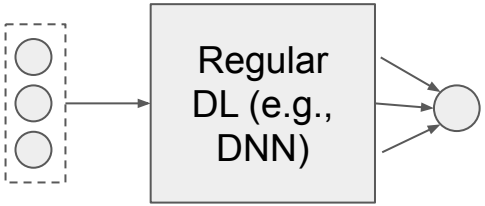
Graph Neural Network

Introduction

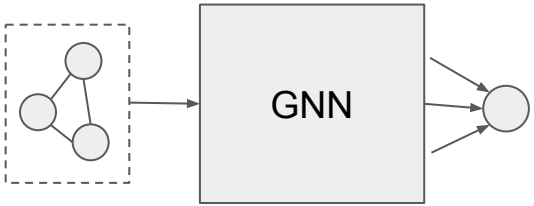
Introduction What is GNN is used for

Input elements are usually independent to each other



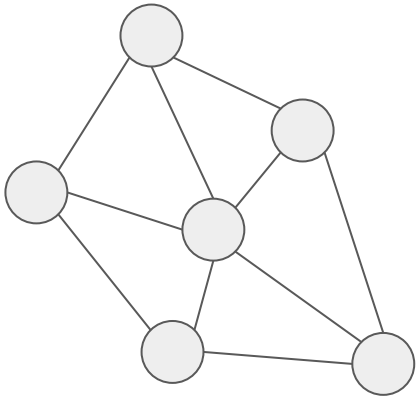
For example, we know people's age and occupation, and would like to predict the income. Age and occupation are independent to each other

Input elements have certain connections



For example, we know a person have 3 friends, and would like to predict if another people can be his friend. The 3 friends (elements) here are connected to each other

Introduction What is graph



A graph contains **two** elements

Node

- *Each node contains some information*

Edge

- *Each edge contains some information*
- *Each edge can be directive or not*

Introduction Image as a graph

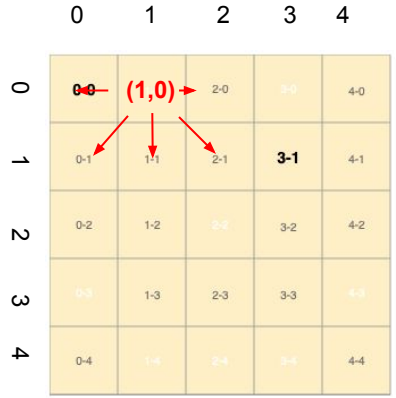
How to represent an image ?

Option 1: thinking of images as rectangular grids with image channels, representing them as arrays *(e.g., 5x5x3)*

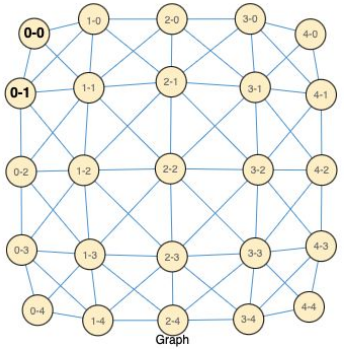
Option 2: thinking of images is as graphs with regular structure, where each pixel represents a node and is connected via an edge to adjacent pixels.

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.

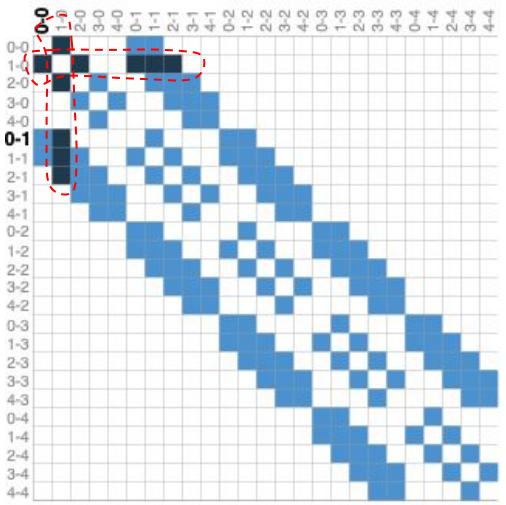
Assuming we have an image with 5x5 pixels:



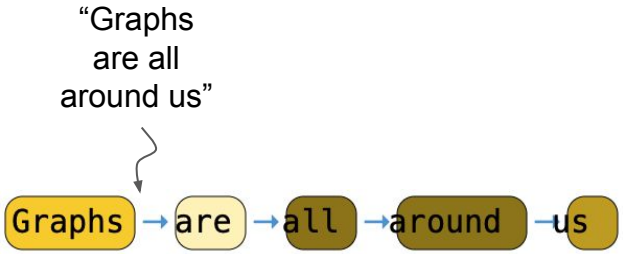
The image can be represented by the graph



We can use "adjacency matrix" to represent the adjacent nodes, e.g., the left shows the nearby nodes for the node (1,0)

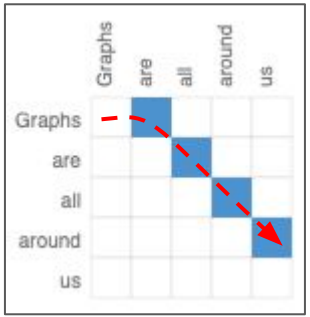


Introduction Text as a graph



We can digitize text by associating “node” to each character, word, or token, and representing text as a sequence of these indices.

This creates a simple directed graph, where each word is a node and is connected via an edge to the node that follows it.

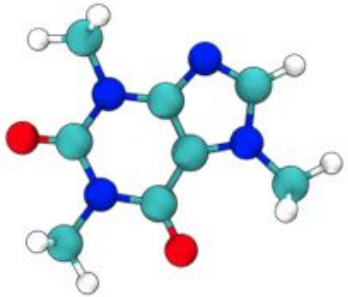


The *adjacency matrix* *then can be constructed like the above*

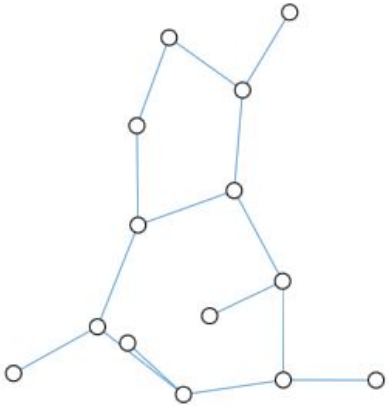
Introduction Molecules as a graph

Molecules are the building blocks of matter, and are built of atoms and electrons in 3D space.

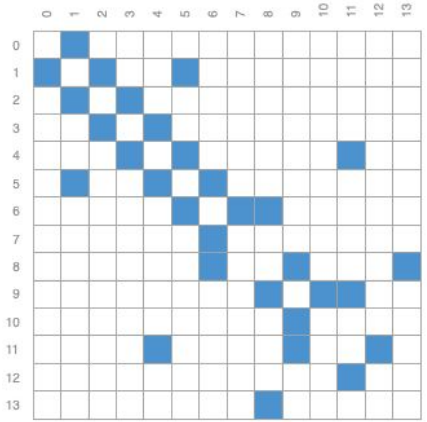
All particles are interacting, it's a very convenient to describe this 3D object as a graph, where nodes are atoms and edges represent their connections



Molecules can be represented by a graph



The *adjacency matrix* *then can be constructed like the above*

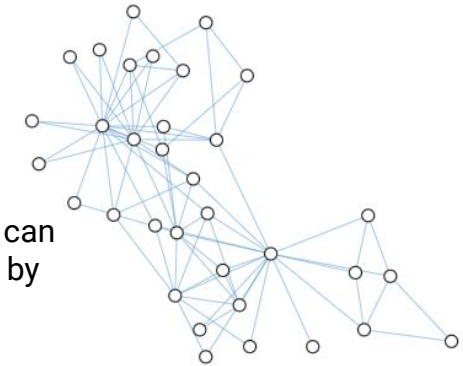


Introduction Social network as a graph

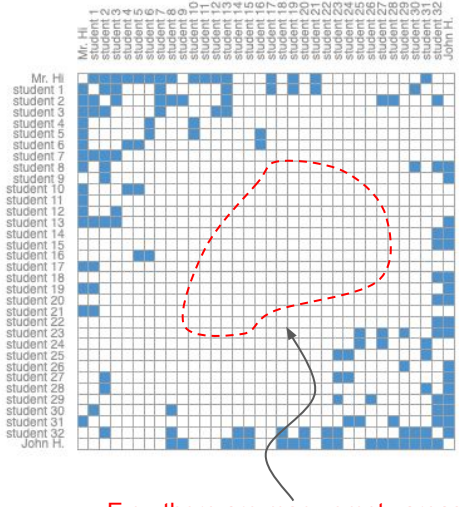
Social networks are tools to study patterns in collective behaviour of people, institutions and organizations. We can build a graph representing groups of people by modelling individuals as nodes, and their relationships as edges.

Unlike image and text data, social networks do not have identical adjacency matrices (e.g., it's not a systemic matics, as individual does not identical contacts compared to each other)

Social network can be represented by a graph



The *adjacency matrix then can be constructed like the above*



E.g., there are many empty areas in the middle of matrix, this means that there are students who don't have any interactions

Tasks around GNN

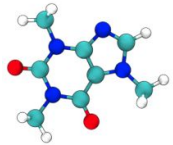
There are three general types of prediction tasks on graphs:

- Graph-level: we predict a single property for a whole graph
- Node-level: we predict some property for each node in a graph
- Edge-level: we predict the property or presence of edges in a graph

There are three general types of prediction tasks on graphs:

- Graph-level: we predict a single property for a whole graph
- Node-level: we predict some property for each node in a graph
- Edge-level: we predict the property or presence of edges in a graph

E.g., for a molecule represented as a graph, we might want to predict what the molecule smells like ?



What it smells ?

There are three general types of prediction tasks on graphs:

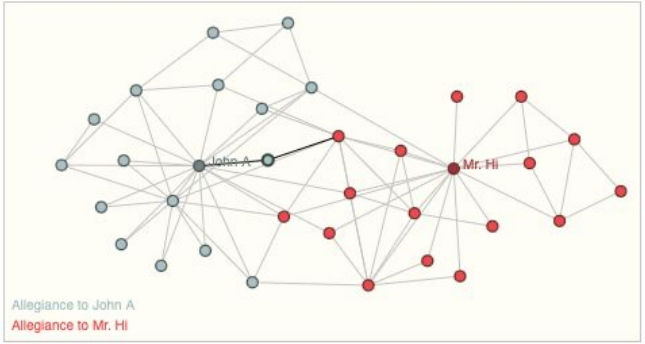
- Graph-level: we predict a single property for a whole graph
- Node-level: we predict some property for each node in a graph
- Edge-level: we predict the property or presence of edges in a graph

Zach's karate club is a social network, where:

- the nodes represent individual karate practitioners, and
- the edges represent interactions between these members outside of karate (For example, if two karate practitioners are friends on Facebook, there would be an edge between them in the social network).

After some disagreements, the karate club split into two factions, one loyal to Mr. Hi and the other loyal to John H. The node-level prediction problem is to classify whether a given karate practitioner will become loyal to Mr. Hi or John H.

One way to solve this problem is to use the distance between a node and either Mr. Hi or John H. The closer a node is to a particular person, the more likely it is to be loyal to that person (e.g., if a karate practitioner is friends with Mr. Hi on Facebook, they are more likely to be loyal to Mr. Hi than if they are not friends with Mr. Hi on Facebook).



There are three general types of prediction tasks on graphs:

- Graph-level: we predict a single property for a whole graph
- Node-level: we predict some property for each node in a graph
- Edge-level: we predict the property or presence of edges in a graph

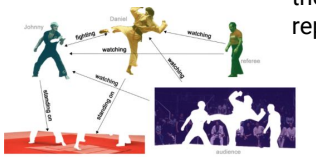
One example of edge-level inference is in image scene understanding.

Deep learning models can be used to predict the relationship between different objects in an image.

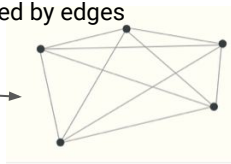
We can phrase this as an edge-level classification: given nodes that represent the objects in the image, we wish to predict which of these nodes share an edge or what the value of that edge is.

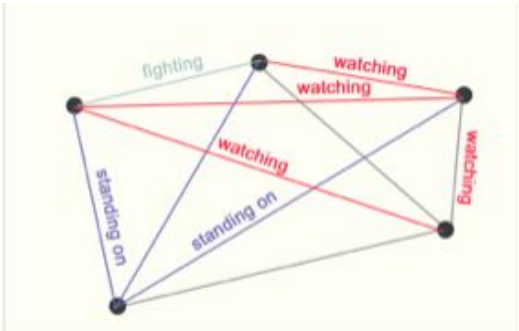
Original image

The original image can be segmented into 5 objects:
- Fighter 1
- Fighter 2
- Referee
- Audience
- Mat



Each object can be represented by a node, and their interactions are represented by edges





Node and edge can be labelled accordingly

Challenges of using graph in machine learning

How to represent graphs to be compatible with neural networks ?

We need to think about what information we want to use to make predictions. In general, there are four types of information that we might want to use:

- **Nodes:** The nodes represent the individual entities in the graph. For example, in a social network, the nodes might represent people.
- **Edges:** The edges represent the relationships between the nodes. For example, in a social network, the edges might represent friendships.
- **Global context:** The global context is information about the entire graph, such as the number of nodes and edges.
- **Connectivity:** The connectivity is information about how the nodes are connected to each other. For example, in a social network, the connectivity might represent the number of friends that each person has.

Easy to represent

Difficult to represent

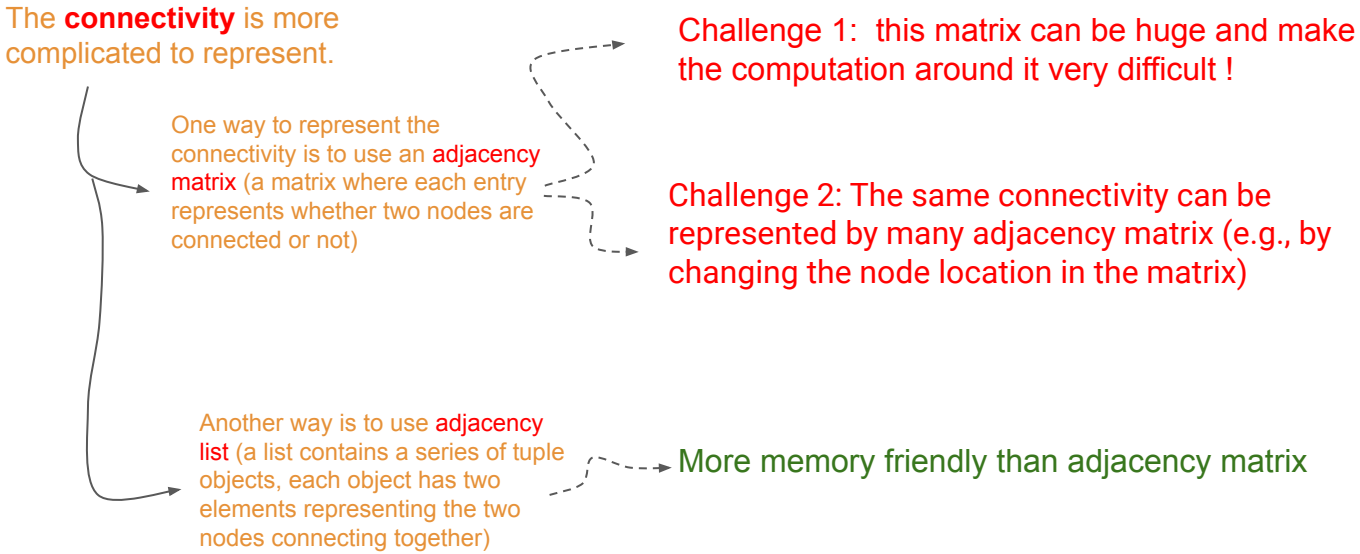
The nodes can be represented as a matrix, where each row represents a node and each column represents a feature of the node

The edges can be represented as a matrix, where each row represents an edge and each column represents the nodes that are connected by the edge.

The connectivity is more complicated to represent. One way to represent the connectivity is to use an adjacency matrix. An adjacency matrix is a matrix where each entry represents whether two nodes are connected or not. However this matrix can be huge and make the computation around it very difficult !

How to represent graphs to be compatible with neural networks ?

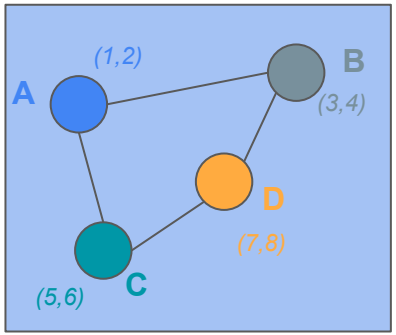
Connectivity: The connectivity is information about how the nodes are connected to each other. For example, in a social network, the connectivity might represent the number of friends that each person has.



How GNN works

Step 1: Message Passing

Assuming the question is: we want to know if a new person, with certain features, can potentially be Andy's best friend



Assuming that we have 4 person (nodes): A, B, C and D, in Andy's social circle

Each node has its own features, e.g., A has a feature vector of (1,2)

Step 1: each node sends its own feature vector to its neighbors

For example, node A sends its feature vector (1,2) to node B and node C. Similarly, node B sends its feature vectors (3,4) to nodes A and D

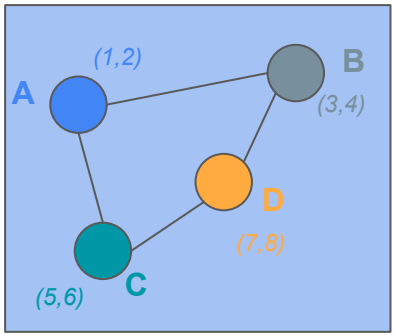
Step 2: each node aggregates the feature vectors of its neighbors (e.g., we can use a summation operation here in this example)

For example, node A receives the feature vector (3,4) (from node B) and vector (5,6) (node C), and sum them up to get (8, 10).

Step 3: each node updates its own feature vector by concatenating its original feature vector with the aggregated feature vectors from neighbours.

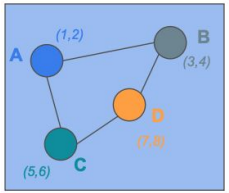
For example, node A gets its updated feature as (1, 2, 8, 10)

Step 1: Message Passing

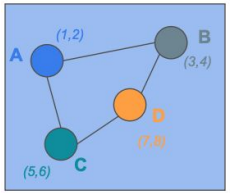


All of the nodes in a graph: A, B, C and D are finishing their message passings, e.g.,

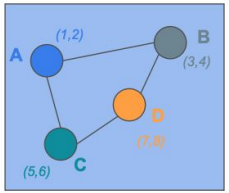
- A receives message from B and C (as last example)
- D receives message from B and C
- B receives message from A and D
- C receives message from A and D



Layer 1



Layer 2



Layer 3

.....

For example, A can be represented by:

*(1, 2, 8, 10, *, *, … *, *)*

- Before message passing
- After 1st message passing
- After 2nd message passing
- After N message passing

Input

Final understanding of a node

With the above message passing iteratively happen many times (e.g., each layer is one full graph passage passing)

Step 2: Embedding

After the message passing, in order to make the explanation easier, assuming that we can represent each node with a two dimensional tuple:

- A = (4.0, 1.0)
- D = (2.5, 1.0)
- B = (5.0, 1.0)
- C = (4.0, 0.0)

- The first column can be age,
- The second column can be a flag for the person's gender

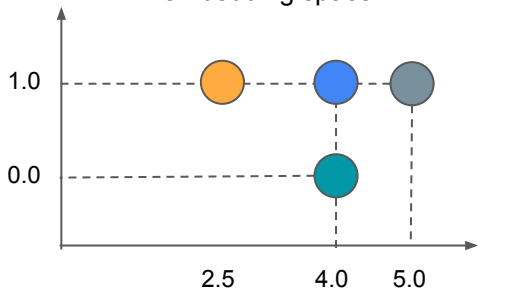
Step 2: Embedding

After the message passing, in order to make the explanation easier, assuming that we can represent each node with a two dimensional tuple:

- A = (4.0, 1.0)
- D = (2.5, 1.0)
- B = (5.0, 1.0)
- C = (4.0, 0.0)

- The first column can be age,
- The second column can be a flag for the person's gender

The final representation of nodes can be viewed in the following coordinate ~ this is called "node embedding", and the coordinate is called "embedding space"



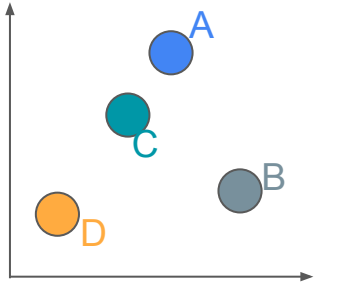
Step 2: Embedding

After the message passing, in order to make the explanation easier, assuming that we can represent each node with a two dimensional tuple:

- A = (4.0, 1.0)
- D = (2.5, 1.0)
- B = (5.0, 1.0)
- C = (4.0, 0.0)

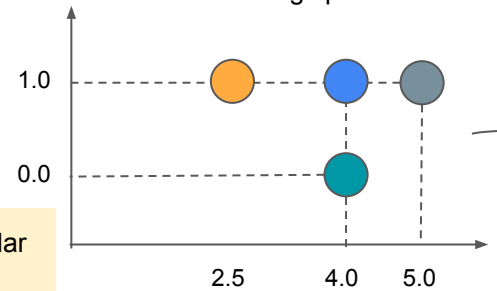
- The first column can be age,
- The second column can be a flag for the person's gender

In reality, each column will be assigned a weight, e.g., **w**, to represent the importance of different features



Weighted distribution (e.g., each column will be assigned a weight, **w**)

The final representation of nodes can be viewed in the following coordinate ~ this is called "node embedding", and the coordinate is called "embedding space"



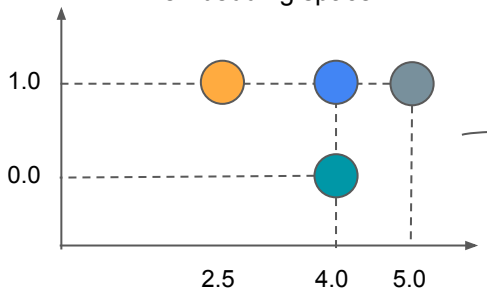
Nodes are similar will be more closer to each other in the embedding space

Step 3: Training

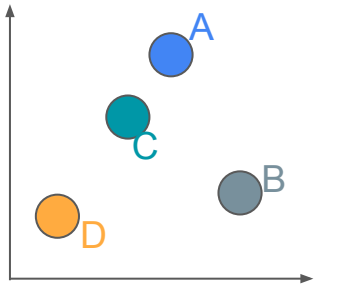
After the message passing, in order to make the explanation easier, assuming that we can represent each node with a two dimensional tuple:

- A = (4.0, 1.0)
- D = (2.5, 1.0)
- B = (5.0, 1.0)
- C = (4.0, 0.0)

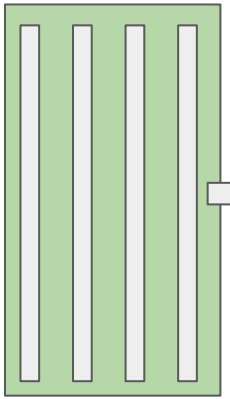
The final representation of nodes can be viewed in the following coordinate ~ this is called “node embedding”, and the coordinate is called “embedding space”



In reality, each column will be assigned a weight, e.g., **W**, to represent the importance of different features



Weighted distribution (e.g., each column will be assigned a weight, **W**)



A label indicates whether A,B,C and D are Andy’s friend or not (or the percentage that they are Andy’s friend)

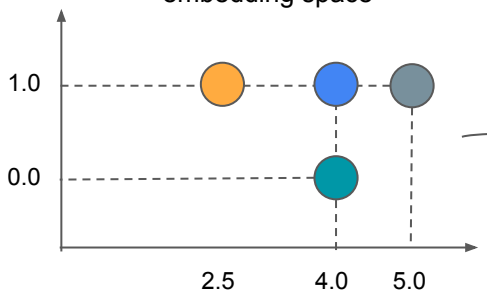
A regular neural network (e.g., DNN): the training will learn the weights **W** from the embedding space, and the regular parameters in DNN

Step 3: Training

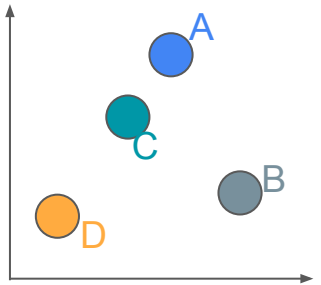
After the message passing, in order to make the explanation easier, assuming that we can represent each node with a two dimensional tuple:

- A = (4.0, 1.0)
- D = (2.5, 1.0)
- B = (5.0, 1.0)
- C = (4.0, 0.0)

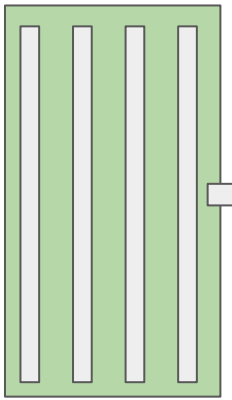
The final representation of nodes can be viewed in the following coordinate ~ this is called “node embedding”, and the coordinate is called “embedding space”



In reality, each column will be assigned a weight, e.g., **W**, to represent the importance of different features/nodes



Weighted distribution (e.g., each column will be assigned a weight, **W**)



A label indicates whether A,B,C and D are Andy’s friend or not (or the percentage that they are Andy’s friend)

A regular neural network (e.g., DNN): the training will learn the weights **W** from the embedding space, and the regular parameters in DNN

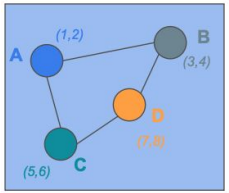
Math behind GNN

Step 1: The message passing step can be represented by a function, e.g., when the Node A is receiving messages, we have

$$xa = xa + xb + xc + xd$$

New msg for A

Existing msg from A, B, C and D



Step 2: We also can assign weight to each node to represent their importances, e.g.,

$$xa = wa*xa + wb*xb + wc*xc + wd*xd$$

wb, wc and wd here are weights representing the importances of Node A to D

Step 3: We also can assign weight to each feature of a node to represent their importances, e.g.,

$$xa = wa*xa*Wf + wb*xb*Wf + wc*xc*Wf + wd*xd*Wf$$

"Wf" here represents the importance of each feature of the node

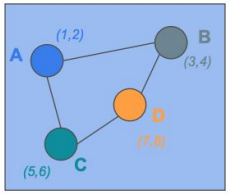
Step 4: Nonlinearity can be added to the message passing process with a function such Sigmoid function

$$xa = \sigma\,(wa*xa*Wf + wb*xb*Wf + wc*xc*Wf + wd*xd*Wf)$$

σ represents the Sigmoid function

How the final representation of a node (from Step 4) is calculated ?

$$xa = \sigma\,(wa*xa*Wf + wb*xb*Wf + wc*xc*Wf + wd*xd*Wf)$$



	a	b	c	d
a	wa	0	0	0
b	0	wb	0	0
c	0	0	wc	0
d	0	0	0	wd

Weights (importance) of nodes matrix

(4*4)

	a	b	c	d
a	1	1	1	0
b	1	0	0	1
c	1	0	0	1
d	0	1	1	0

Adjacency matrix (4*4)

w (wa, wb, wc, wd)

	Feature 1	Feature 2
a	agea	sexa
b	ageb	sexb
c	agec	sexc
d	aged	sexd

Feature matrix

(4*2)

	Feature 1	Feature 2
Feature 1	W(age)	0
Feature 2	0	W(sex)

Feature importance

(2*2)

xWf

(xa*Wf, xb*Wf, xc*Wf, xd*Wf)

=

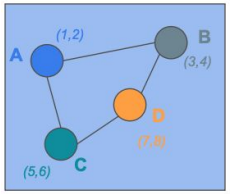
a	xa
b	xb
c	xc
d	xd

New node representations

(with σ)

How the final representation of a node (from Step 4) is calculated ?

$$xa = \sigma\,(wa*xa*Wf + wb*xb*Wf + wc*xc*Wf + wd*xd*Wf)$$



	a	b	c	d
a	1	1	1	0
b	1	0	0	1
c	1	0	0	1
d	0	1	1	0

Adjacency matrix (4*4)

a	wa
b	wb
c	wc
d	wd

Weights (importance) of nodes (4*1)

We can represent the weights as a diagonal matrix

	a	b	c	d
a	wa	0	0	0
b	0	wb	0	0
c	0	0	wc	0
d	0	0	0	wd

Weights (importance) of nodes (4*4)

Element multiplication operation

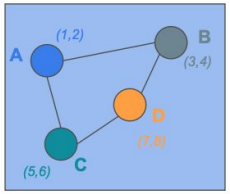
	a	b	c	d
a	wa	wb	wc	0
b	wa	0	0	wd
c	wa	0	0	wd
d	0	wb	wc	0

w (wa, wb, wc, wd)

(4*4)

How the final representation of a node (from Step 4) is calculated ?

$$xa = \sigma(wa*xa*Wf + wb*xb*Wf + wc*xc*Wf + wd*xd*Wf)$$



	a	b	c	d
a	wa	wb	wc	0
b	wa	0	0	wd
c	wa	0	0	wd
d	0	wb	wc	0

w (wa, wb, wc, wd)

(4*4)

X

	Feature 1	Feature 2
a	W(age)*agea	W(sex)*sexa
b	W(age)*ageb	W(sex)*sexb
c	W(age)*agec	W(sex)*sexc
d	W(age)*aged	W(sex)*sexd

xWf

(xa*Wf, xb*Wf, xc*Wf, xd*Wf)

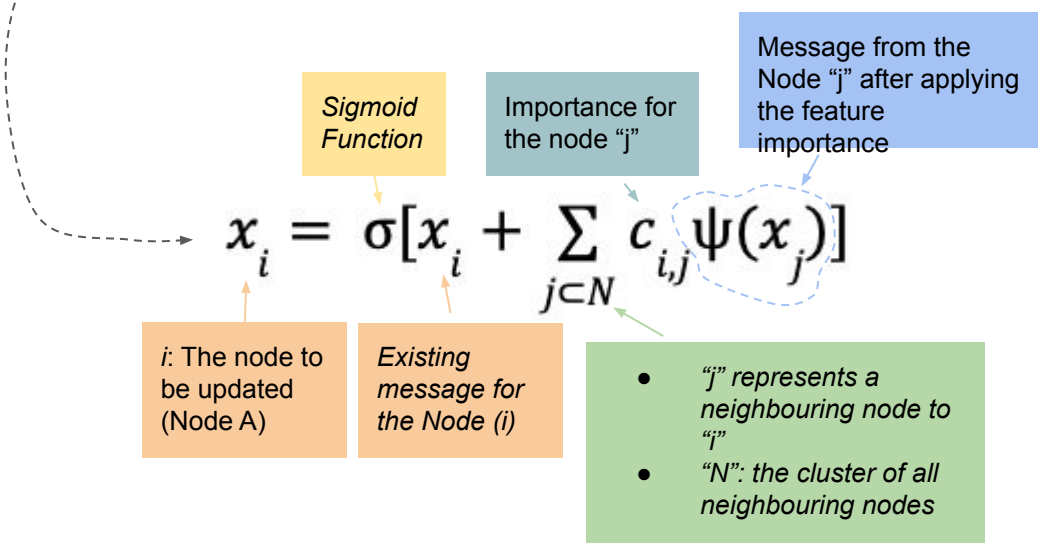
(4*2)

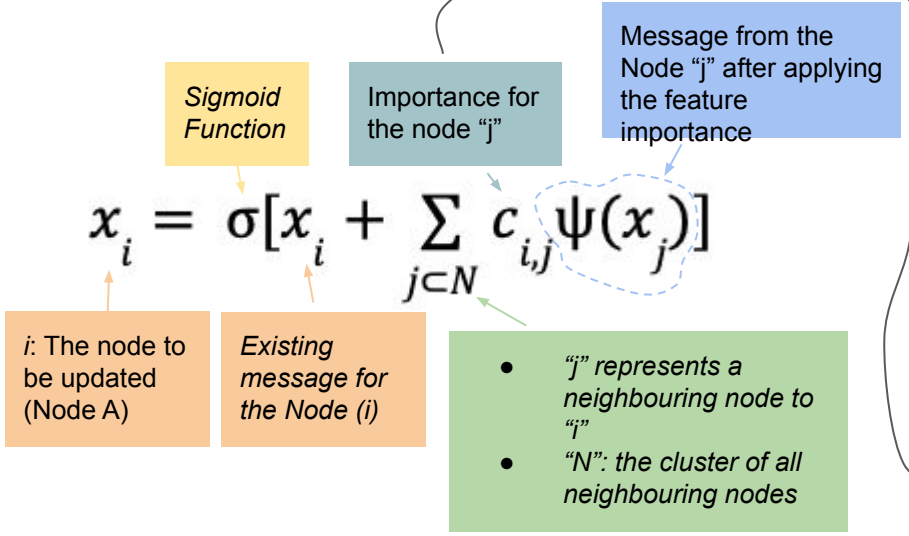
a	xa
b	xb
c	xc
d	xd

New node representations

Combing Step 1 to Step 4, we can get a common function as:

xa = σ (xa + wb*xb*Wf + wc*xc*Wf + wd*xd*Wf)





Approach 1 (convolutional):

The Importance for a node can be fixed based on the graph structure (e.g., it equals to the number of external connections of a node)

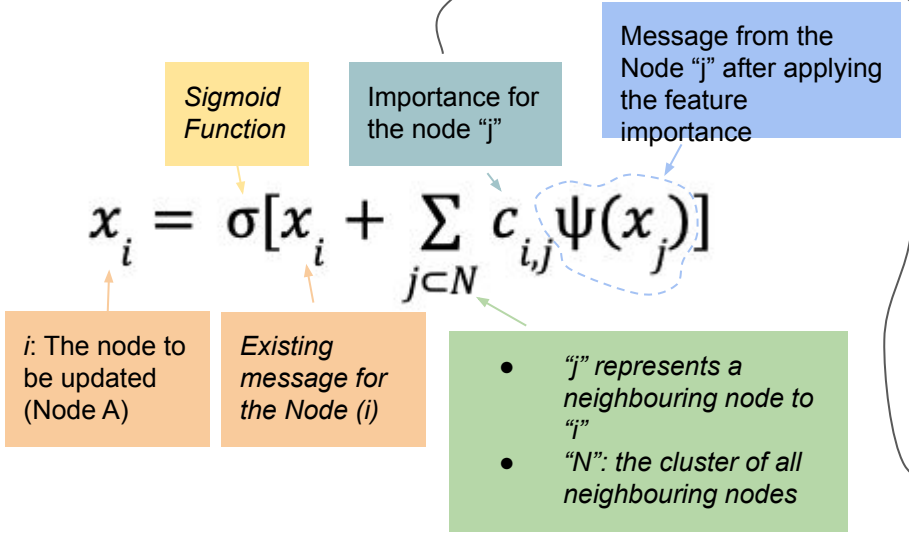
Approach 2 (attentional):

The Importance for a node is learned based on the interaction of features between different nodes (Node "i" and Node "j").

$$c_{i,j} = a(x_i, x_j)$$

So the message passing function can be written as:

$$x_i = \sigma[x_i + \sum_{j \subset N} a(x_i, x_j)\psi(x_j)]$$



Approach 1 (convolutional):

The Importance for a node can be fixed based on the graph structure (e.g., it equals to the number of external connections of a node)

Approach 2 (attentional):

The Importance for a node is learned based on the interaction of features between different nodes (Node "i" and Node "j").

$$c_{i,j} = a(x_i, x_j)$$

So the message passing function can be written as:

$$x_i = \sigma[x_i + \sum_{j \subset N} a(x_i, x_j)\psi(x_j)]$$