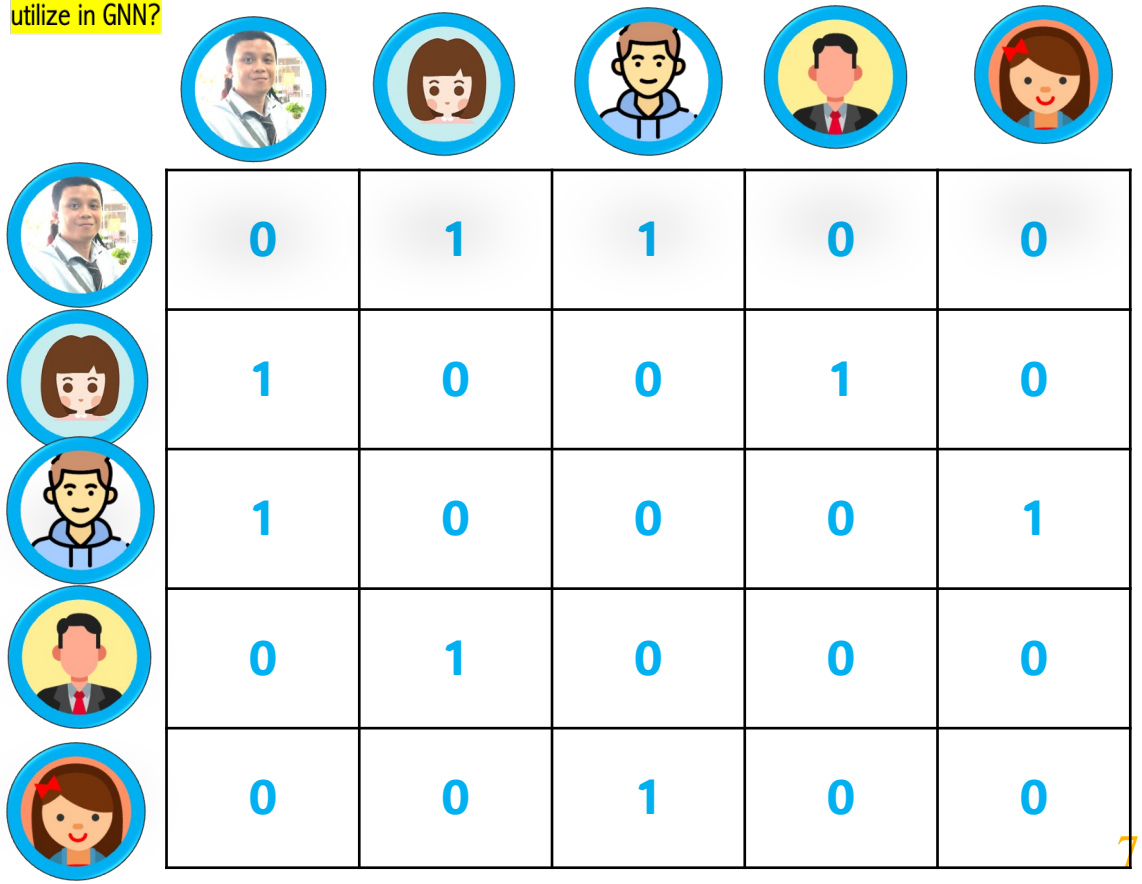
Age	Weight	Drink

Edge Feature in GNN



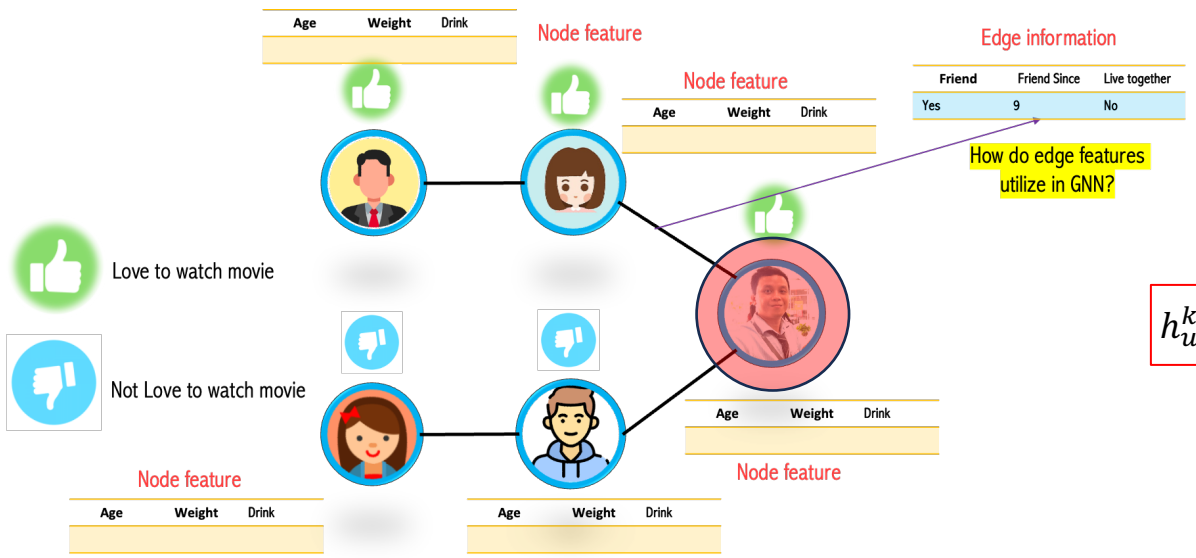
How to use Edge Feature:
Hot Research Topic

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

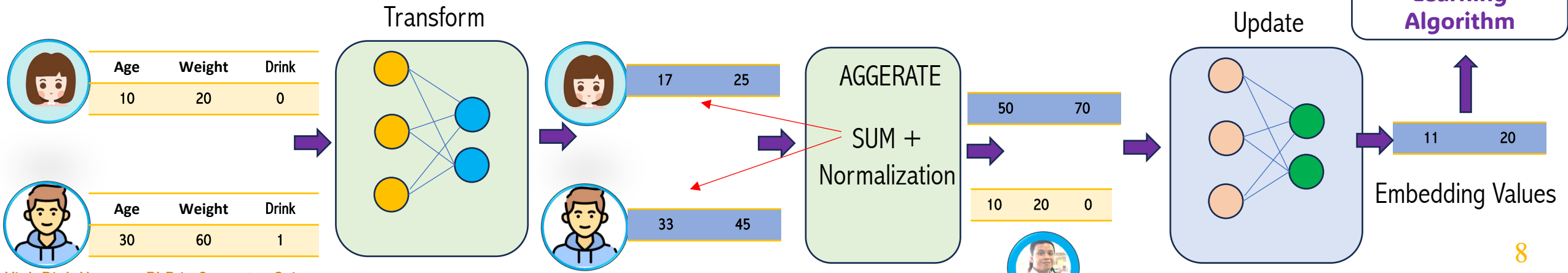
Node Embedding for Vinh Nguyen



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

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

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

Maching Learning Algorithm

Transform

Age	Weight	Drink
10	20	0

Age	Weight	Drink
30	60	1

17 25

33 45

AGGERATE

SUM + Normalization

50 70

10 20 0

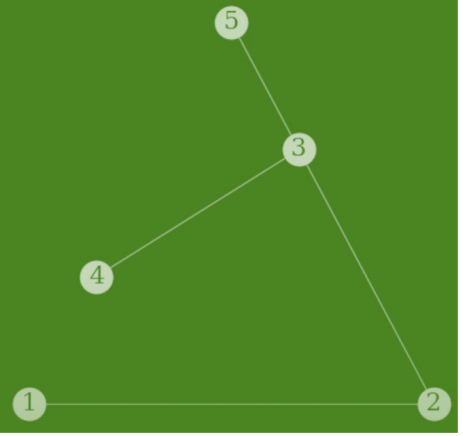
Update

11 20

Embedding Values

Example

	$Node_1$	$Node_2$	$Node_3$	$Node_4$	$Node_5$
$Node_1$	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



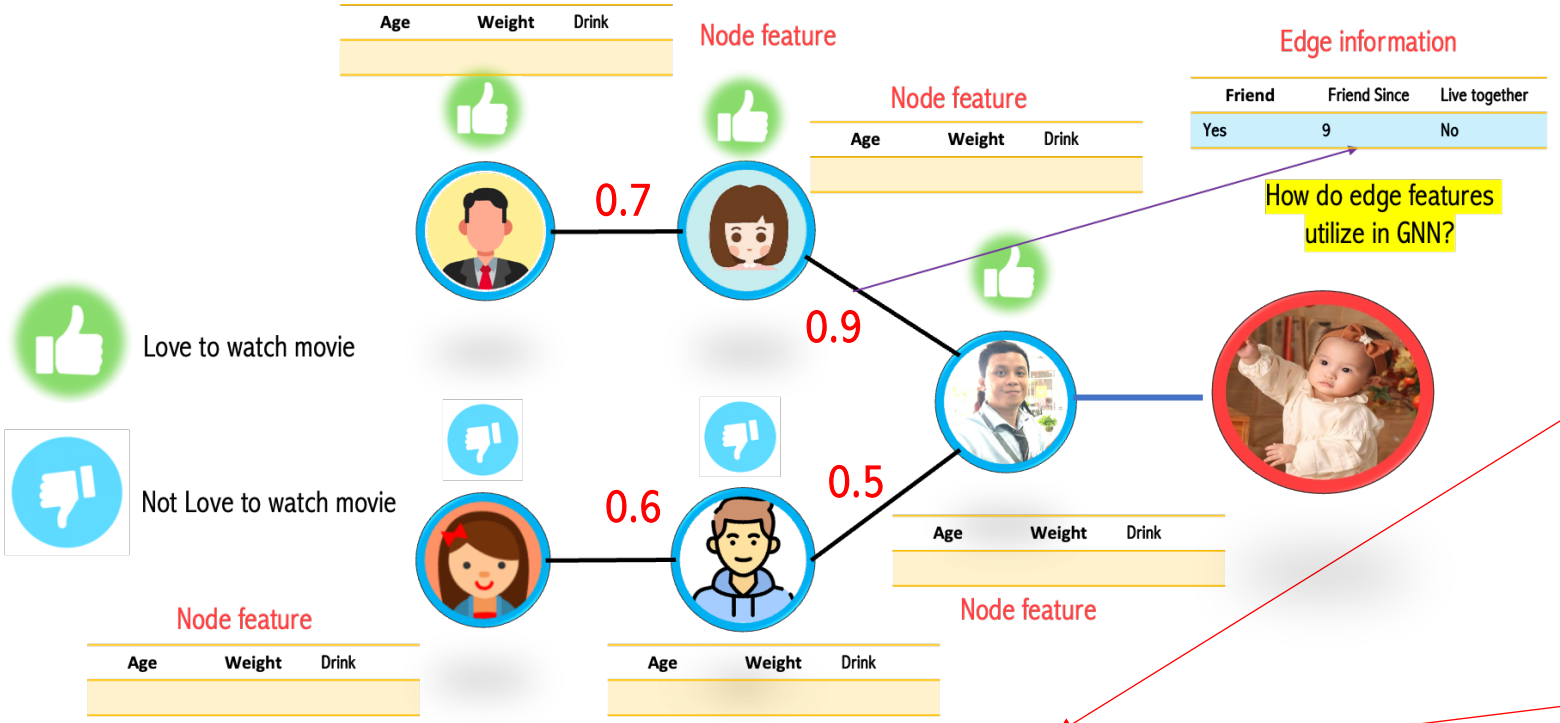
The '0' of the adjacency matrix cancel the contribution of all connected nodes, resulting a sum of just the connected nodes.

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

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

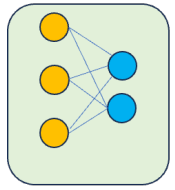
Adjacency matrix: $\tilde{A} = A + I_N$

Activation Function

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

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

Node feature

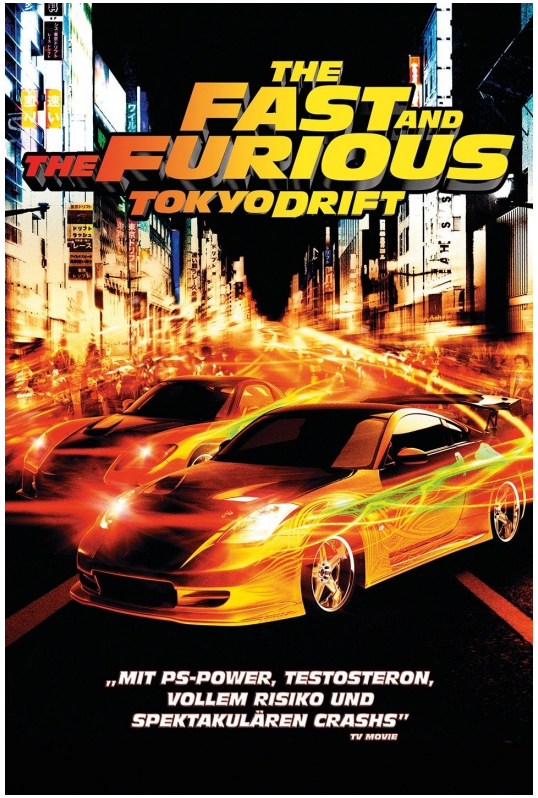


layer-specific trainable weight matrix

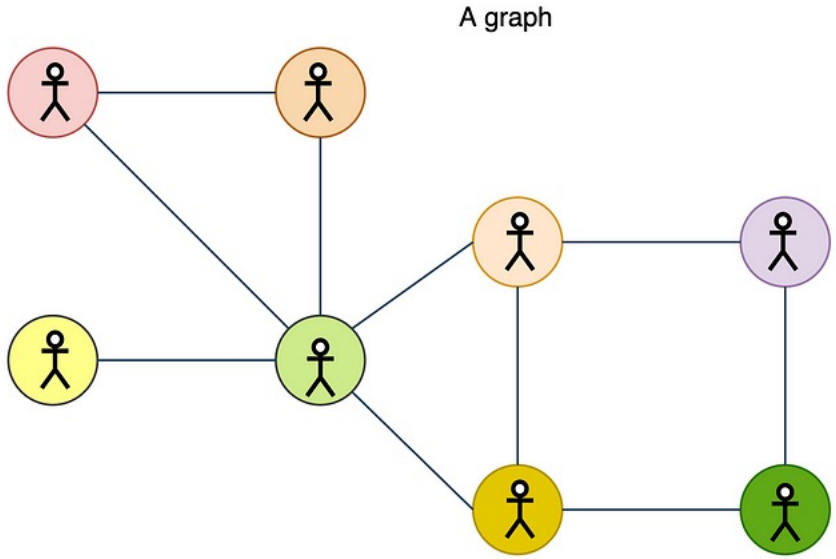
GNN: Review

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

The Fast and the Furious: Tokyo Drift.

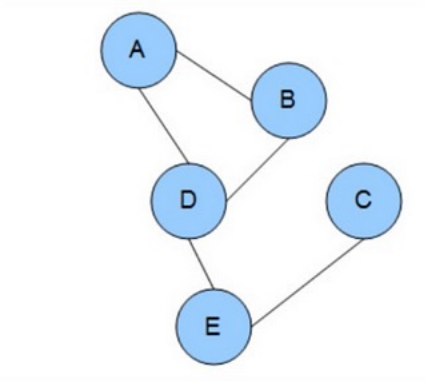


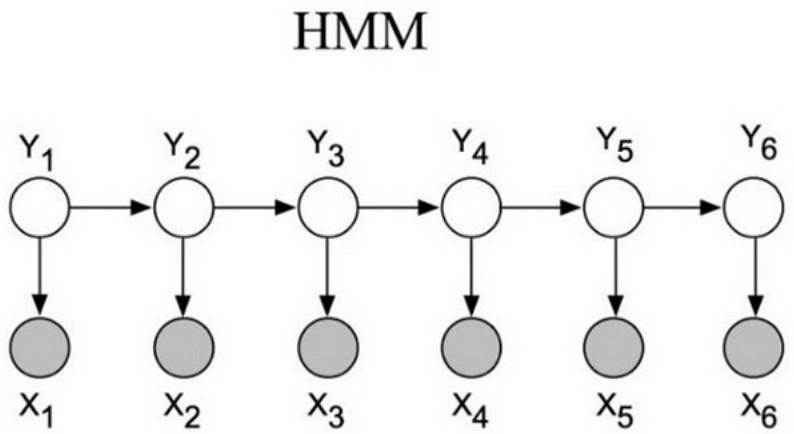
GNN: Review



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

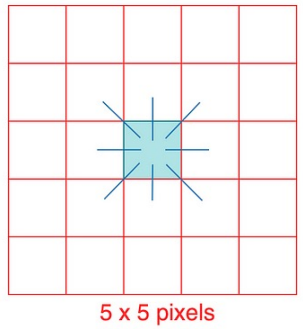
GNN: Review

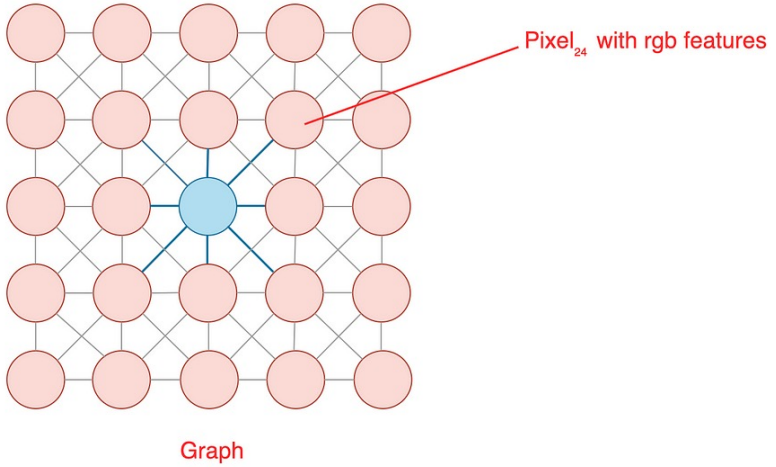




In speech recognition, the phoneme Y_i and the acoustic model x_i form an HMM (a graph for speech recognition).

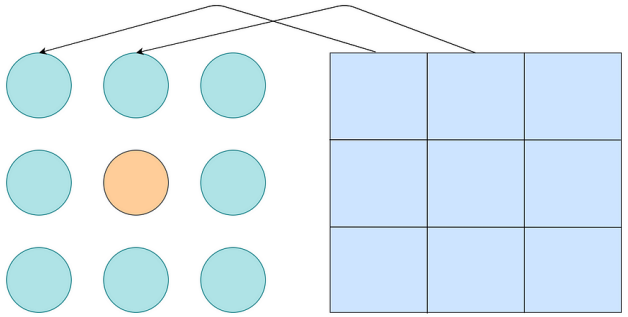
GNN: Review

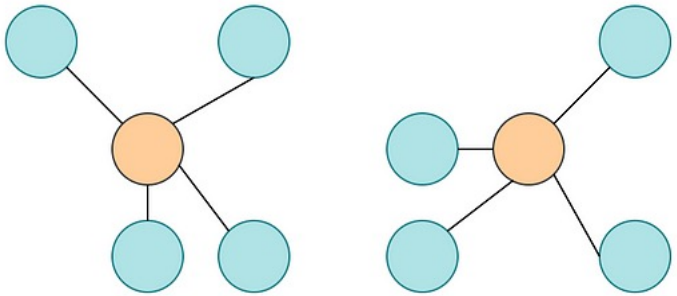




Even on CNN, an input image can be modeled as a graph. For example, the graph for a 5 × 5 image. Each node represents a pixel and for the case of a 3 × 3 filter, every node is connected to its eight immediate neighbors.

GNN: Review

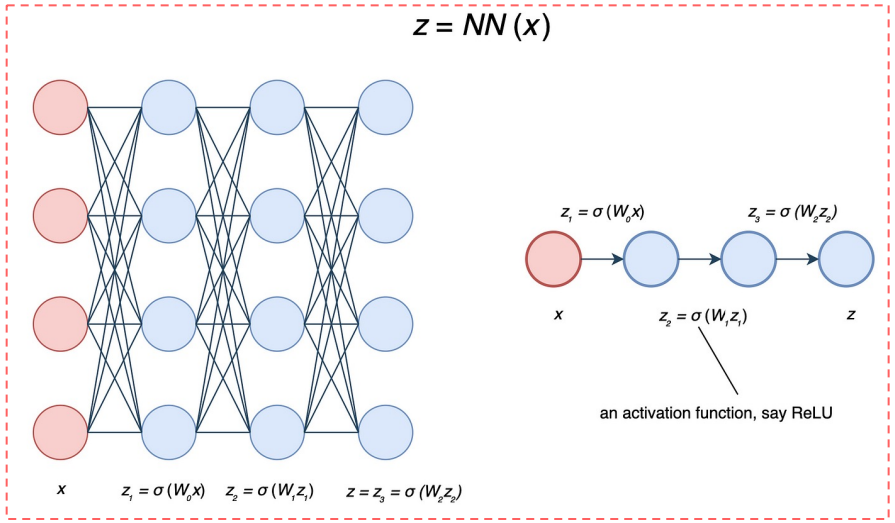


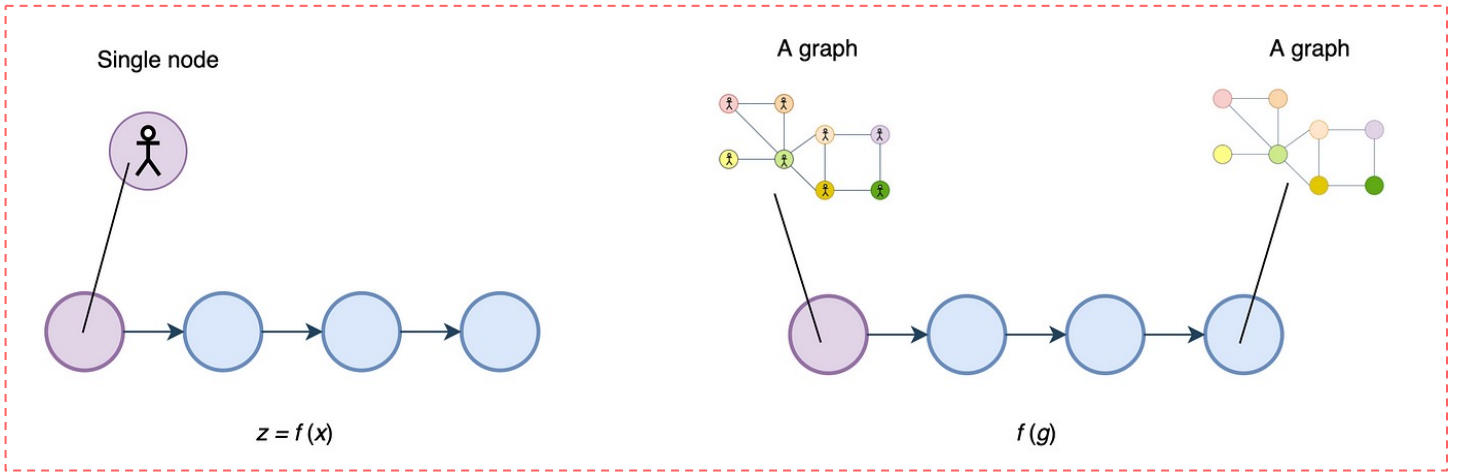


In particular, when the relationships between neighboring nodes are irregular and high dimensional, we need to define them explicitly in order to solve them efficiently. In CNN, we work in a Euclidean space. How weights are associated with the input features (pixels) is well defined.

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

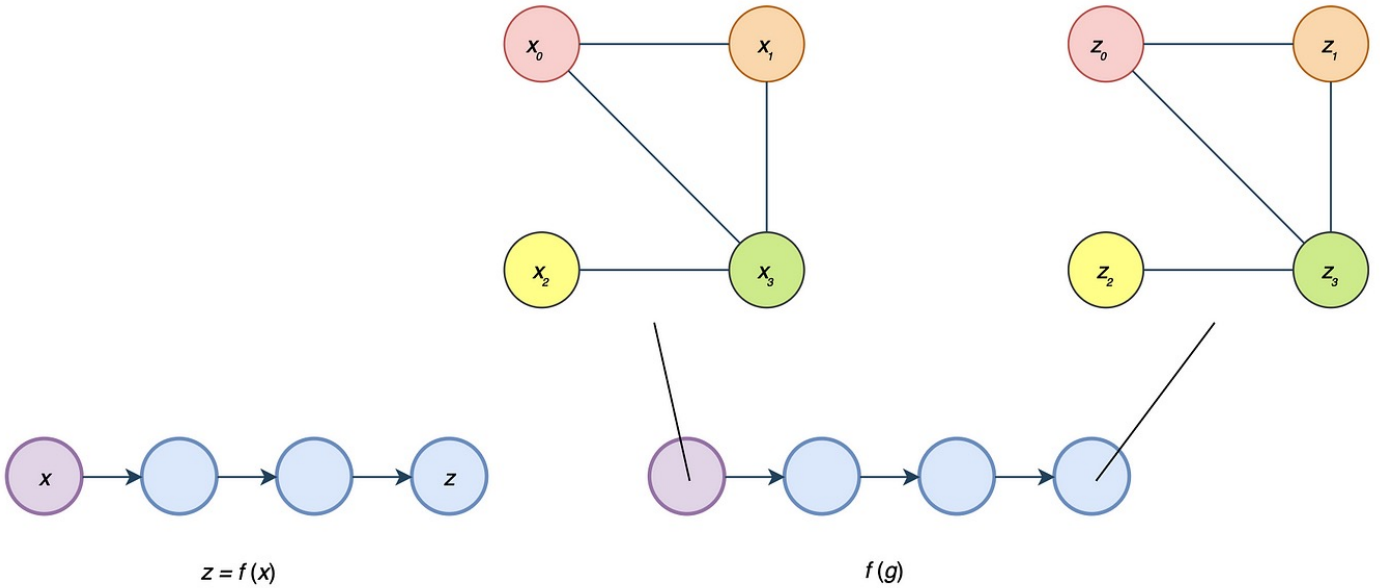




In general, neural networks (NNs) takes an input x to predict z.

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

GNN: Review

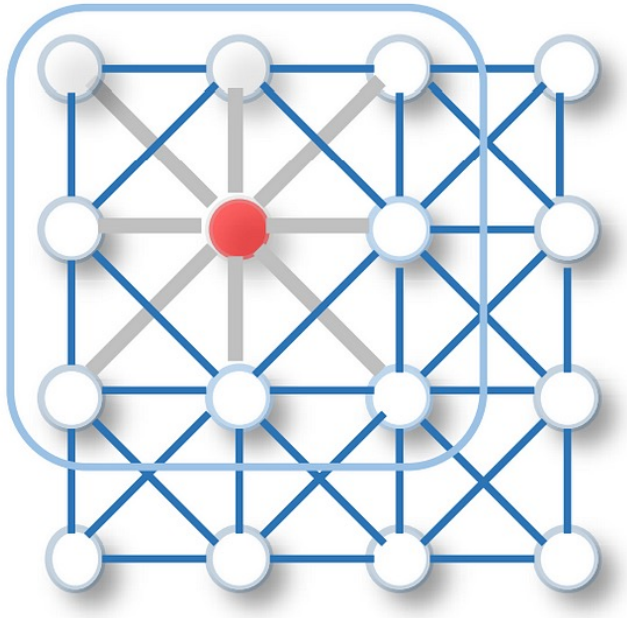


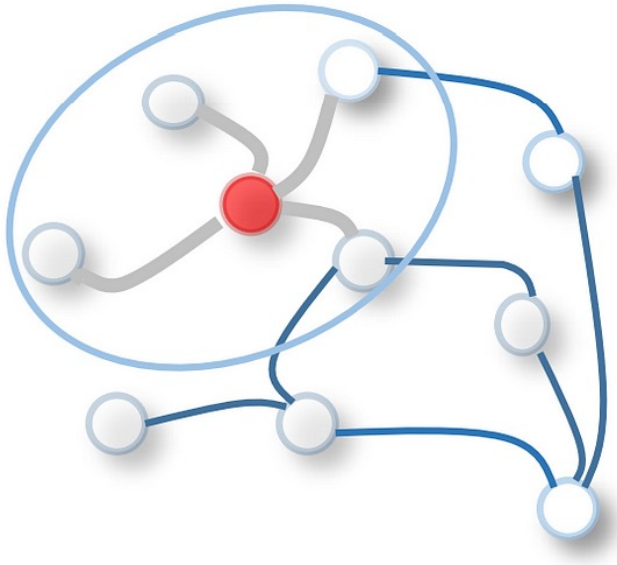
In GCN (Graph Convolutional Network), the input to the NN will be a graph. Also, instead of inferring a single z, it infers the value z_i for each node i in the graph. And to make predictions for Z_i, GCN utilizes both X_i and its neighboring nodes in the calculation.

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)

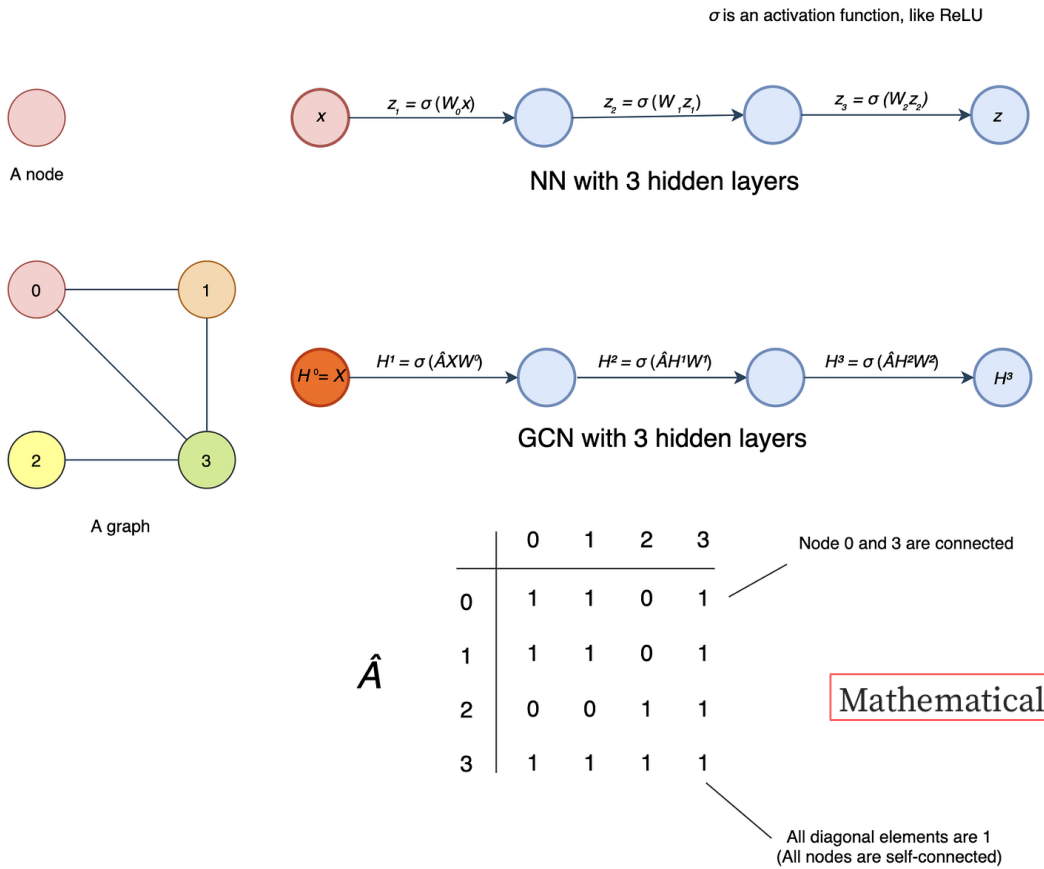
That comes to the output of the hidden layer to be $\sigma(\hat{A}H^iW^i)$. If we ignore W for a second, for each node in a hidden layer, $\hat{A}H^i$ sums up features on each node with its neighbors.

⬇

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 $\hat{D}^{-1}$ where $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$ in measuring the degree of each node

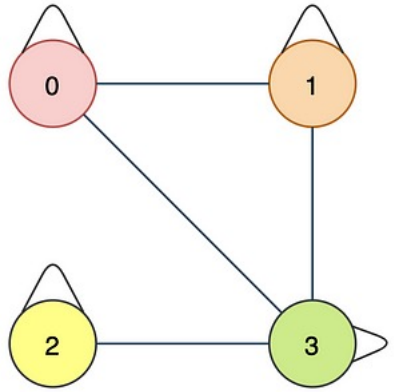
GNN: Review

$\hat{A}$

	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



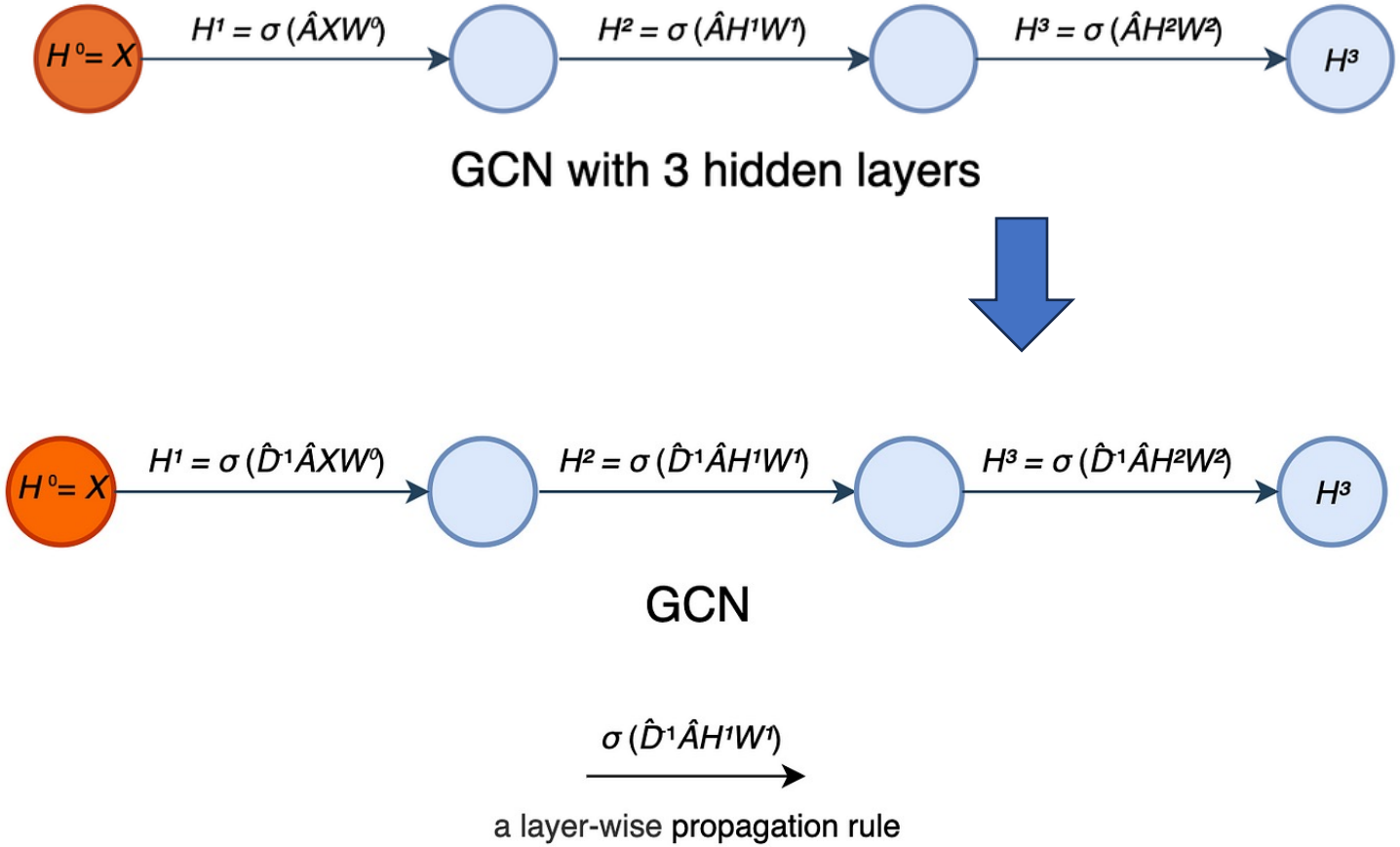
For an undirected graph, the degree of a node is counted as the number of times an edge terminates at that node. So a self-loop will count twice. In our example, node 0 has 2 edges connecting to its neighbors plus a self-loop. Its degree equals 4 (i.e. 2 + 2). For node 3, its degree equals 5 (3 + 2).

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

	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 $\sigma(\hat{D}^{-1}\hat{A}H^iW^i)$. The equation used to compute a hidden layer output from the last layer output is called the **propagation rule**.

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

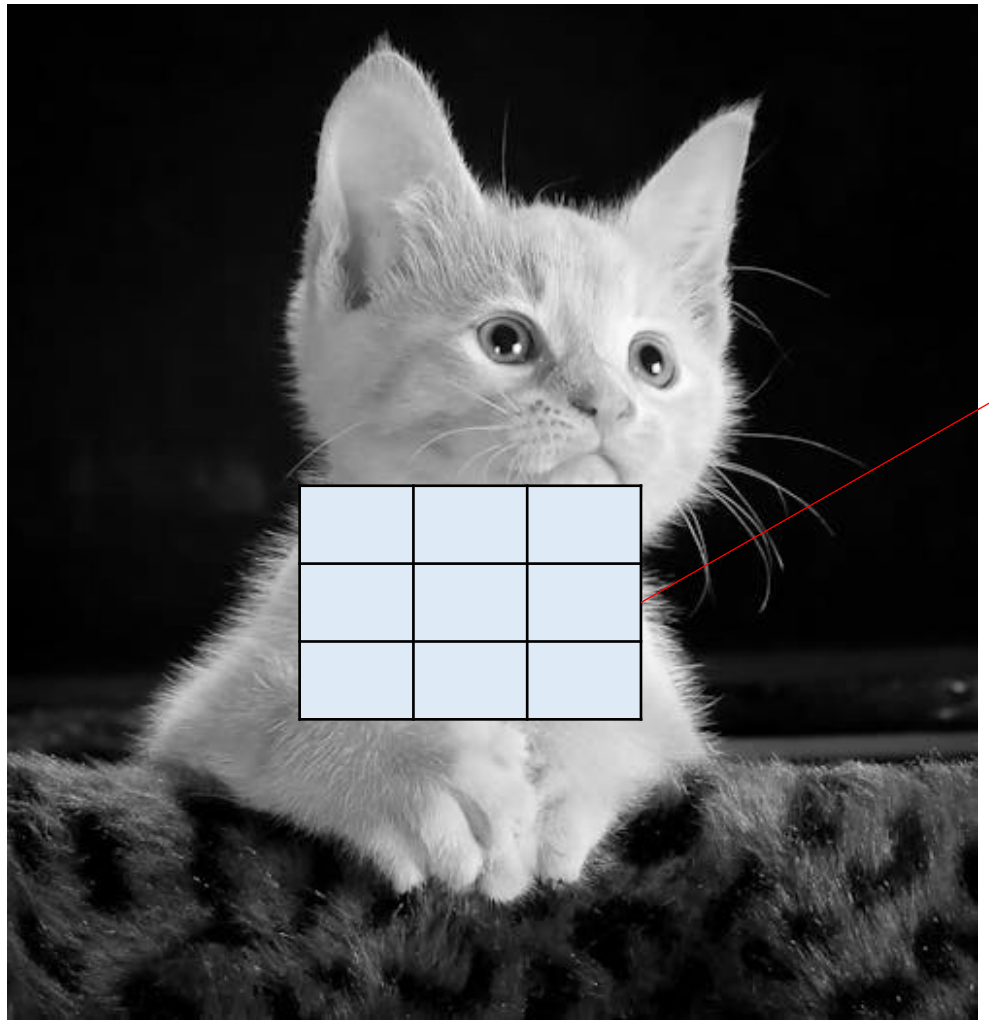
$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where

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

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

From CNN to GNN



Convolutional Operation

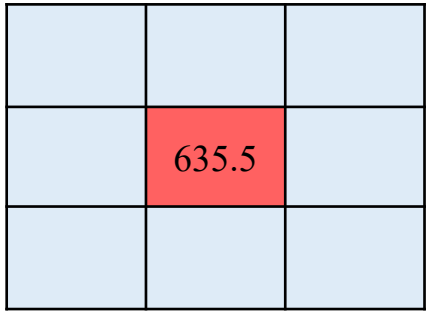
ROI

48	109	57
17	52	126
13	13	64

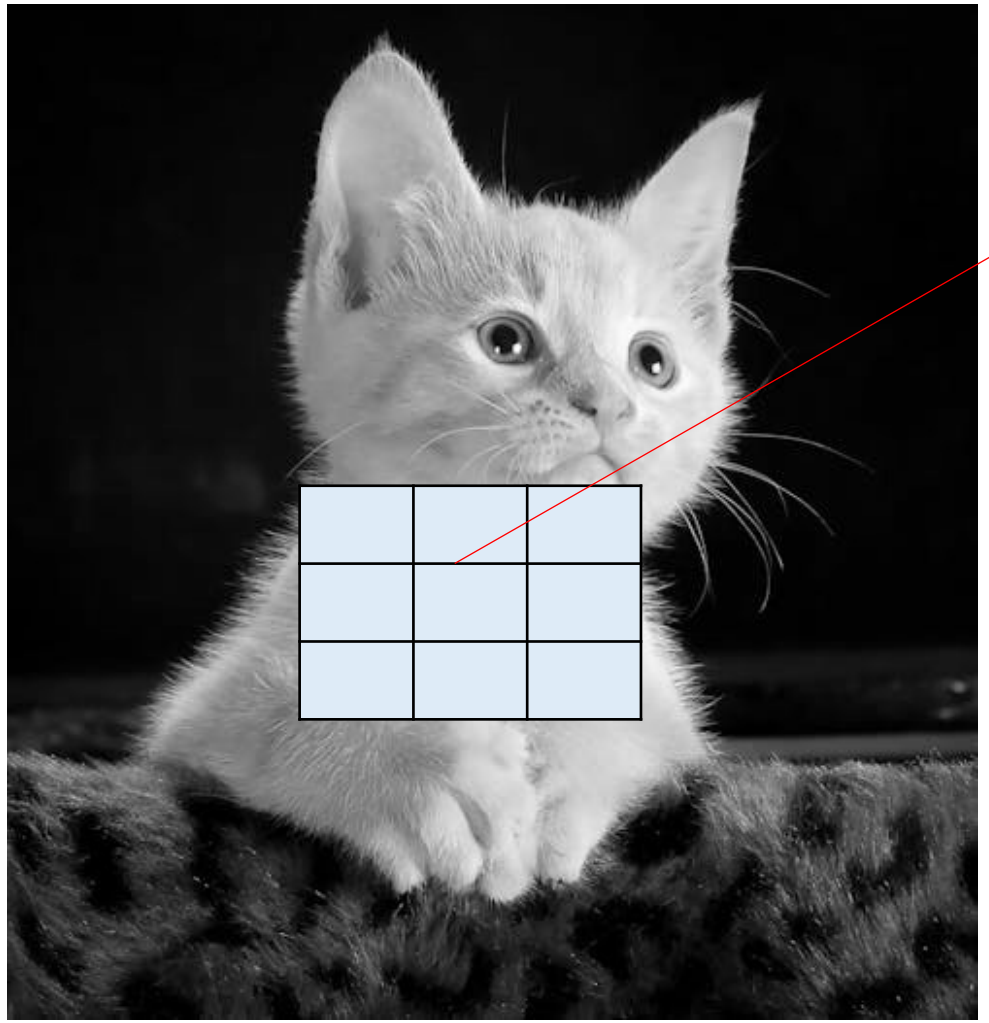
Kernel

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

$0.12\times48 + 0.58\times109 + 1.17\times57 - 0.44\times17 + 3.11\times52 - 0.8\times126 + 5.11\times13 - 0.31\times13 + 4.17\times64 = 635.5$

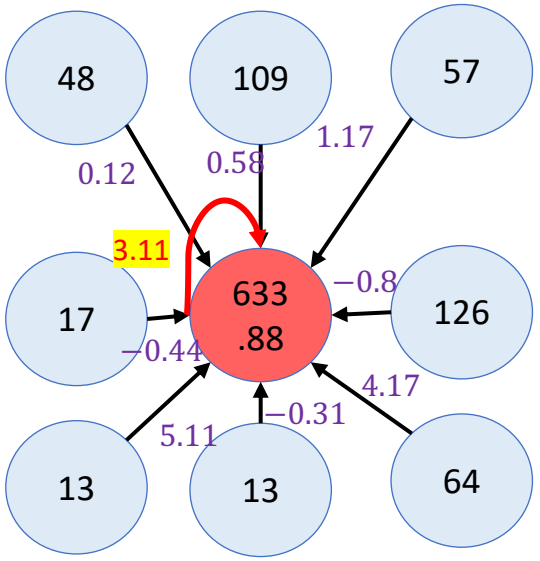


From CNN to GNN



ROI

48	109	57
17	52	126
13	13	64

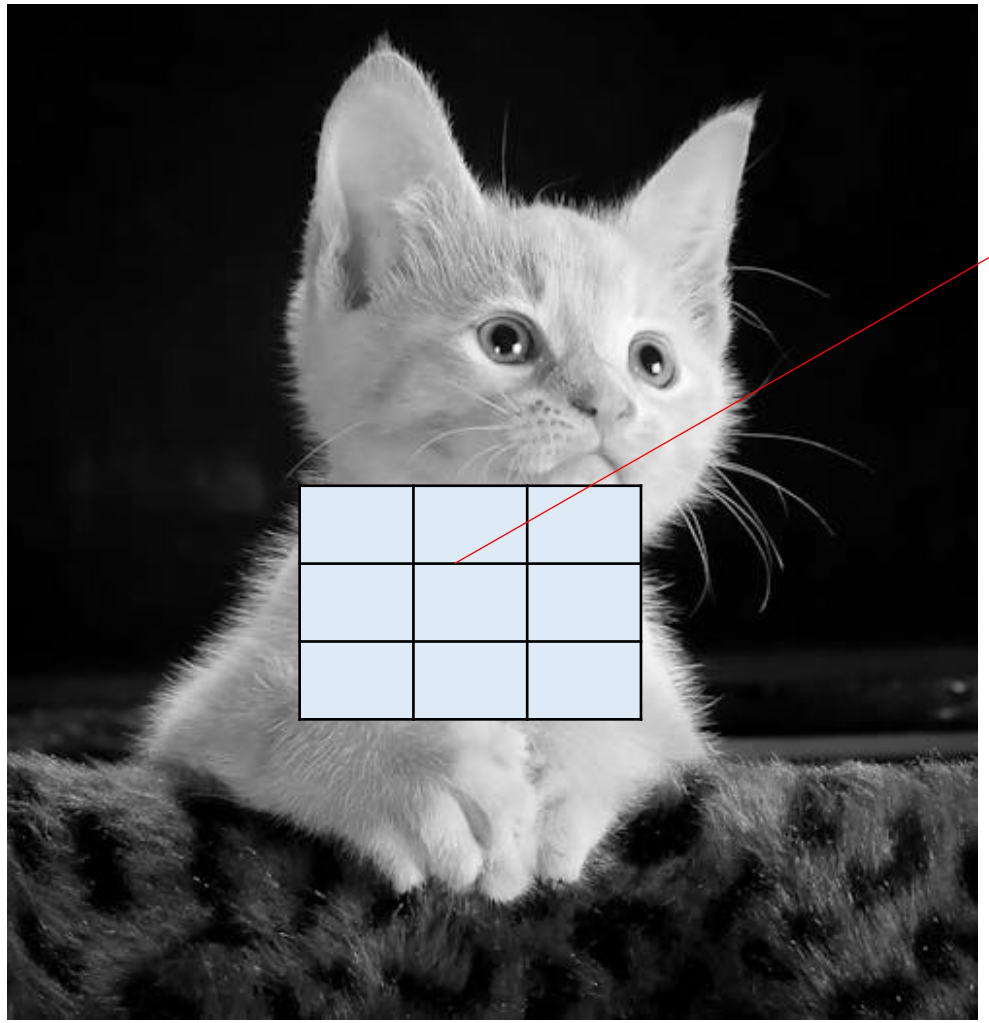


Kernel

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

$0.12\times48 + 0.58\times109 + 1.17\times57 - 0.44\times17 - 0.8\times126 + 5.11\times13 - 0.31\times13 + 4.17\times64 = 633.88$

From CNN to GNN

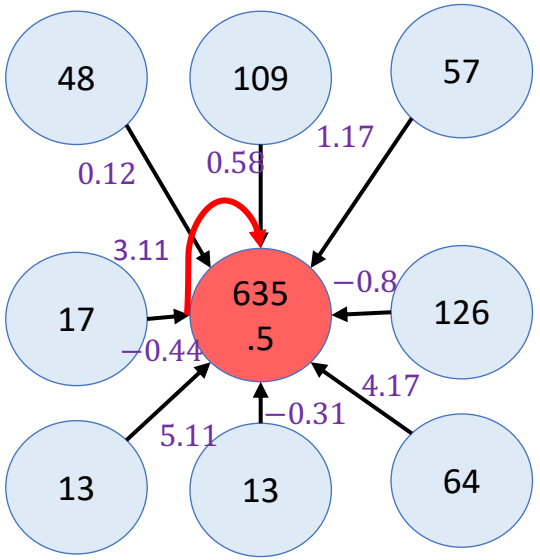


ROI

48	109	57
17	52	126
13	13	64

Kernel

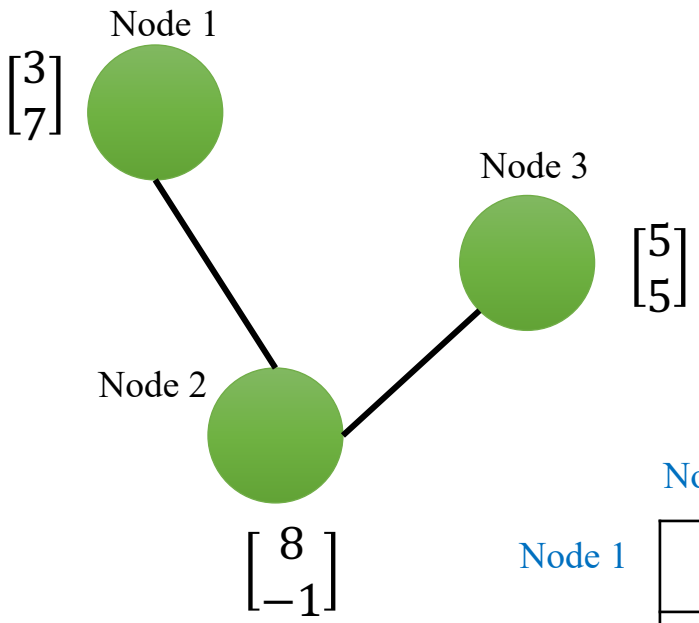
$$\begin{bmatrix} 0.12 & 0.58 & 1.17 \\ -0.44 & 3.11 & -0.8 \\ 5.11 & -0.31 & 4.17 \end{bmatrix}$$



$0.12\times48 + 0.58\times109 + 1.17\times57 - 0.44\times17 - 0.8\times126 + 5.11\times13 - 0.31\times13 + 4.17\times64 = 633.88$

$0.12\times48 + 0.58\times109 + 1.17\times57 - 0.44\times17 + 3.11\times52 - 0.8\times126 + 5.11\times13 - 0.31\times13 + 4.17\times64 = 635.5$

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

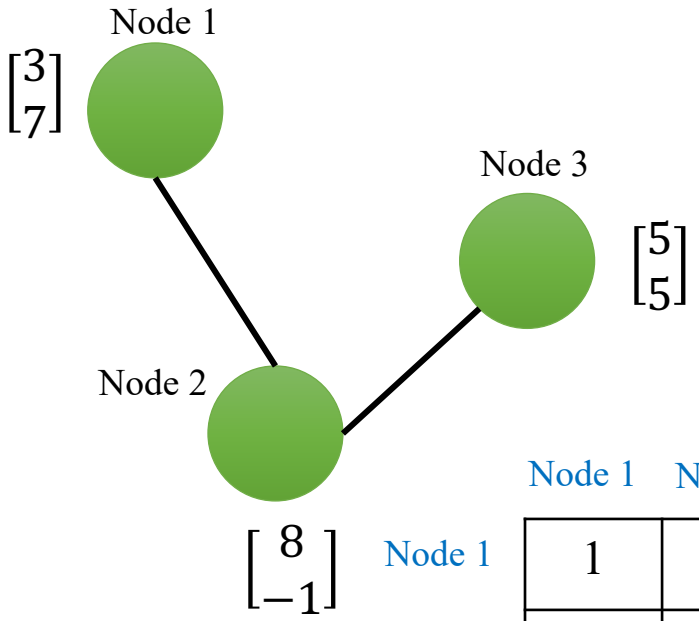
=

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

×

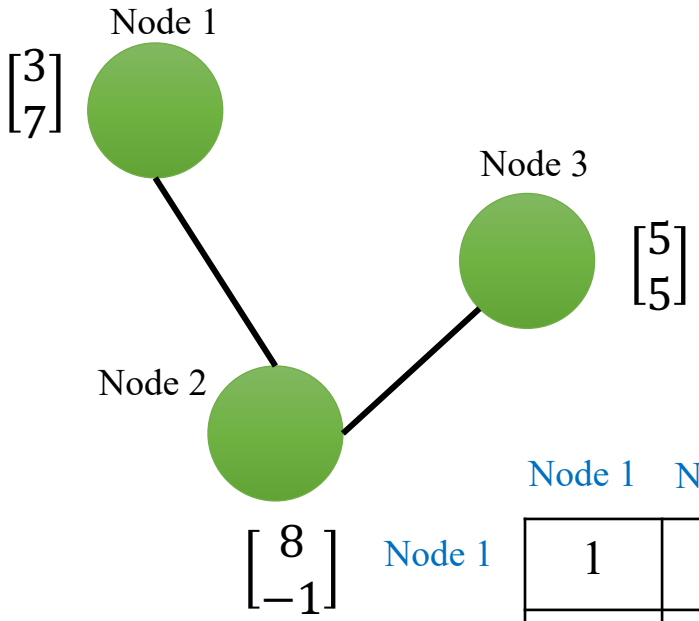
W11	W12
W21	W22

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

	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

×

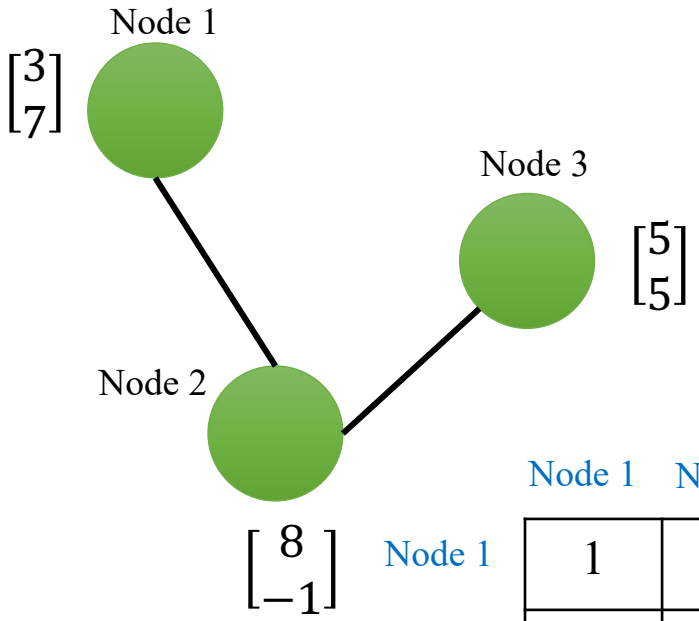
W11
W21

W: weight matrix

=

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

From CNN to GNN



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

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

Activation Function

Adjacency matrix:

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
8	-1
5	5

Node 1
Node 2
Node 3

H: node feature

×

W11
W21

W: weight matrix

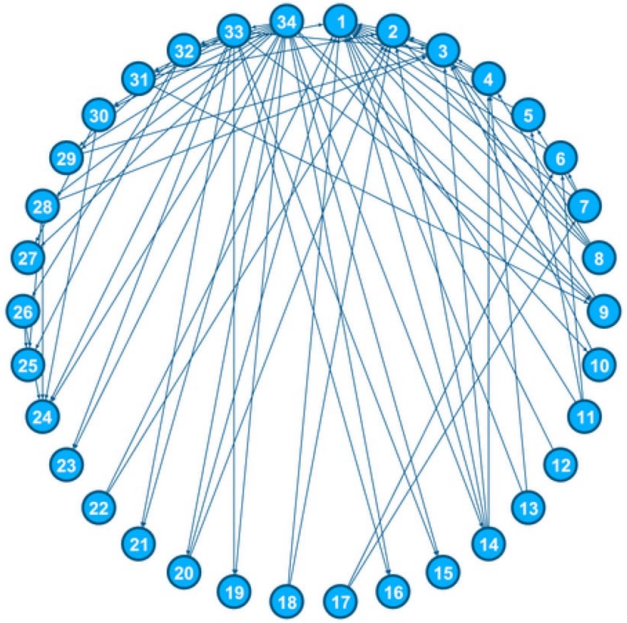
=

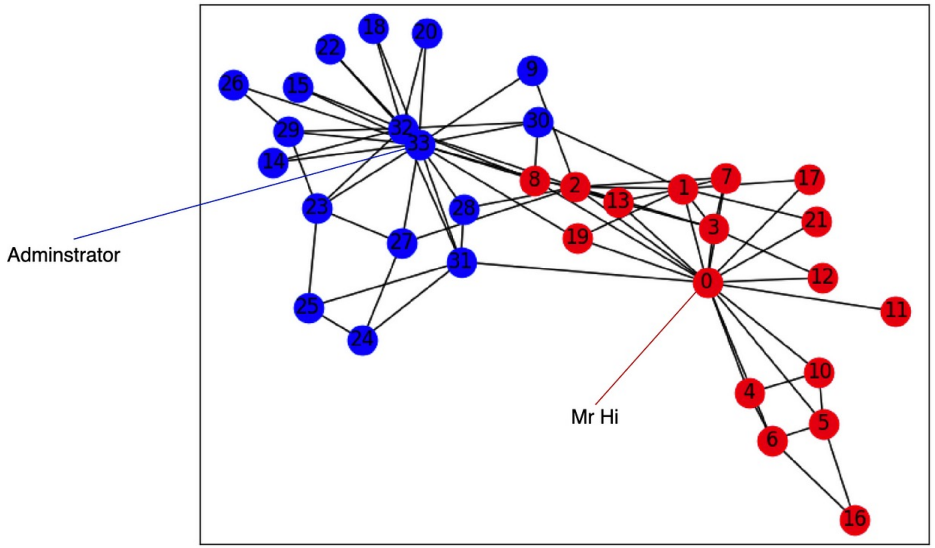
(3+8)W11 + (7-1)W21
(3+8+5)W11+ (7-1+5)W21
(8+5) W11 + (-1+5)W21

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

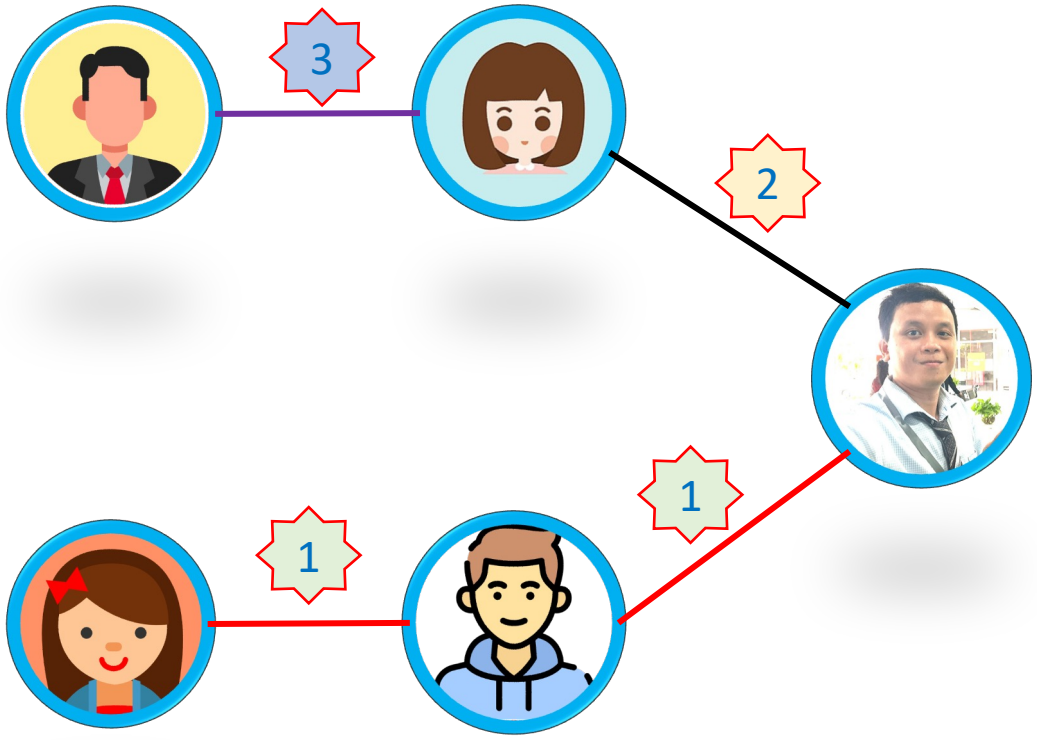
Zachary's karate club

```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



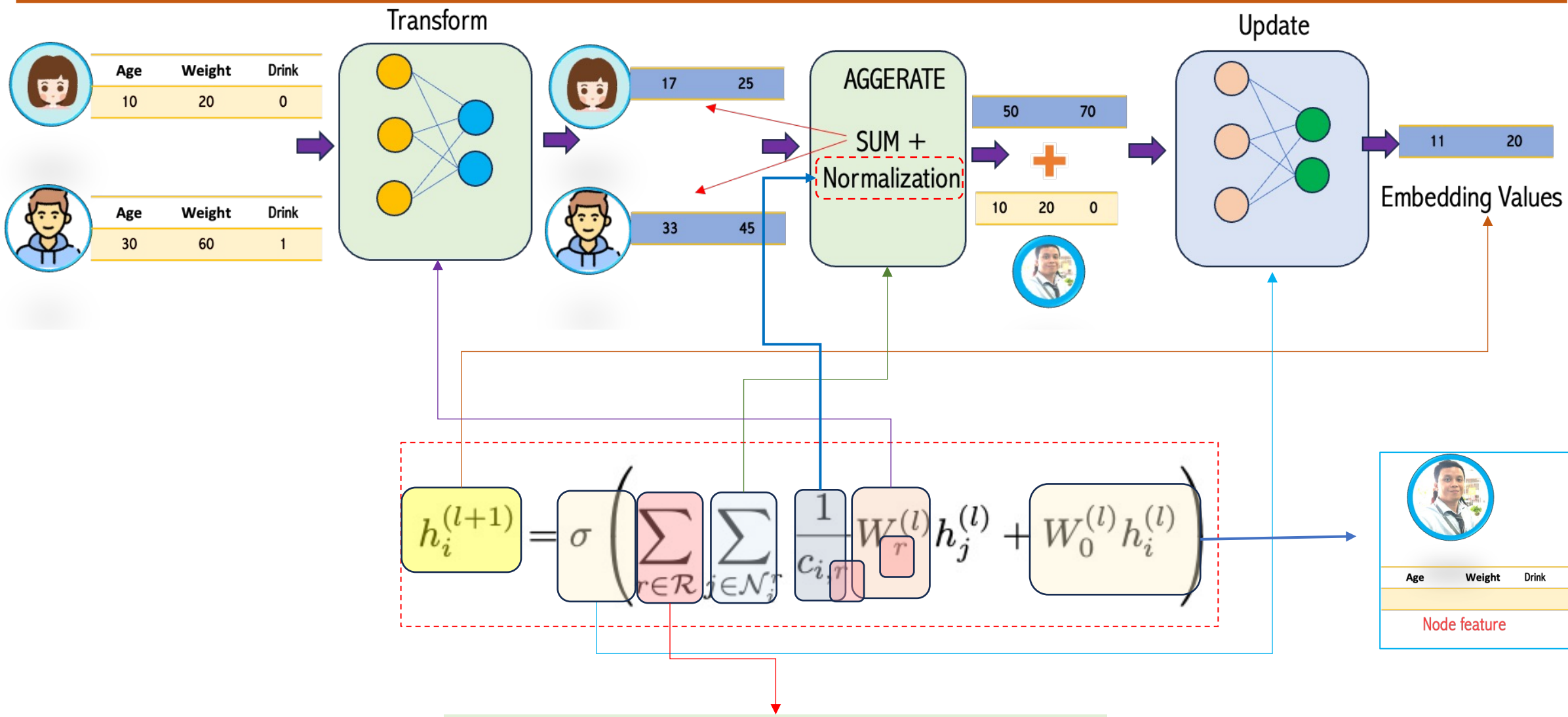
Relational Graph Convolutional Neural Network

1 Friend 2 Family 3 Colleagues

Outline

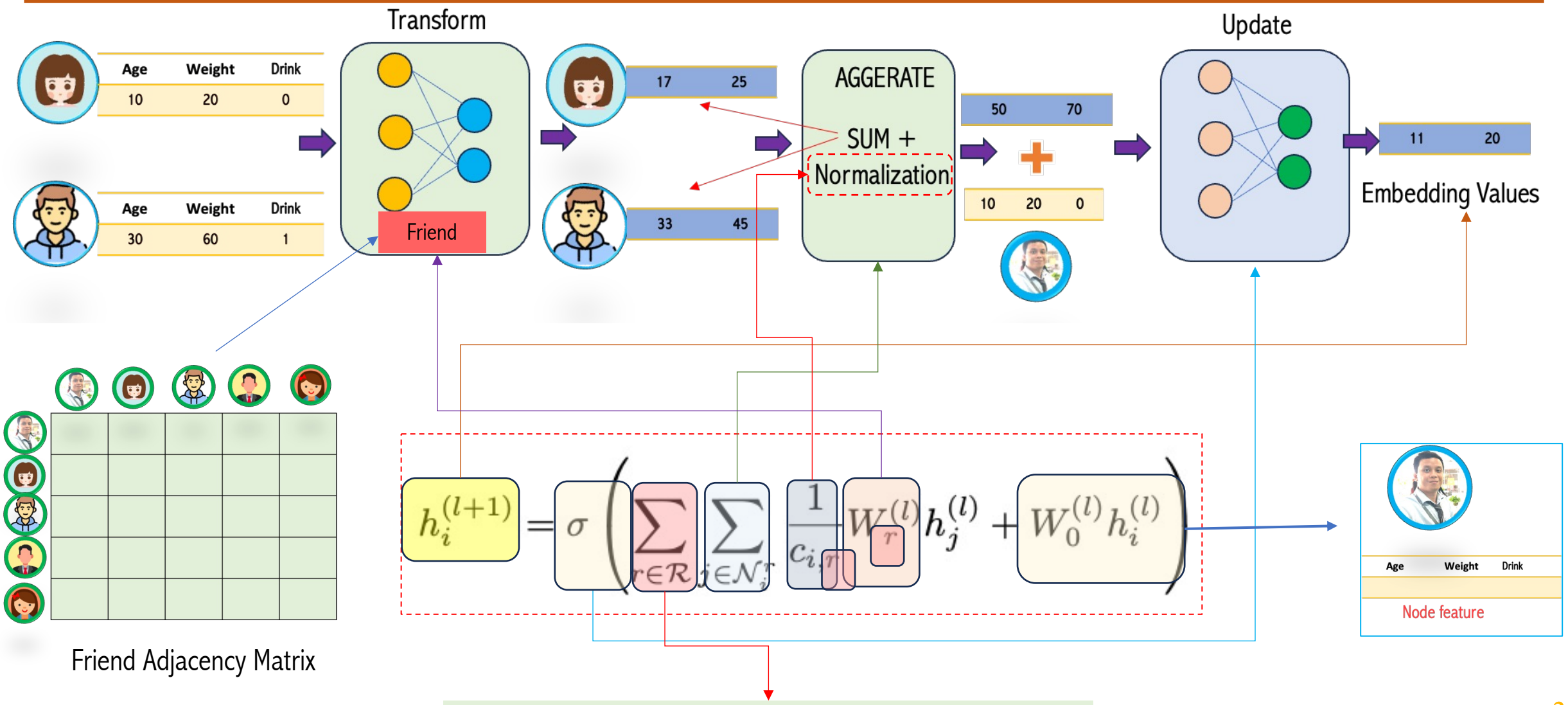
- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

Relational GNN



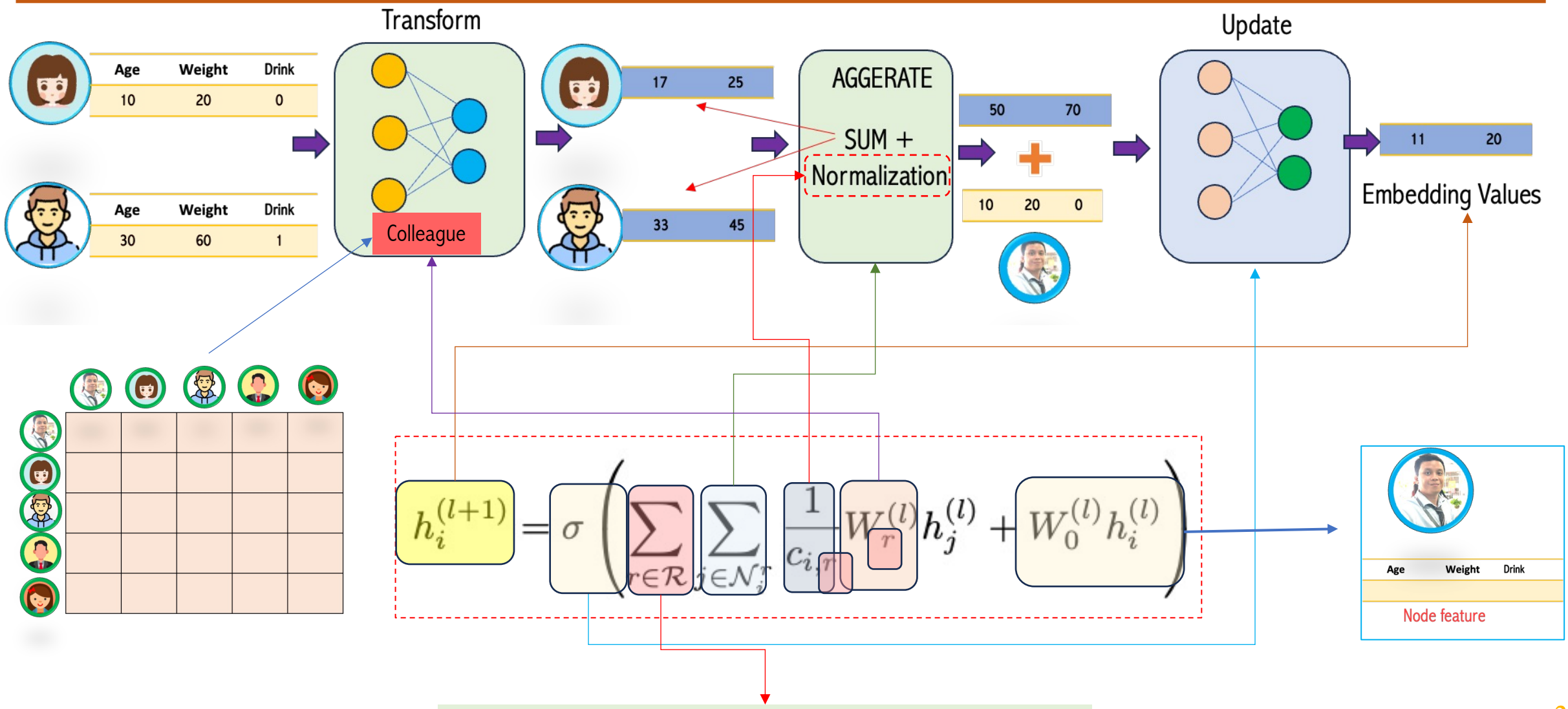
Various types of relationship: Edge conditions GNN

Relational GNN



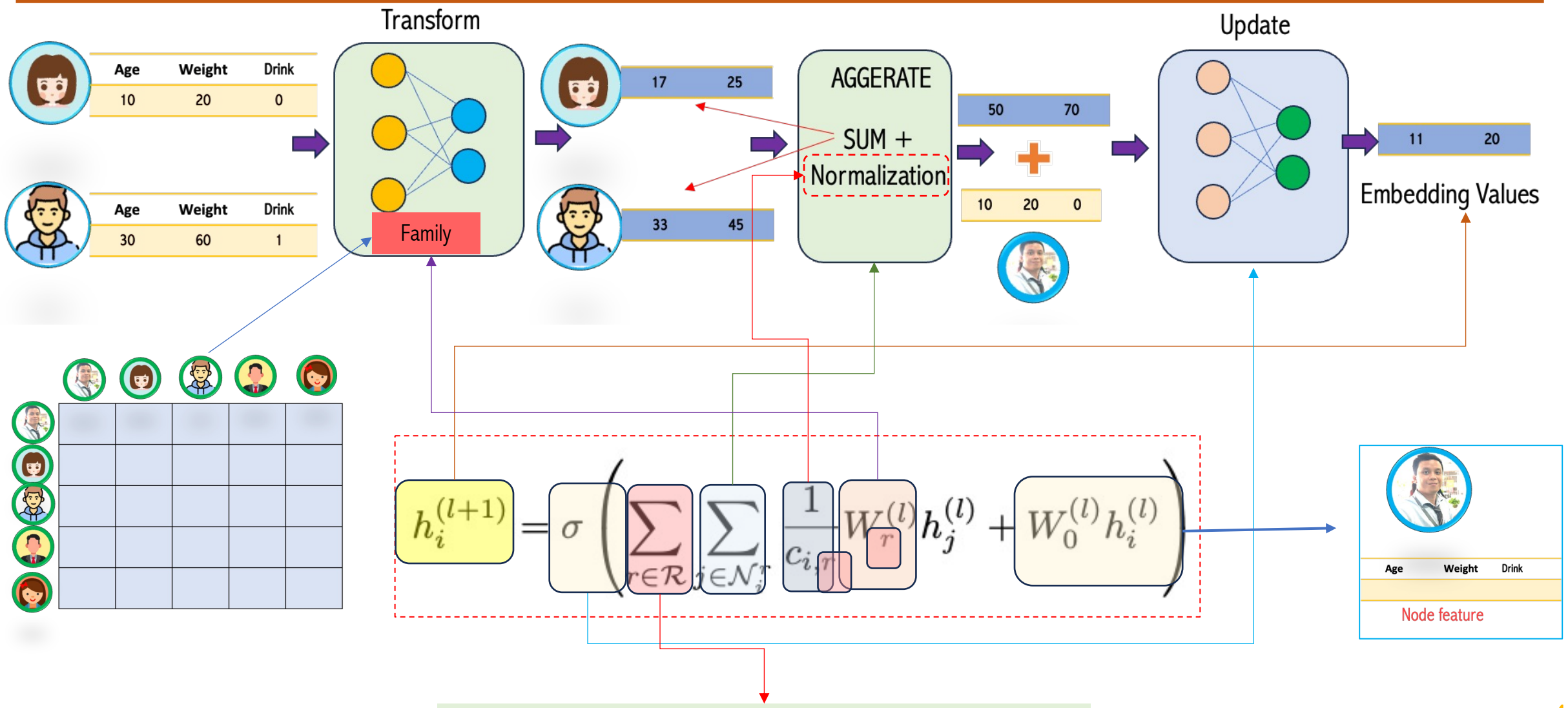
Various types of relationship: Edge conditions GNN

Relational GNN



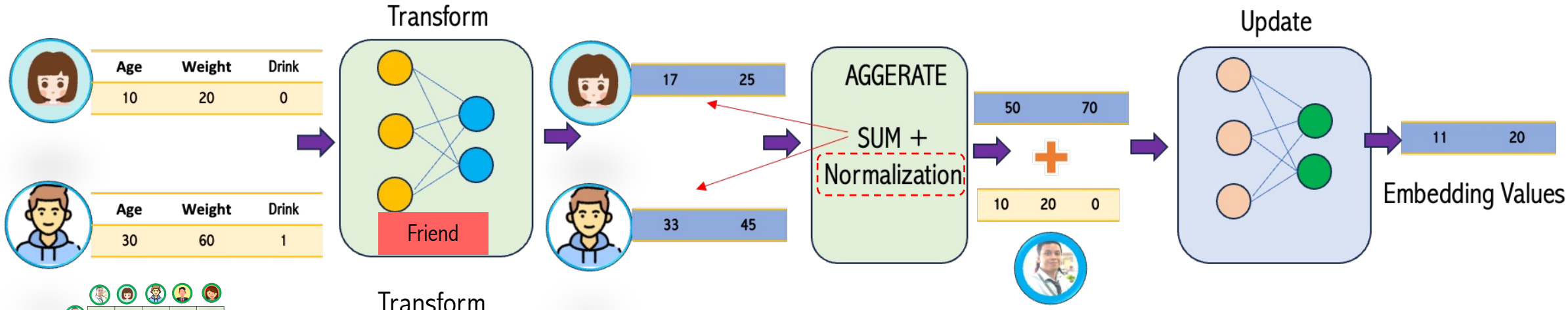
Various types of relationship: Edge conditions GNN

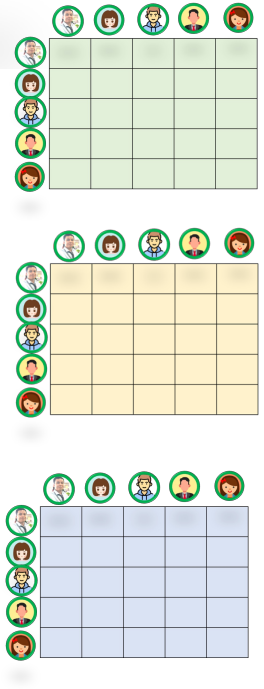
Relational GNN

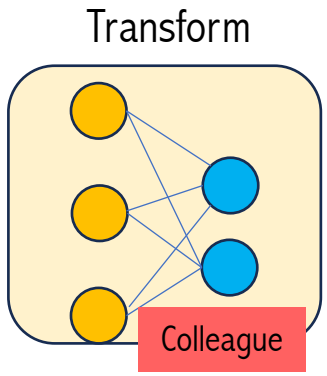


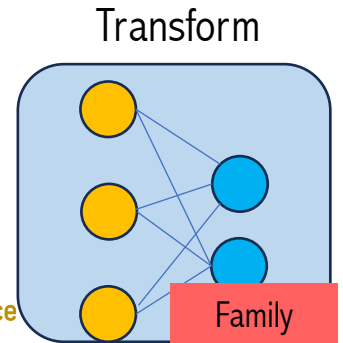
Various types of relationship: Edge conditions GNN

GNN: Feature-wise Linear Modulation









$$\boldsymbol{h}_v^{(t+1)} = l\left(\sum_{u \xrightarrow{\ell} v \in \mathcal{E}} \sigma\left(\boldsymbol{\gamma}_{\ell,v}^{(t)} \odot \boldsymbol{W}_\ell \boldsymbol{h}_u^{(t)} + \boldsymbol{\beta}_{\ell,v}^{(t)}\right) ; \boldsymbol{\theta}_l\right)$$

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

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

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 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

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

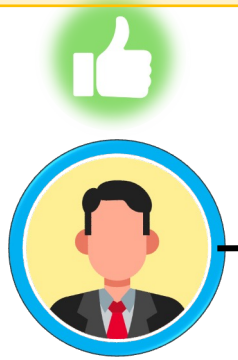
How to integrate multi-dimensional edge features to the transformation of the neighborhood states?

Love to watch movie

Not Love to watch movie

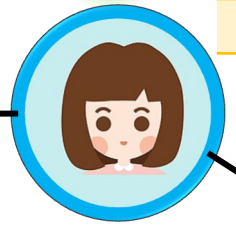
Node feature

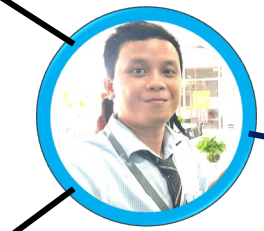
Age	Weight	Drink



Node feature

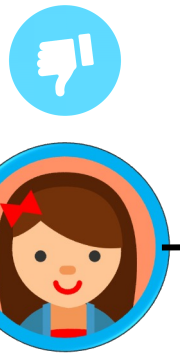
Age	Weight	Drink





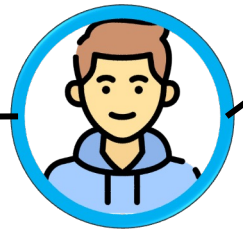
Node feature

Age	Weight	Drink



Node feature

Age	Weight	Drink



Node feature

Age	Weight	Drink

Edge information

Friend	Friend Since	Live together
Yes	9	No

How do edge features utilize in GNN?

Multidimensional Edge Features

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

UPDATE

AGGREGATE

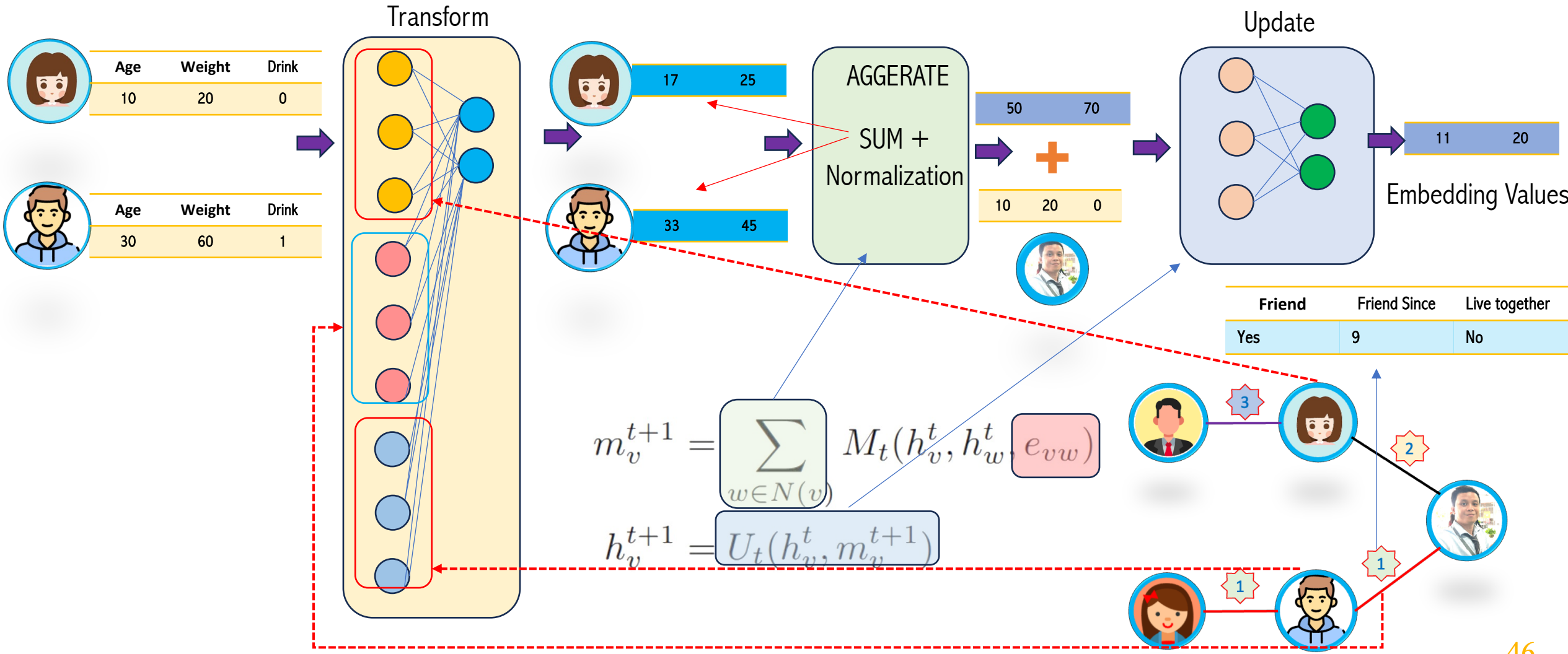
TRANSFORM

Message Passing in Neural Network

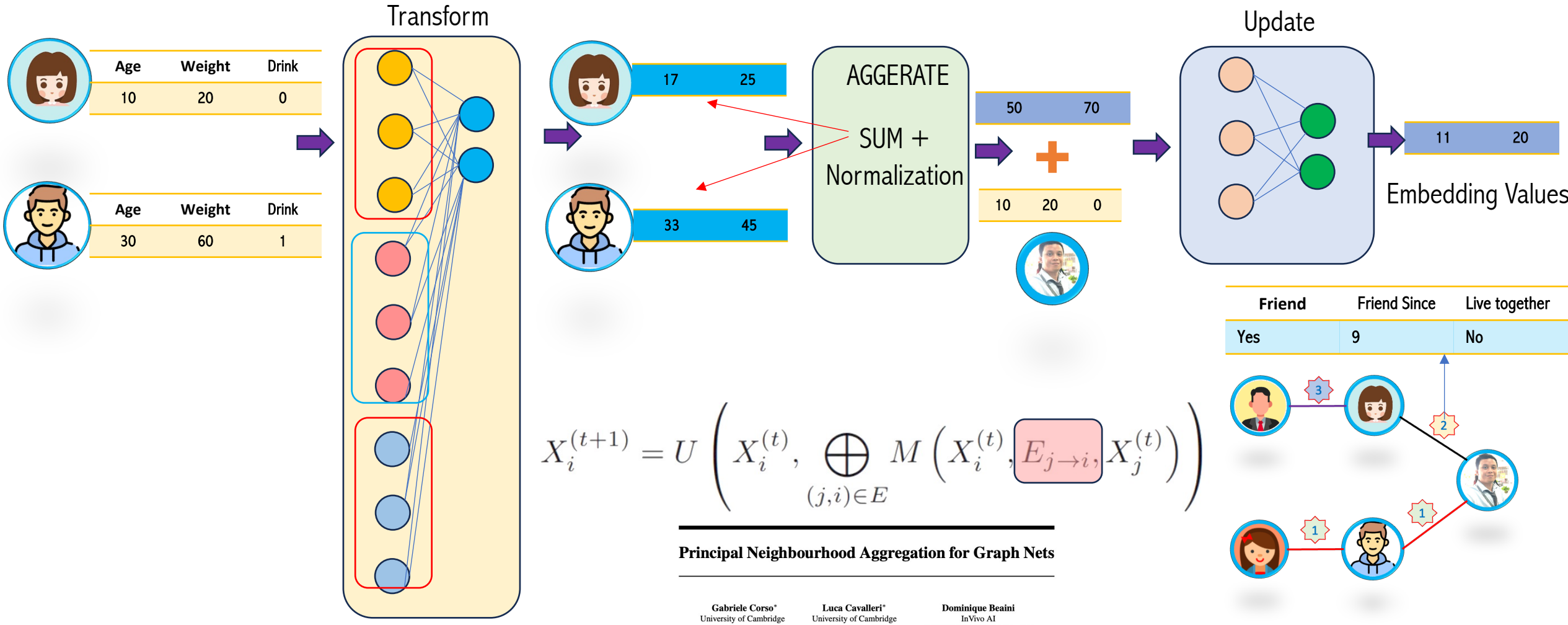
0	[edge feature vector]	[edge feature vector]	0	0
[edge feature vector]	0	0	[edge feature vector]	0
[edge feature vector]	0	0	0	[edge feature vector]
0	[edge feature vector]	0	0	0
0	0	[edge feature vector]	0	0

What is the size of this adjacency matrix?

Multidimensional Edge Features: MLP



Multidimensional Edge Features: PNAConv

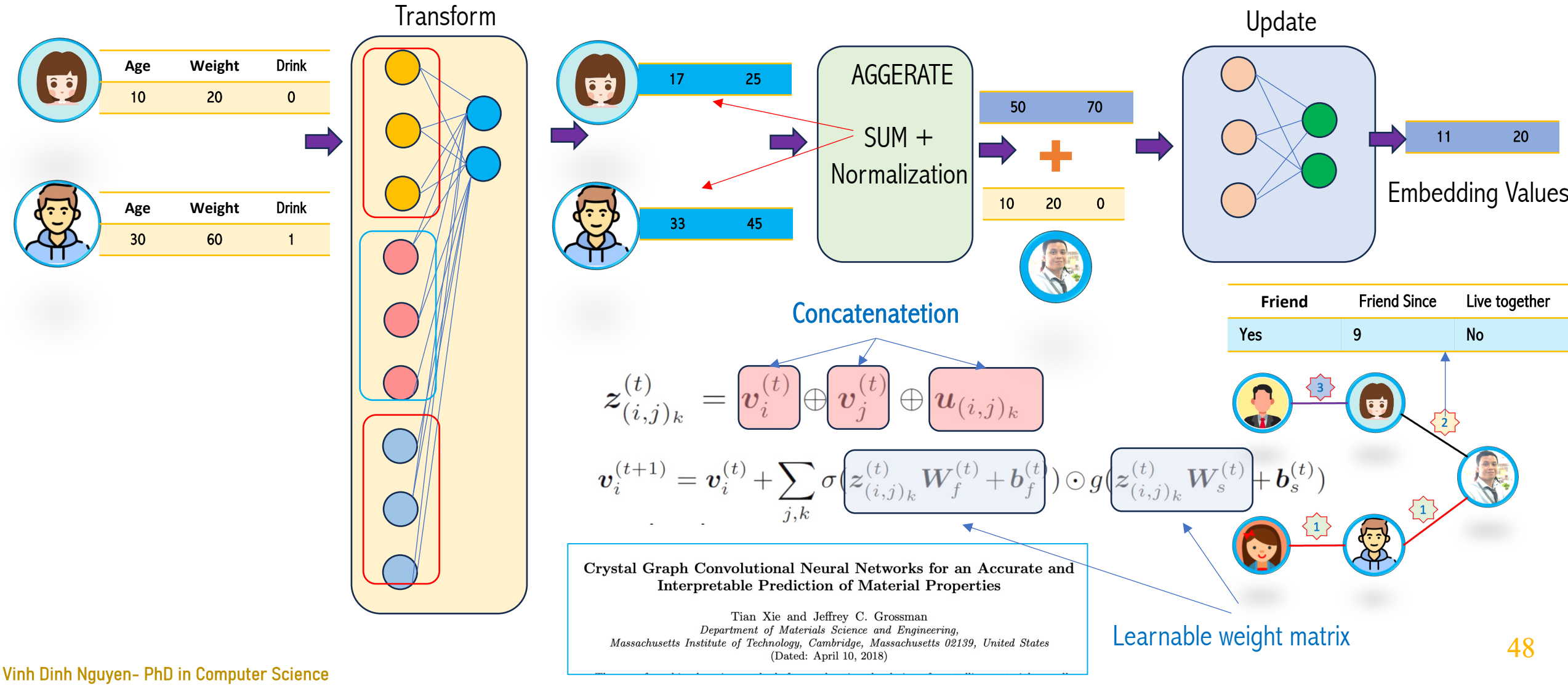


$$X_i^{(t+1)} = U\left(X_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(X_i^{(t)}, E_{j\to i}, X_j^{(t)}\right)\right)$$

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



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

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

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

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

Nodel-level Attention

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

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

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

Edge-level Attention

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

Edge Feature Embedding: NENN

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

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

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

Nodel-level Attention

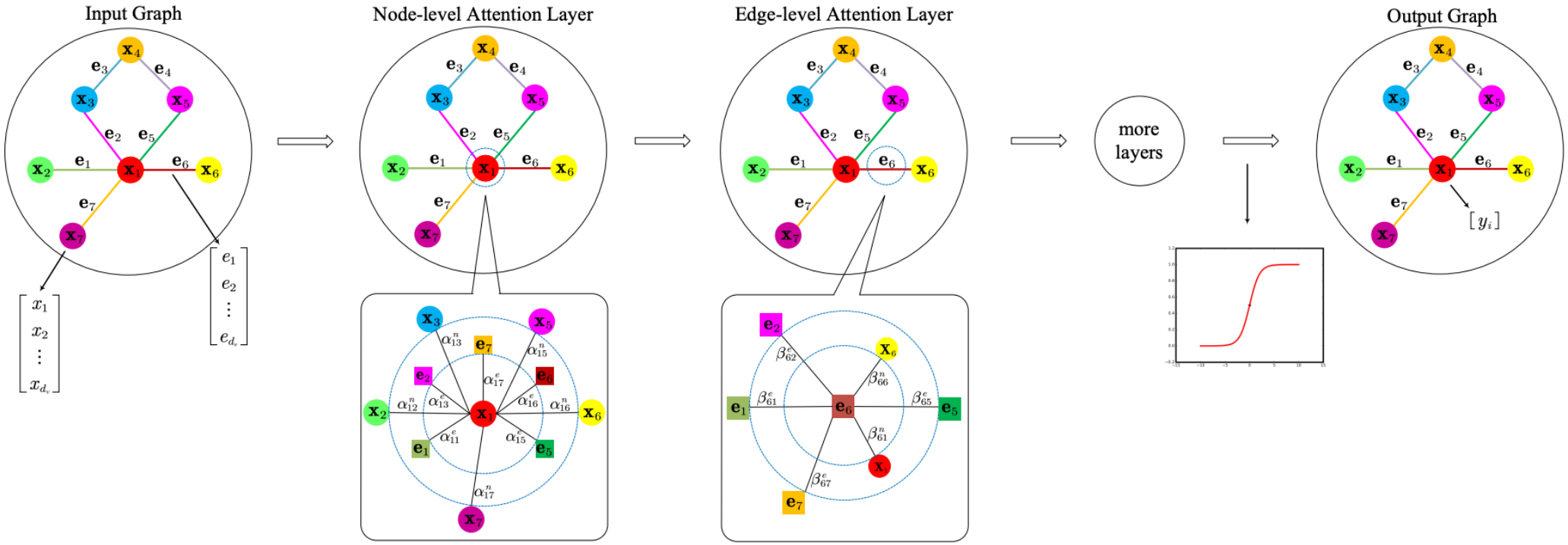
$$\mathbf{e}_{\mathcal{E}_i} = \sigma\Big(W_e \cdot \text{MEAN}(\{\beta_{ik}^e \mathbf{e}_k, \forall_k \in \mathcal{E}_i\})\Big)$$

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

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

Edge-level Attention

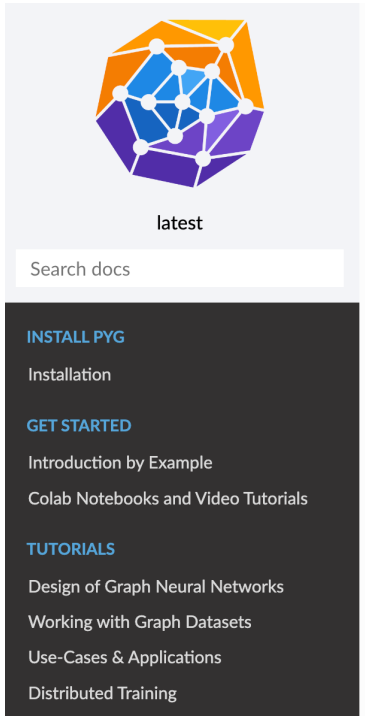
This approach iteratively updates node and edge embeddings in order to merge both information together



Edge Feature: How to Use

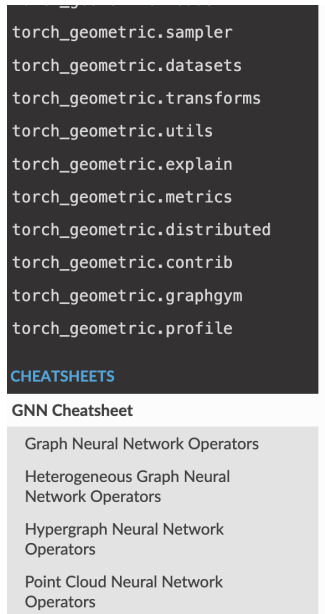
PyTorch geometric

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



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`



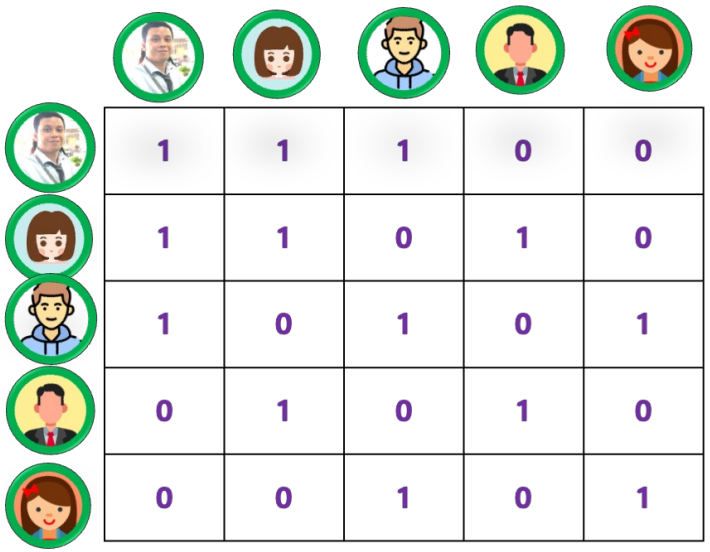
Heterogeneous Graph Neural Network Operators

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

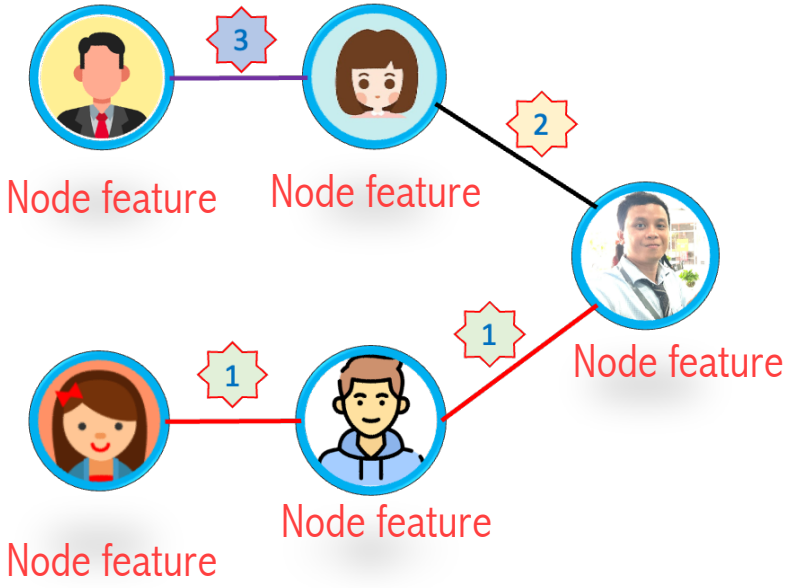
- Edge Feature in GNN
- Edge Weight in GNN
- Relational GNN
- Multidimension Edge Feature
- Attention in GNN
- Example: Graph-Level Prediction
- Summary

Attention in Graph Neural Network

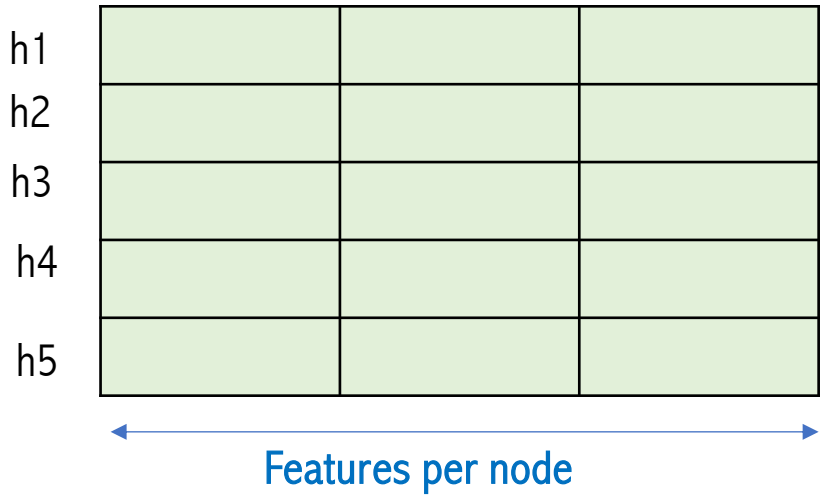


1	1	1	0	0
1	1	0	1	0
1	0	1	0	1
0	1	0	1	0
0	0	1	0	1

Adjacency Matrix

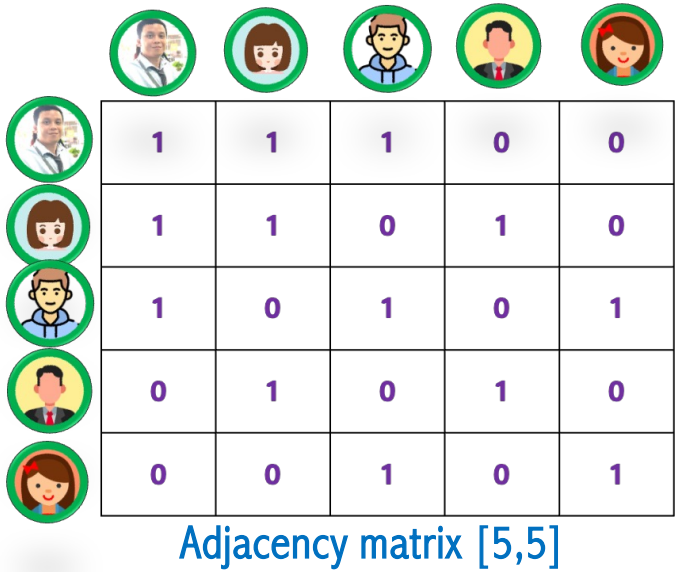


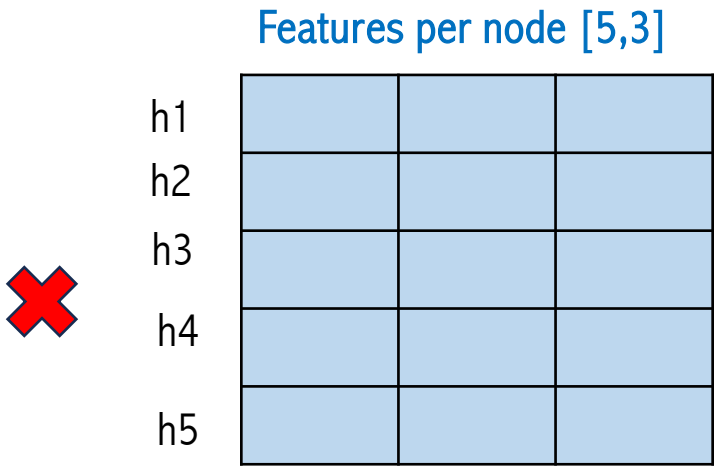
Age	Weight	Drink

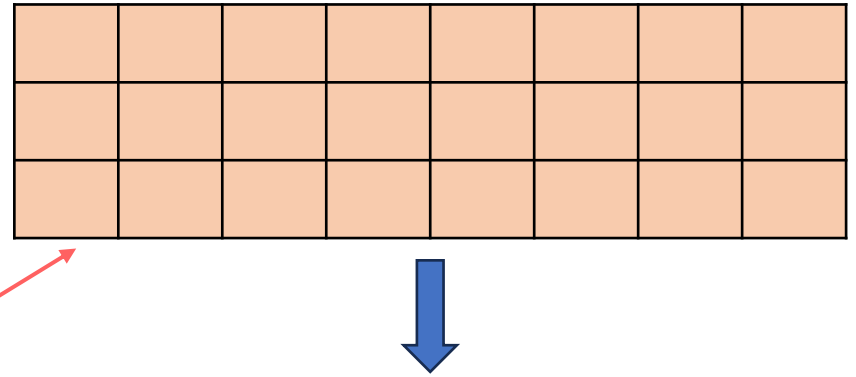


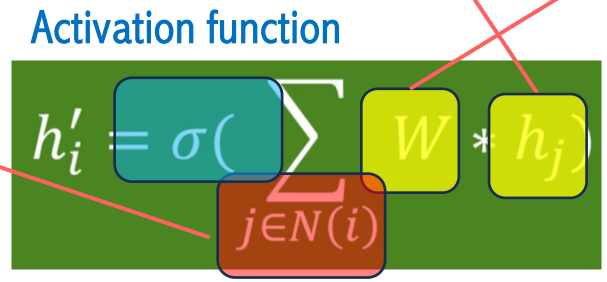
How to use this feature matrix in GNN

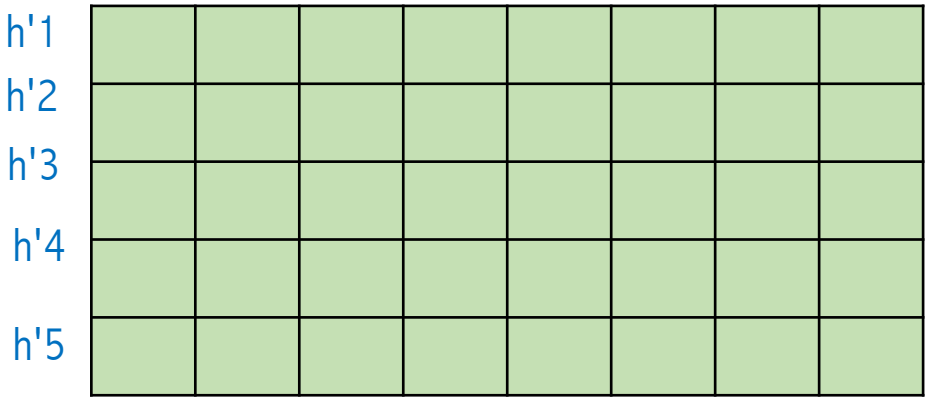
GNN: Look at the 1st Layer





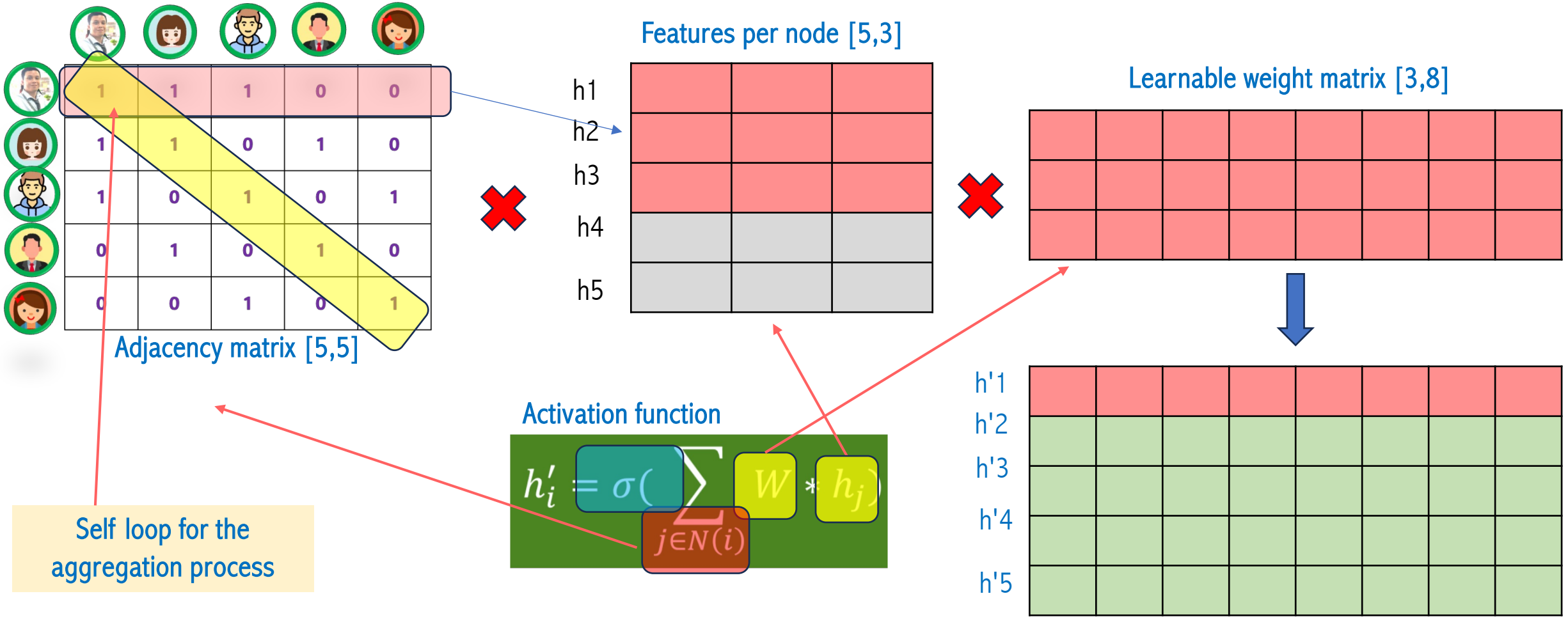






Embedding Features per node[1,8]: information of their own node feature and neigbor node features

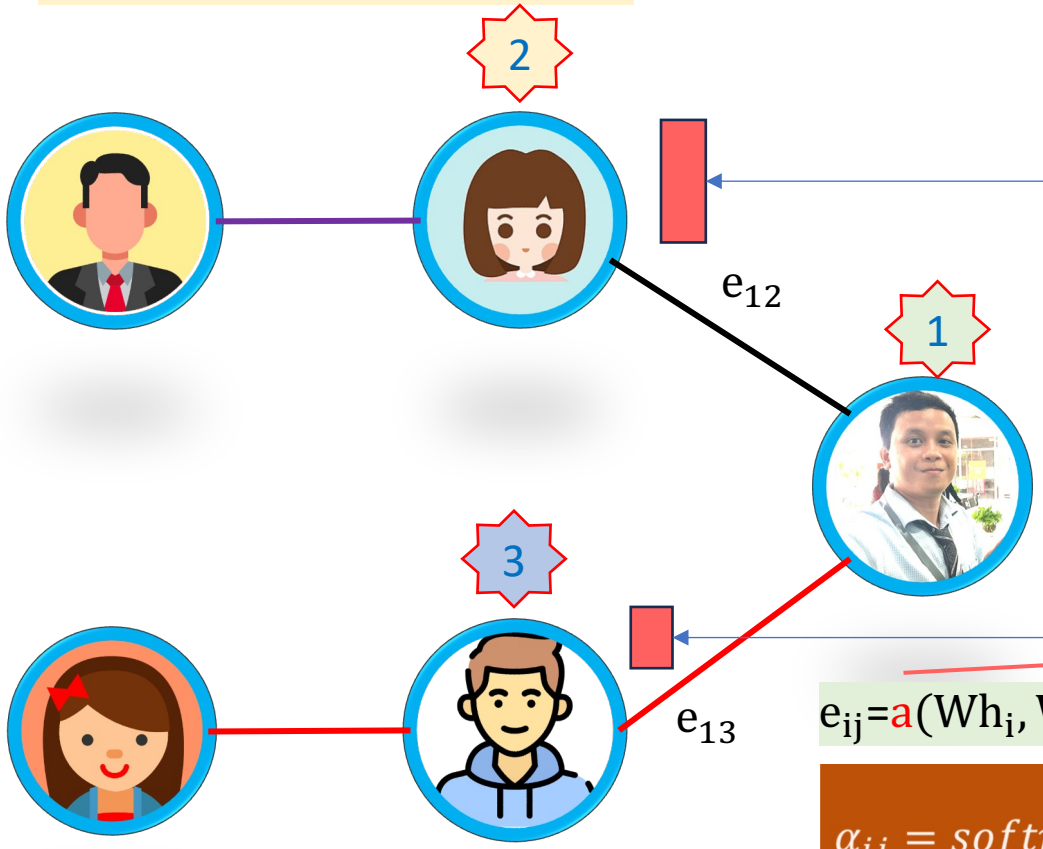
GNN: Look at the 1st Layer



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

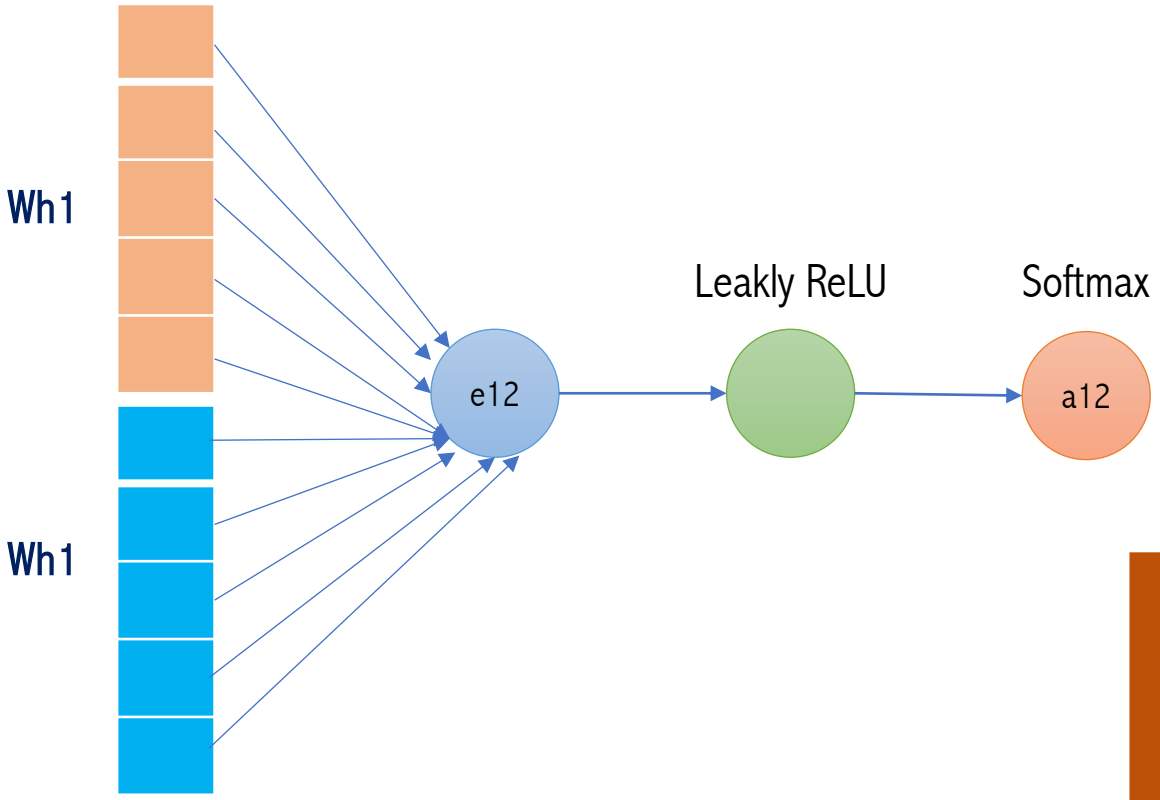
Attention mechanism

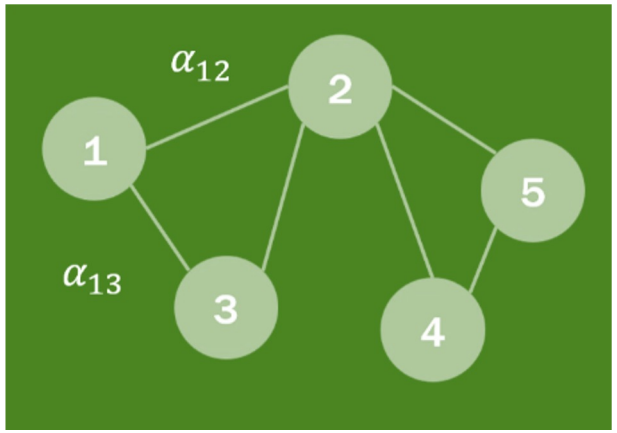
$e_{ij}=a(Wh_i, Wh_j)$

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





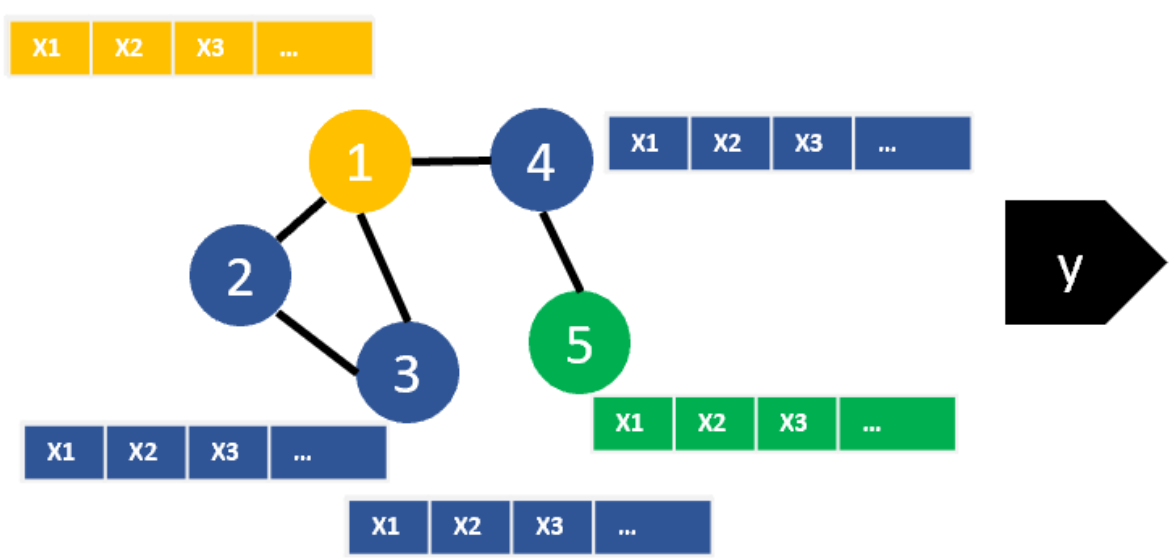
$$\alpha_{ij} = \frac{\exp(LeakyReLU\left(\overrightarrow{w_a^T}\,[Wh_i||Wh_j]\right))}{\sum_{k\epsilon N(i)} \exp(LeakyReLU(\overrightarrow{w_a^T}\,[Wh_i||Wh_j]))}$$


QUIZ TIME

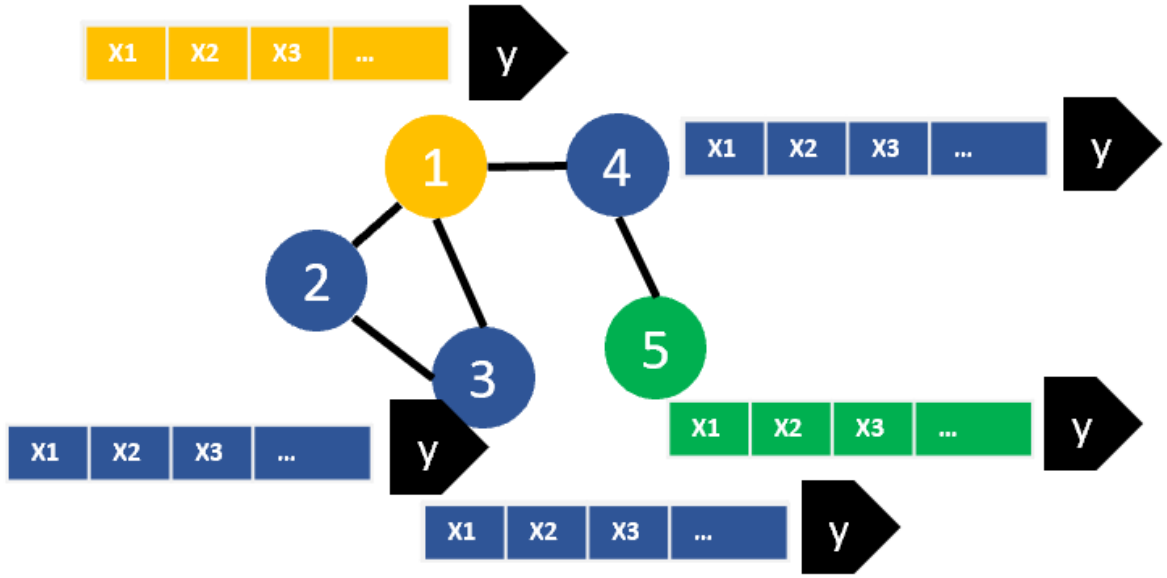
Outline

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

GNN: Graph Prediction



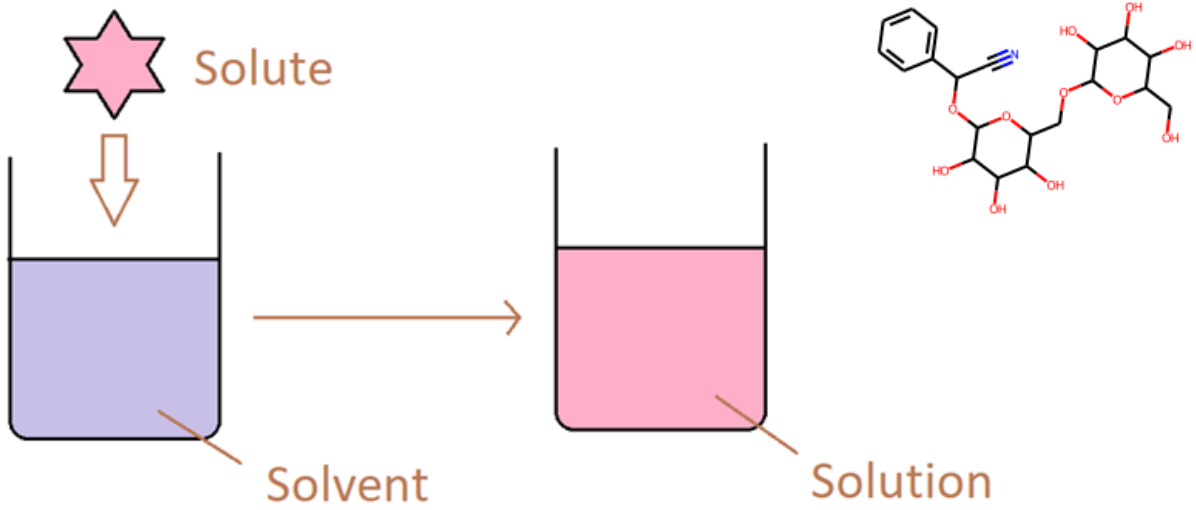
Graph-level prediction

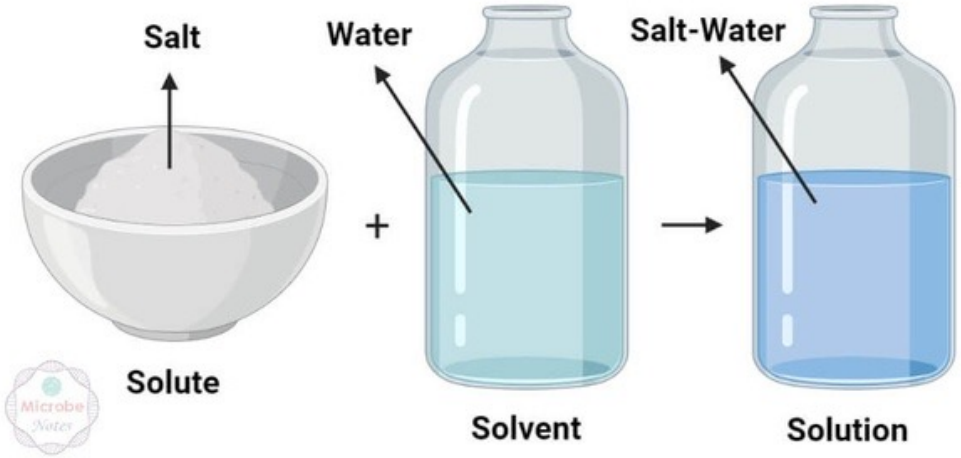


Node-level prediction

GNN: Graph Prediction

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





GNN: Graph Prediction

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

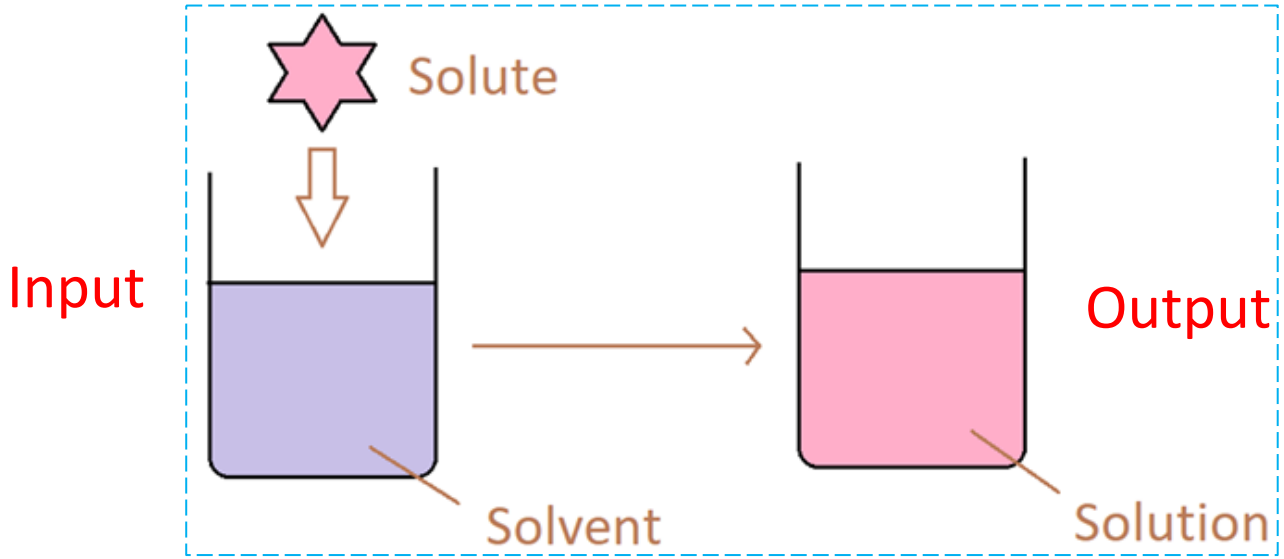
```
!pip install torch-scatter     -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-sparse      -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-cluster     -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-spline-conv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

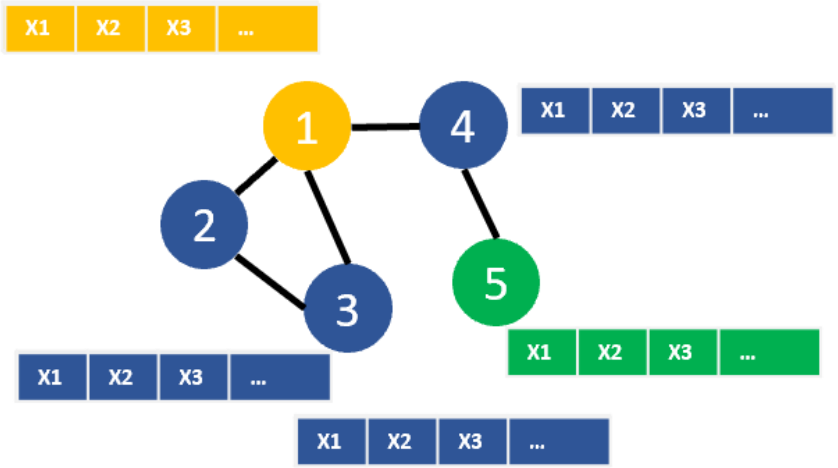
Implementing the Graph Neural Network

```
# 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


```
# 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]])
```

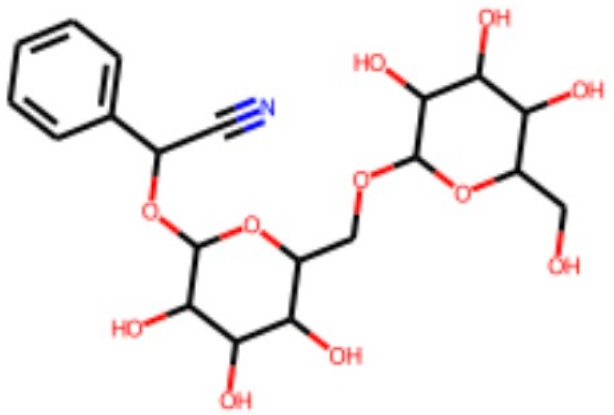
GNN: Graph Prediction

- 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

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

```python
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)
```

AI VIETNAM
All-in-One Course

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
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
```

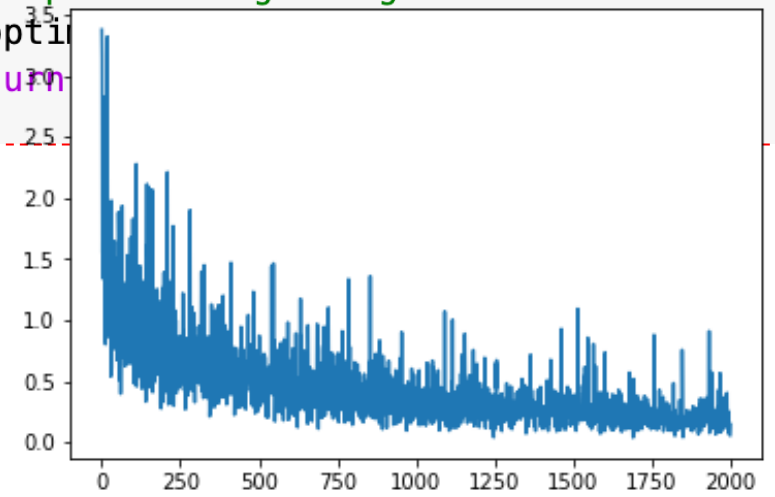
```
GCN(
  (initial_conv): GCNConv(9, 64)
  (conv1): GCNConv(64, 64)
  (conv2): GCNConv(64, 64)
  (conv3): GCNConv(64, 64)
  (out): Linear(in_features=128, out_features=1, bias=True)
)
Number of parameters:  13249
```

AI VIETNAM
All-in-One Course

GNN: Graph Prediction

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

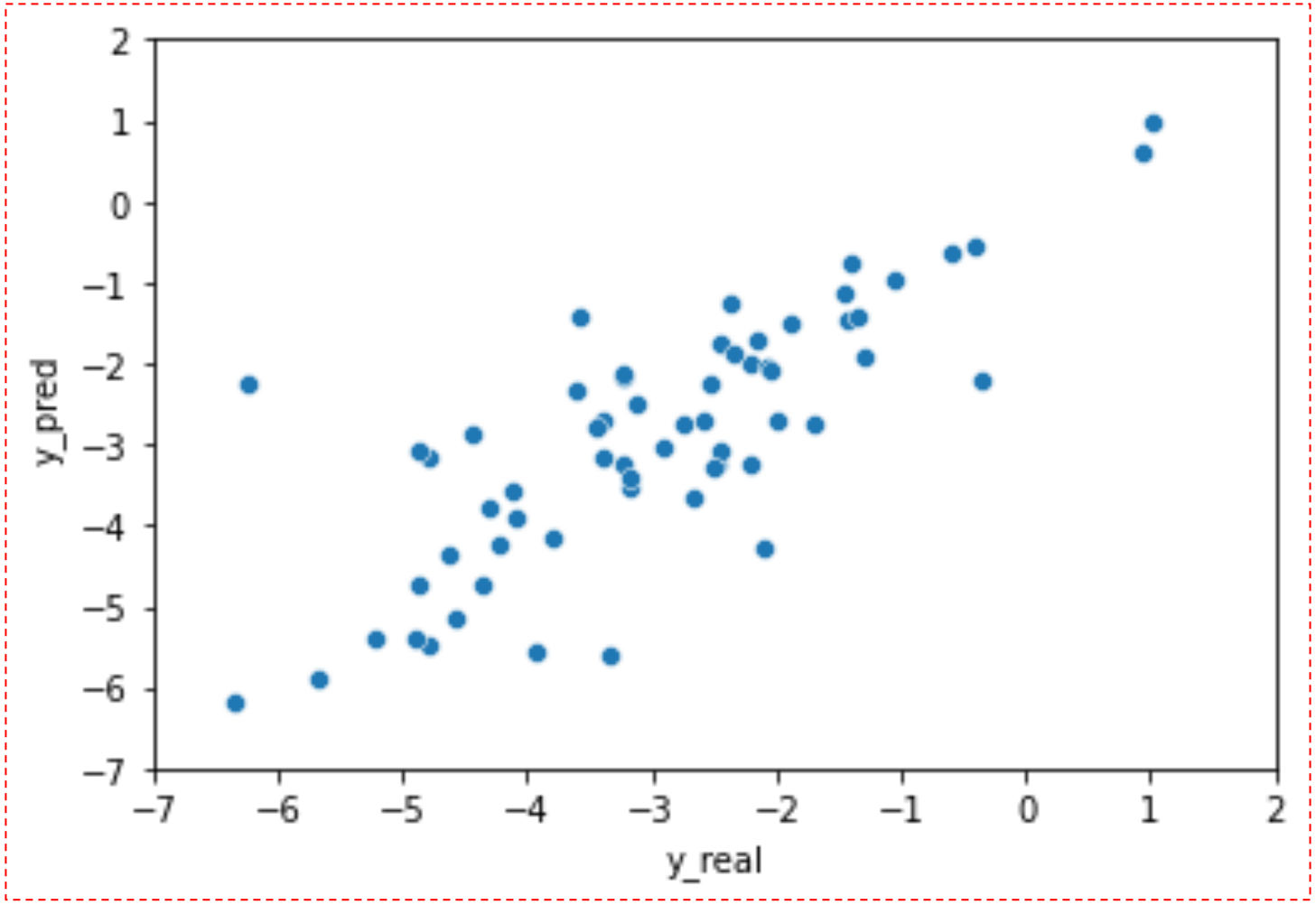
```
def train(data):
    # Enumerate over the data
    for batch in loader:
      # Use GPU
      batch.to(device)
      # Reset gradients
      optimizer.zero_grad()
      # Passing the node features and the connection info
      pred, embedding = model(batch.x.float(), batch
      # Calculating the 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 0.1817774772644043
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.

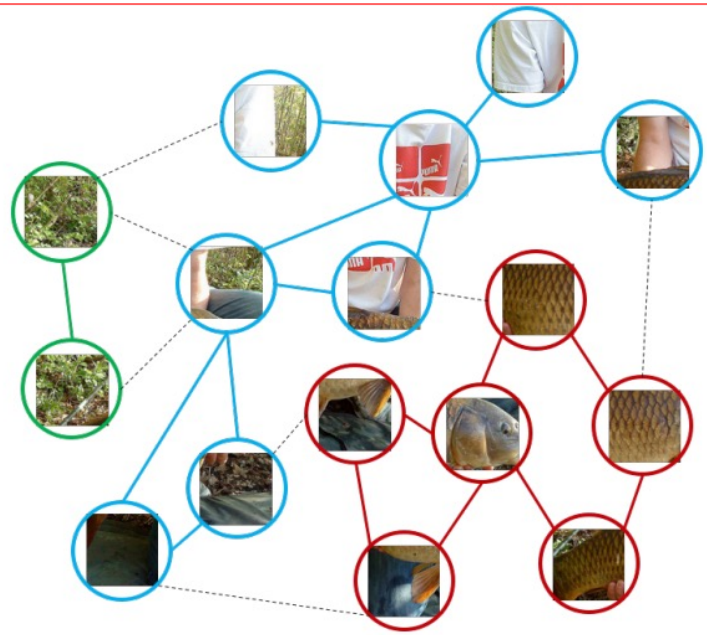
The widely-used convolutional neural network and transformer treat the image as a grid or sequence structure, which is not flexible to capture irregular and complex objects

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.

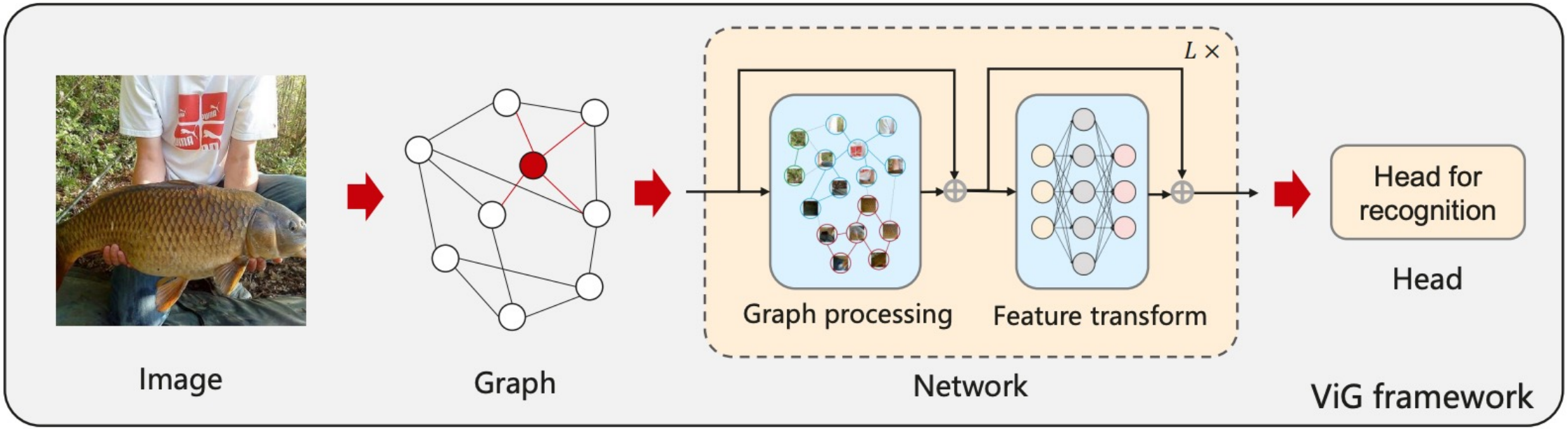
(b) Sequence structure.



(b) Graph structure.

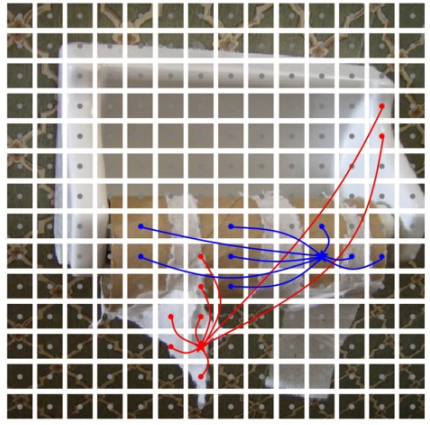
Illustration of the grid, sequence and graph representation of the image

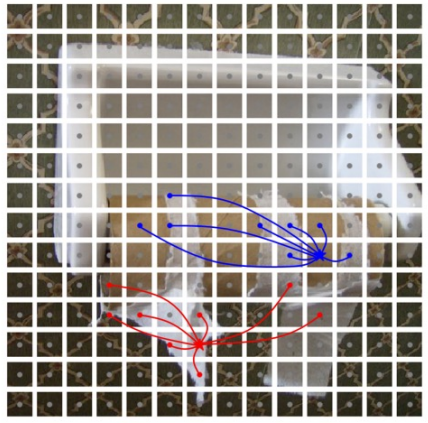
Vision GNN: Read and Understand



The framework of the proposed ViG model

Vision GNN: Read and Understand



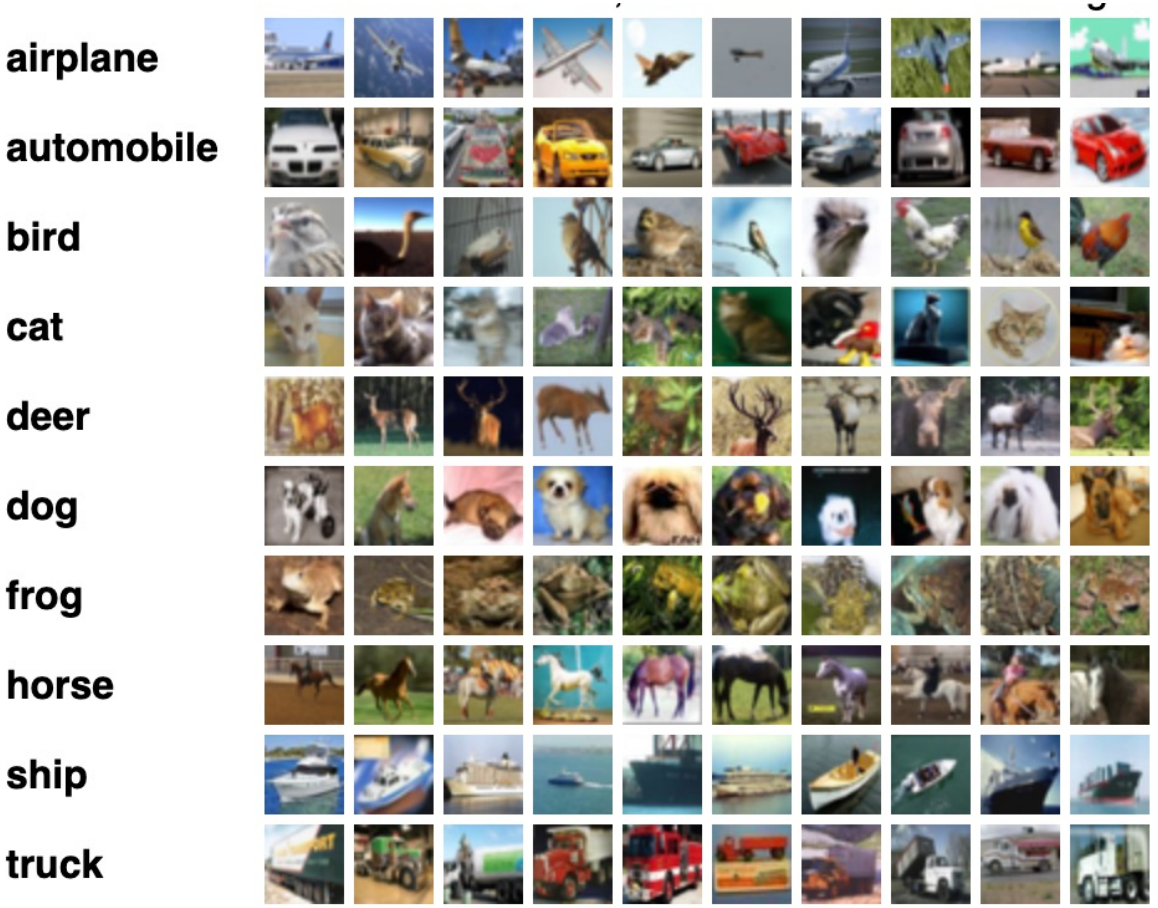


(a) Input image. (b) Graph connection in the 1st block. (c) Graph connection in the 12th block.

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.



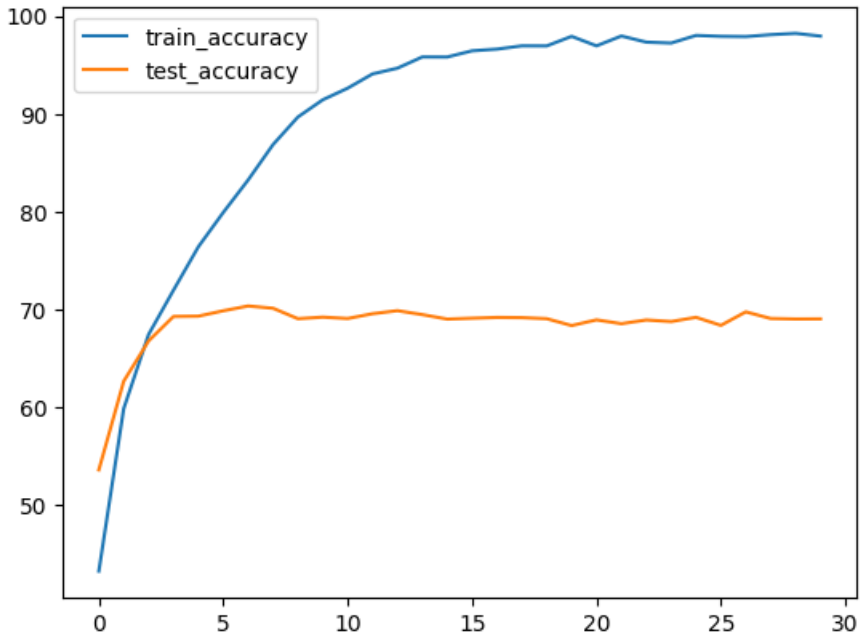
Example: CIFAR-10 Dataset

	#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)
```

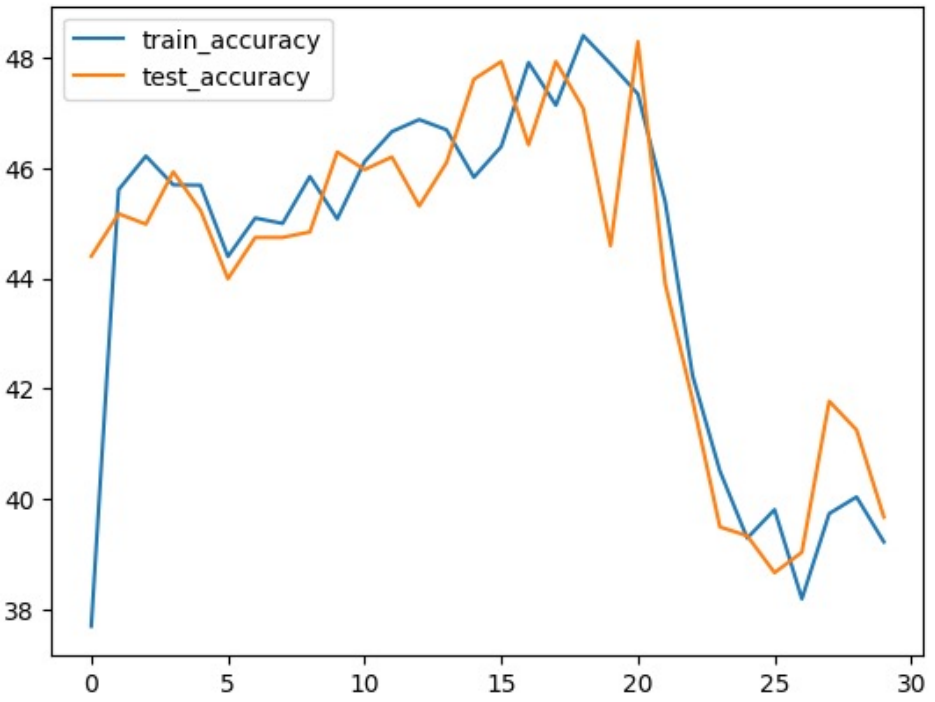
CNN for Image Classification



Example: CIFAR-10 Dataset

```python
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",
```

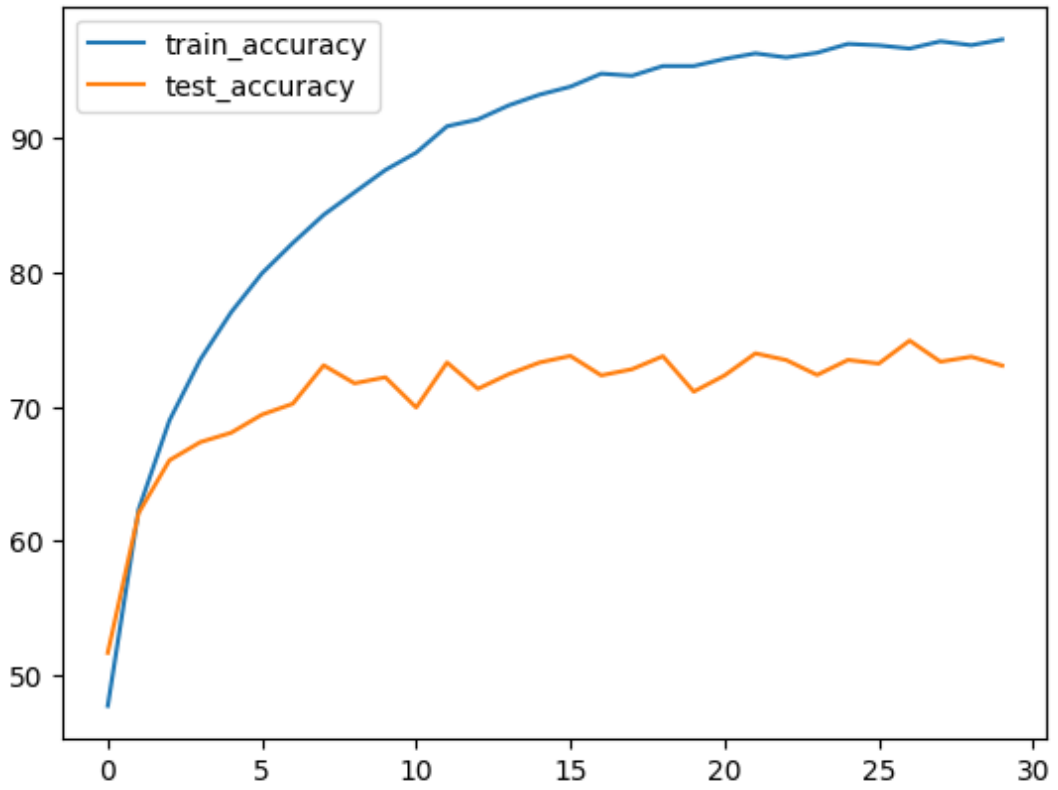
Vision Transformer for Image Classification



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()
```

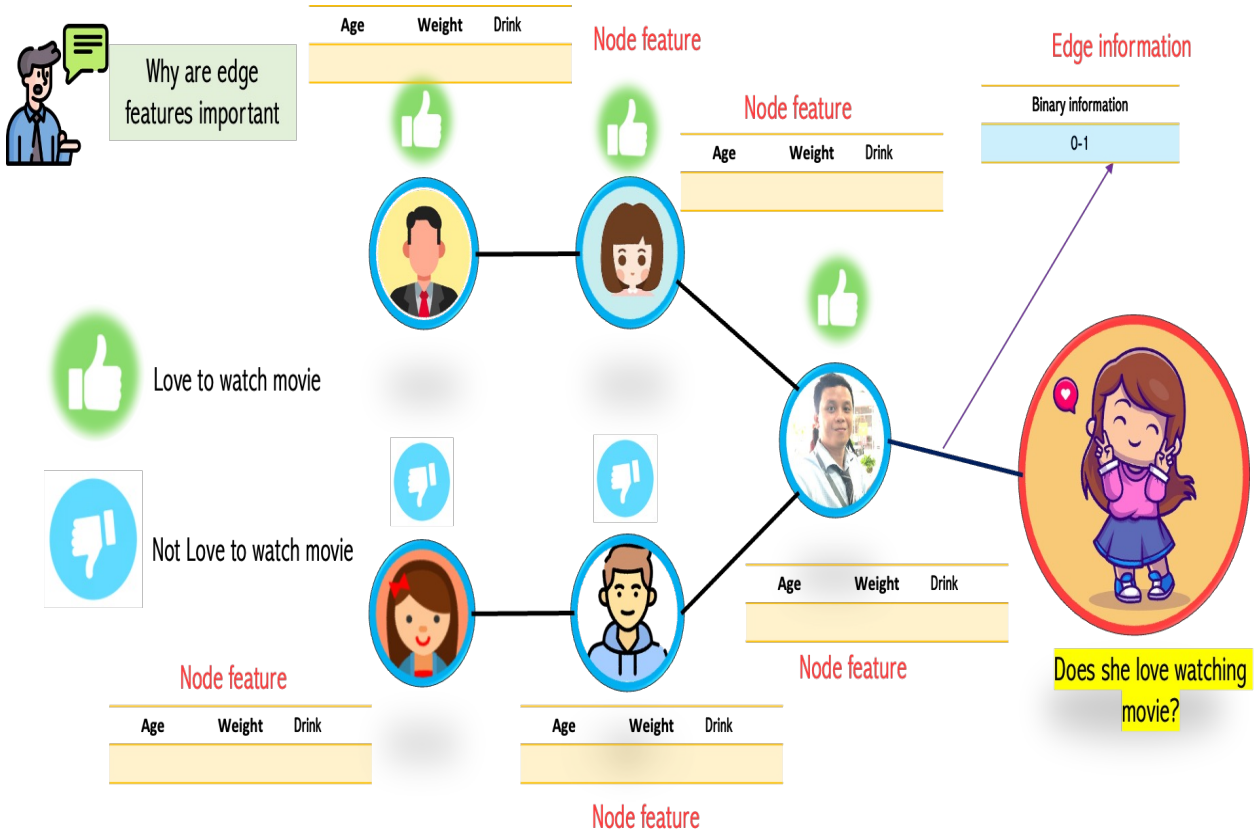
Vision GNN for Image Classification



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