Deep Learning 1

2024-2025 – Pascal Mettes

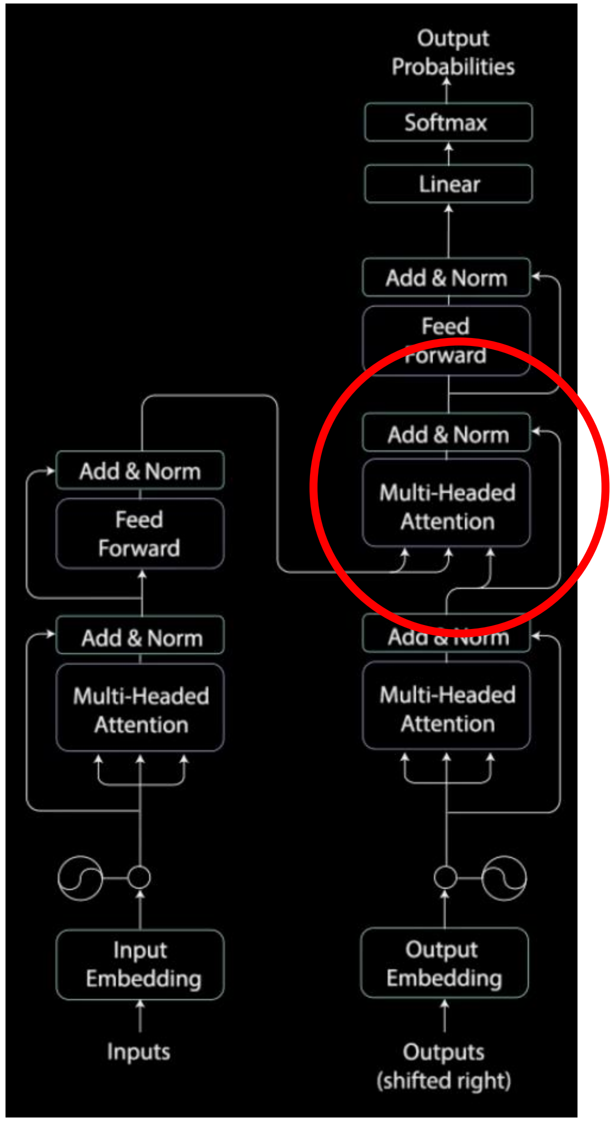
Lecture 8

Graph Neural Networks

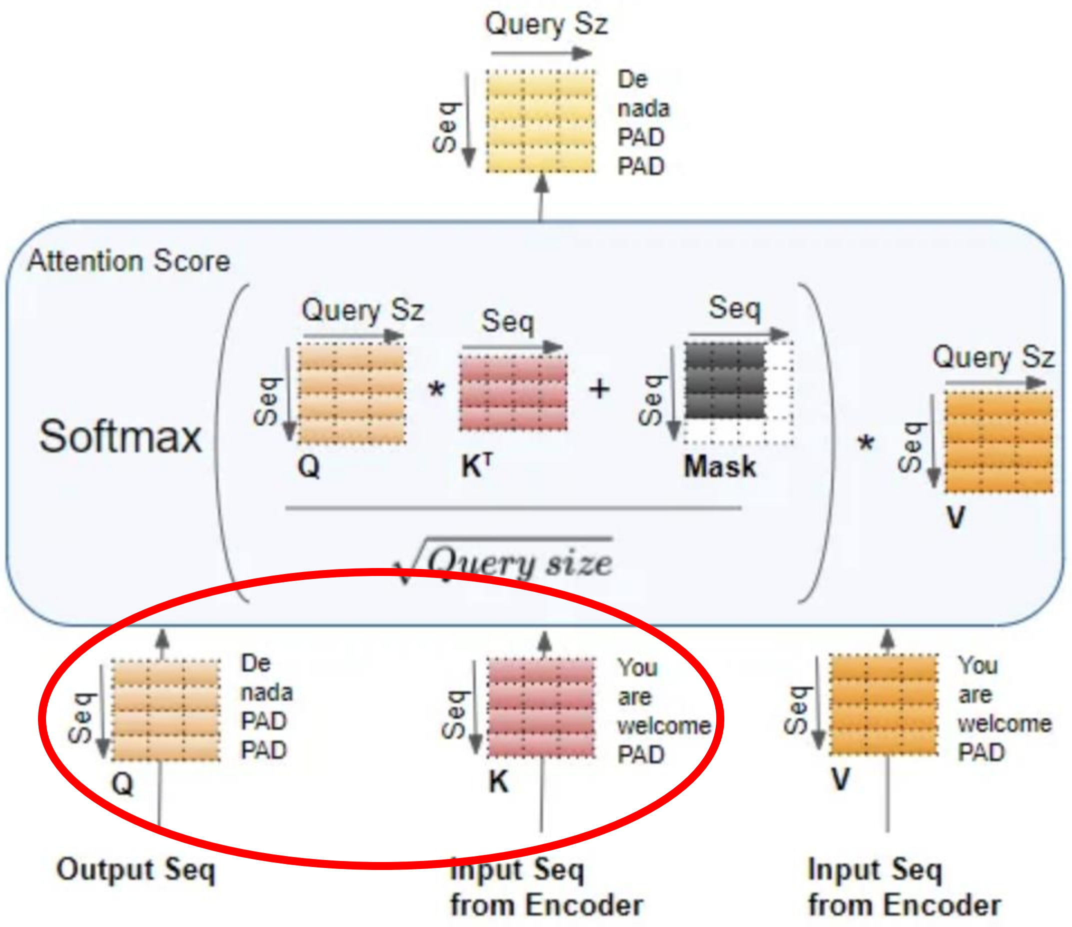
Previous lecture

Lecture	Title	Lecture	Title
1	Intro and history of deep learning	2	Manually forward, automatically backward
3	Deep learning optimization I	4	Deep learning optimization II
5	Convolutional Neural Networks I	6	Convolutional Neural Networks II
7	Attention	8	Graph Neural Networks
9	Self-supervised and vision-language learning	10	Auto-encoding and generation
11	The oddities of deep learning	12	Non-Euclidean deep learning
13	Deep learning for videos	14	Q&A

Open question from last lecture



What is going on here?



Useful source: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853

This lecture

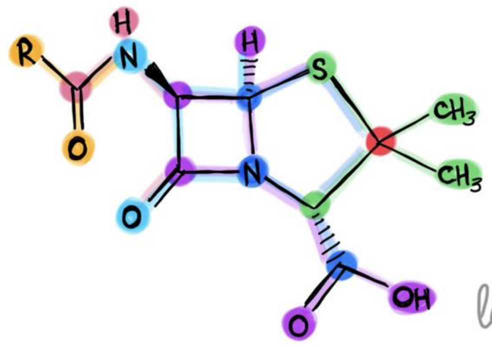
Graphs

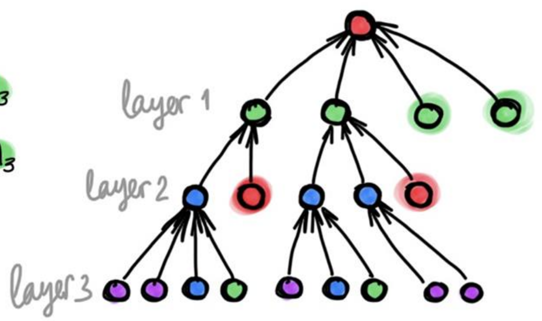
Graph convolutions

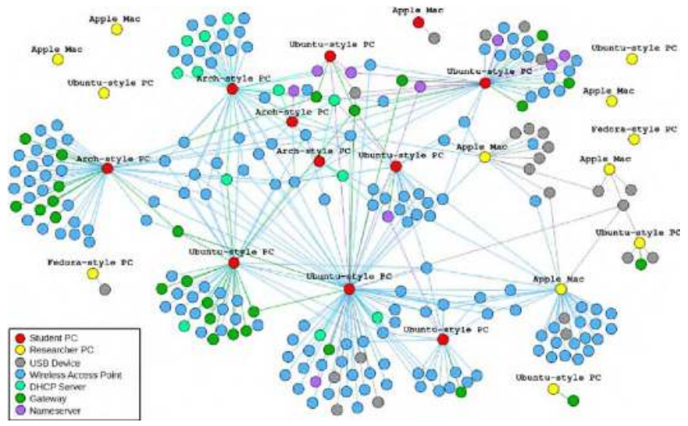
Graph attention

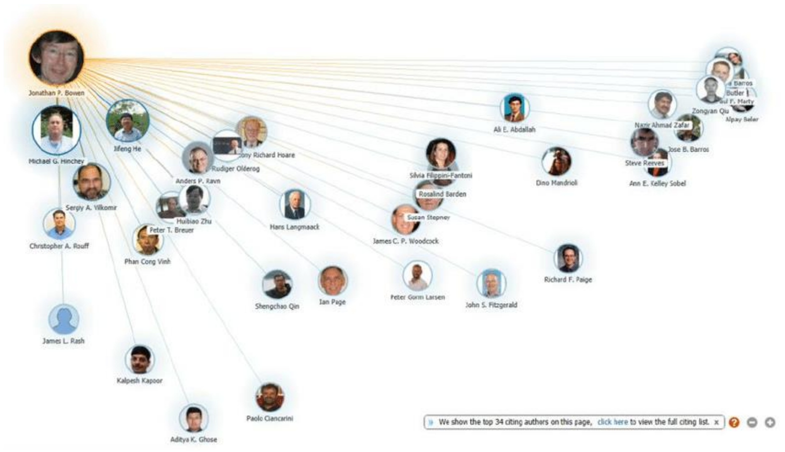
Graph applications

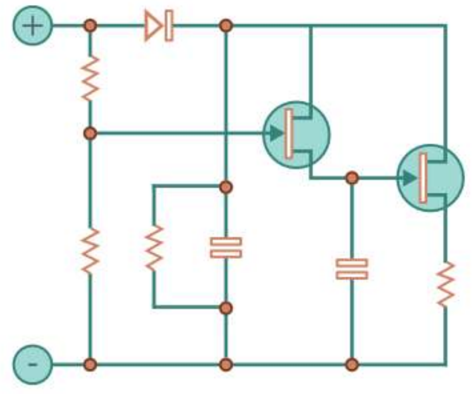
Graphs, more common than you think



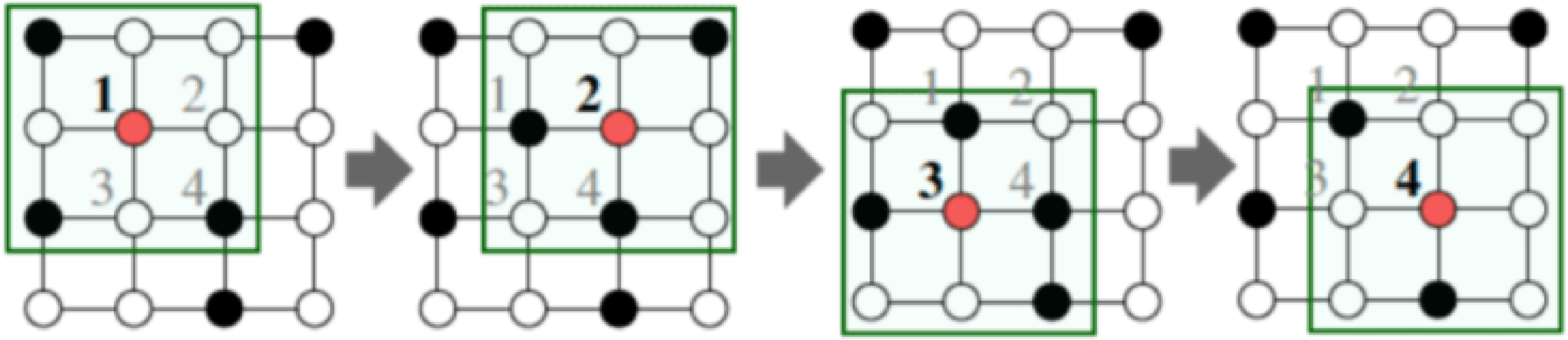




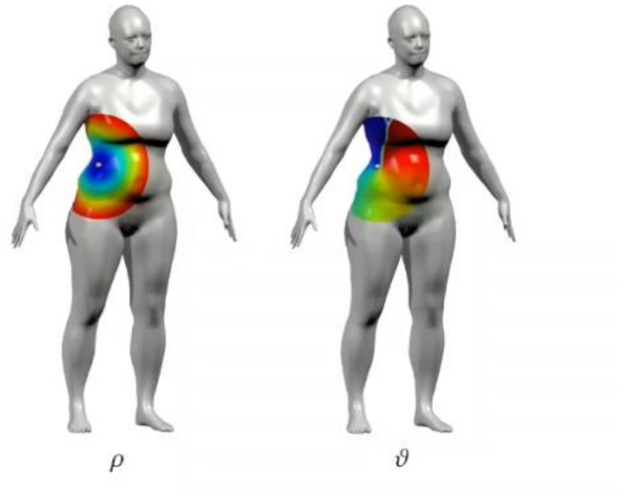


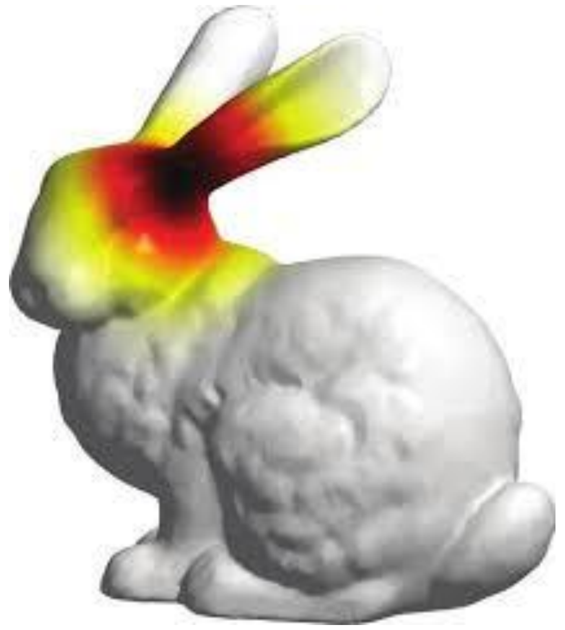


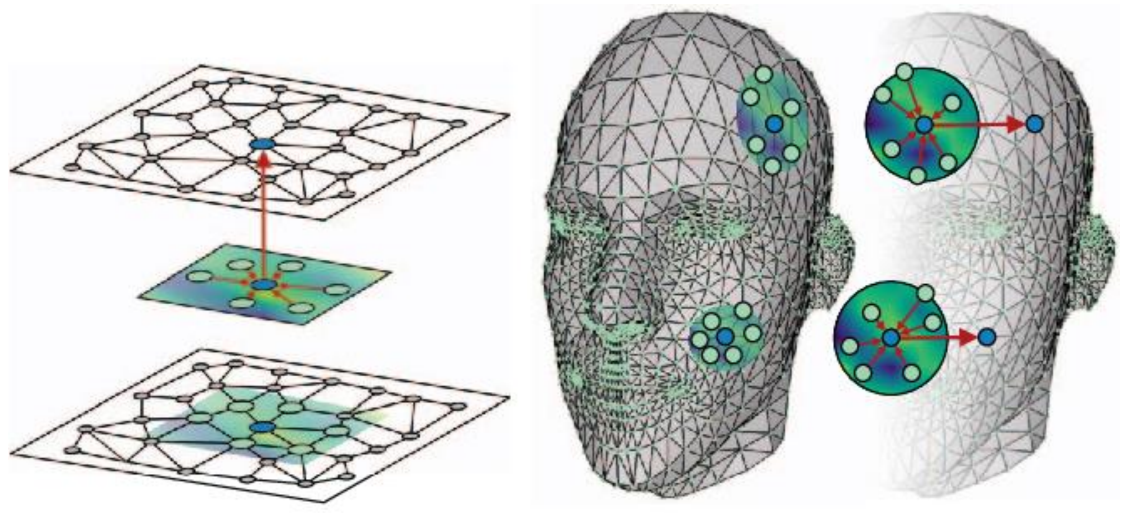
Many structures are special cases of graphs

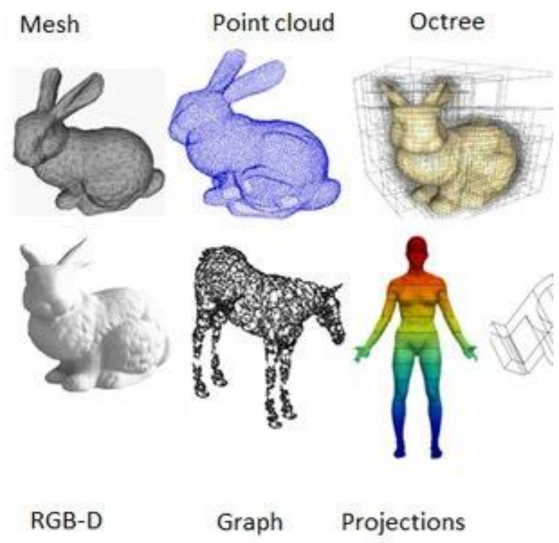


Graphs as geometry



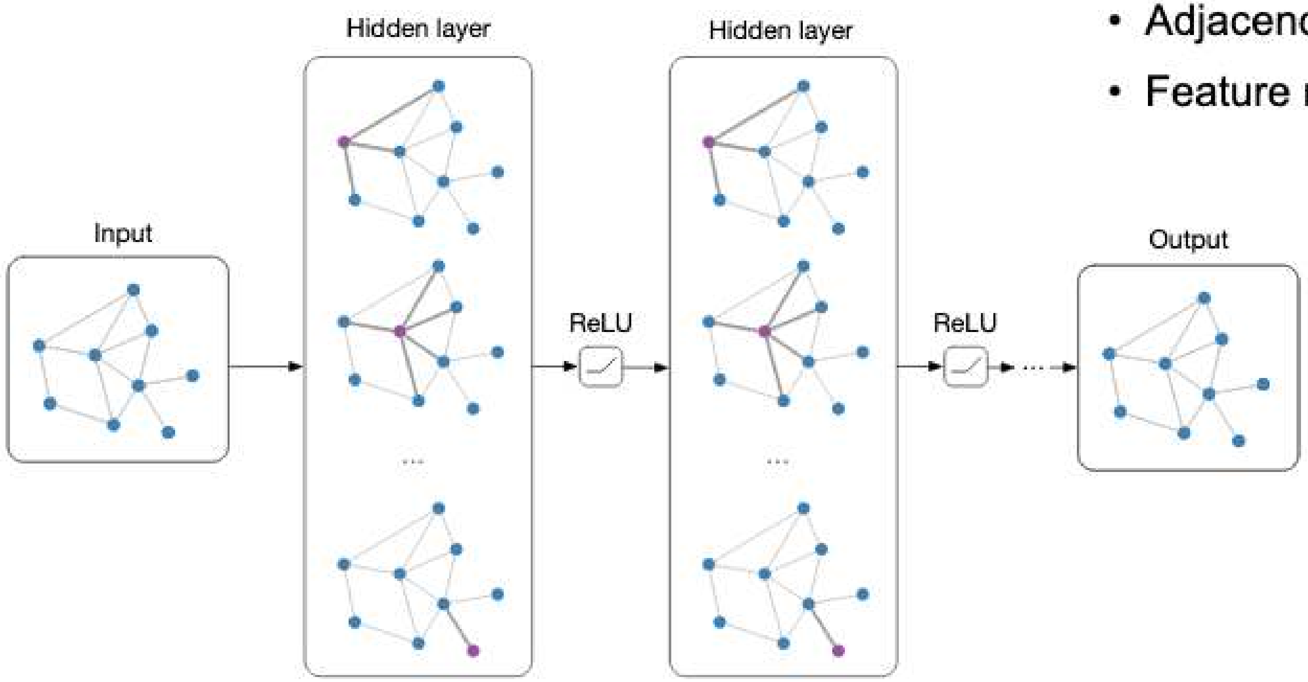






What are graph networks?

The bigger picture:



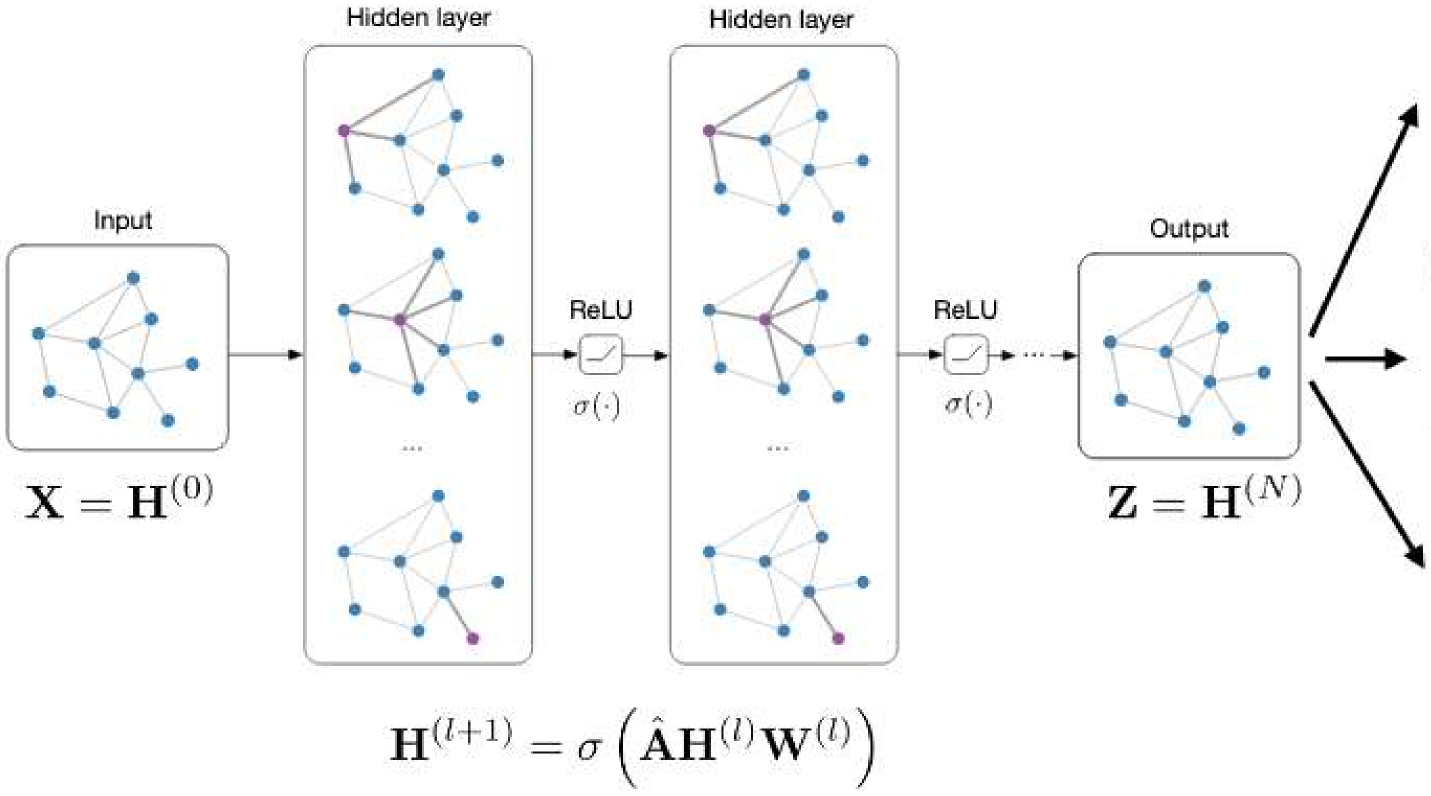
Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$

Main idea: Pass messages between pairs of nodes & agglomerate

Graph networks

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



Input
Hidden layer
Hidden layer
ReLU
$\sigma(\cdot)$
ReLU
$\sigma(\cdot)$
Output
$\mathbf{X} = \mathbf{H}^{(0)}$
$\mathbf{Z} = \mathbf{H}^{(N)}$
$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$

Node classification:

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\mathrm{softmax}(\textstyle\sum_n \mathbf{z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

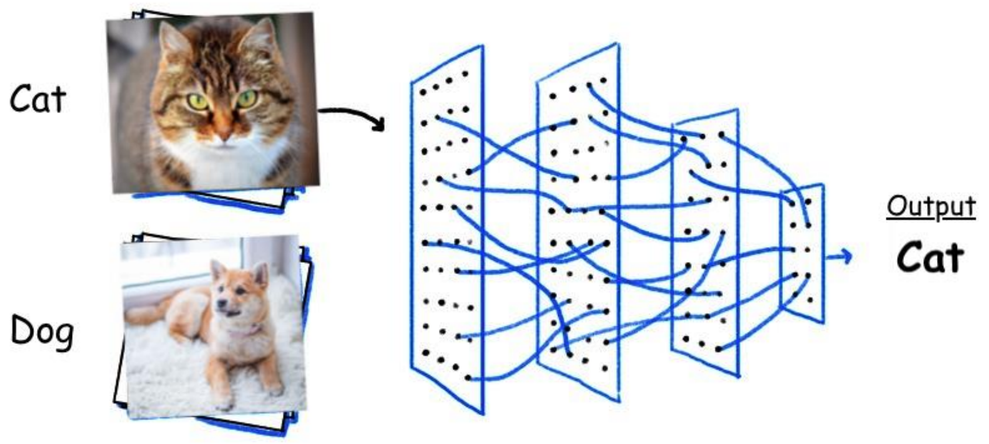
$$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$$

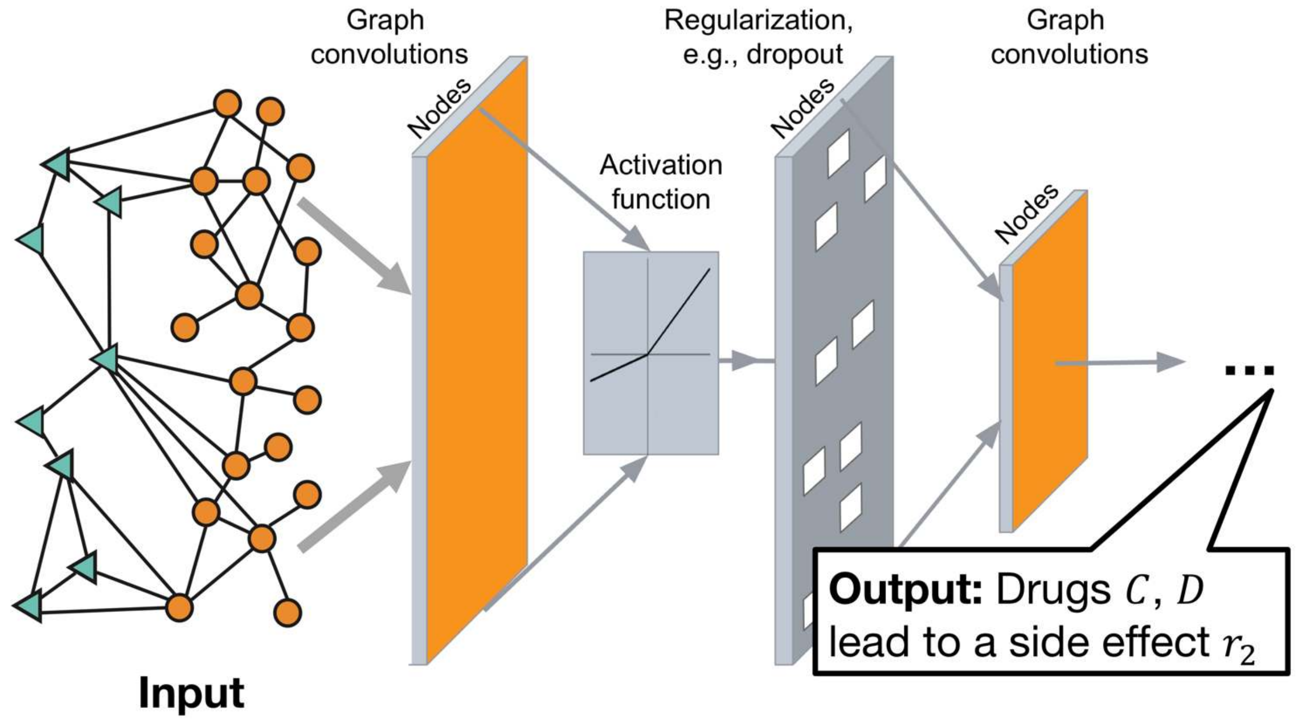
Kipf & Welling (NIPS BDL 2016)
"Graph Auto-Encoders"

1) Graph classification

Make a prediction over the entire graph.

Akin to assigning a label to an entire image.

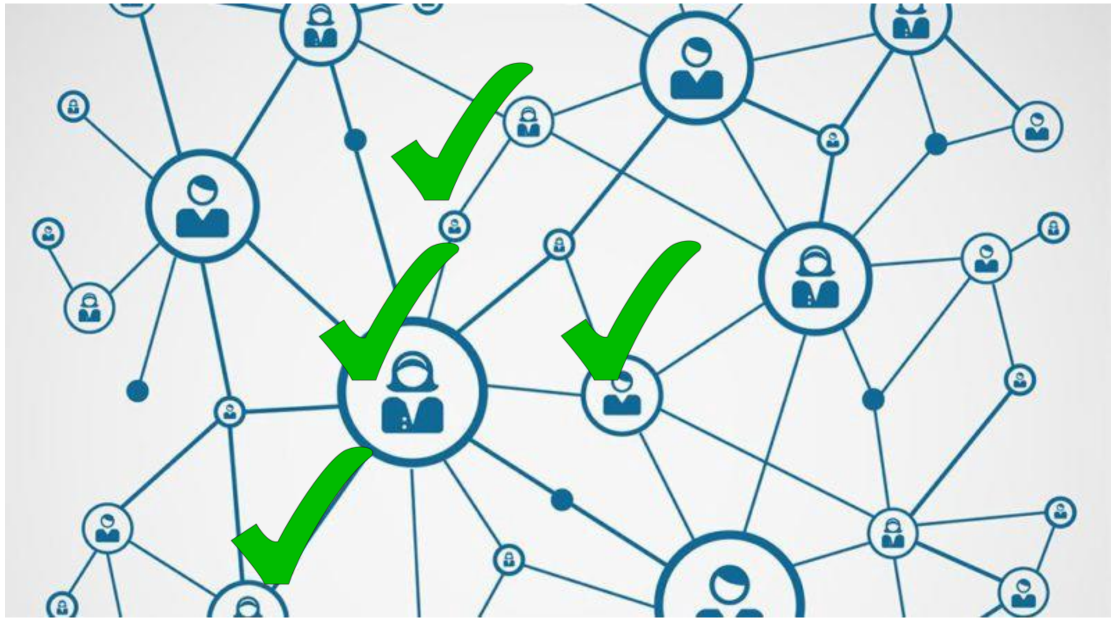



Graph convolutions
Regularization, e.g., dropout
Graph convolutions
Nodes
Nodes
Nodes
Activation function
...
Output: Drugs C, D lead to a side effect r_2
Input

2) Node classification

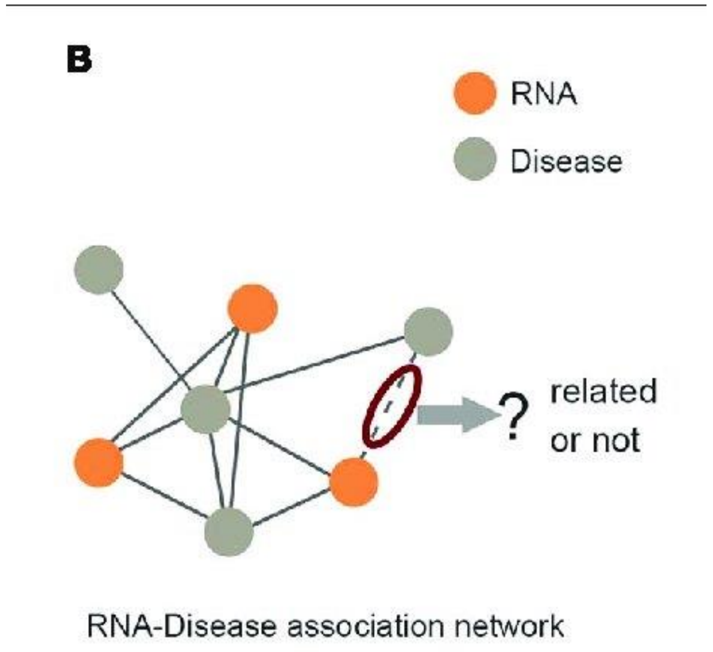
Make a prediction for each individual node.

Akin to segmentation for images.



3) Link prediction

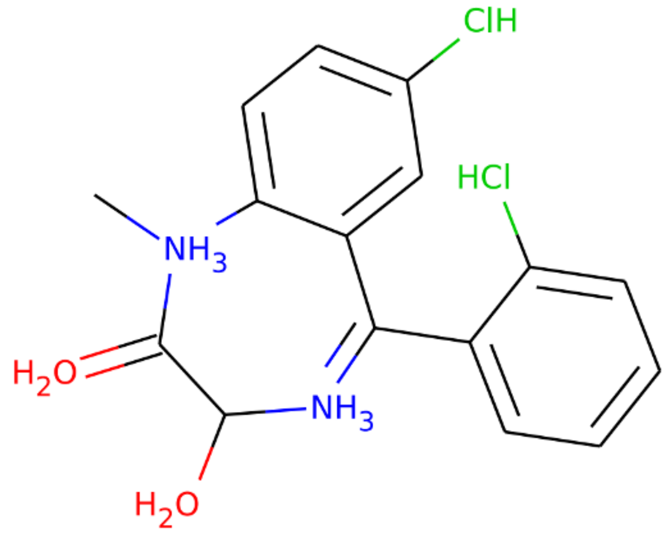
Make a prediction for each edge between two nodes.



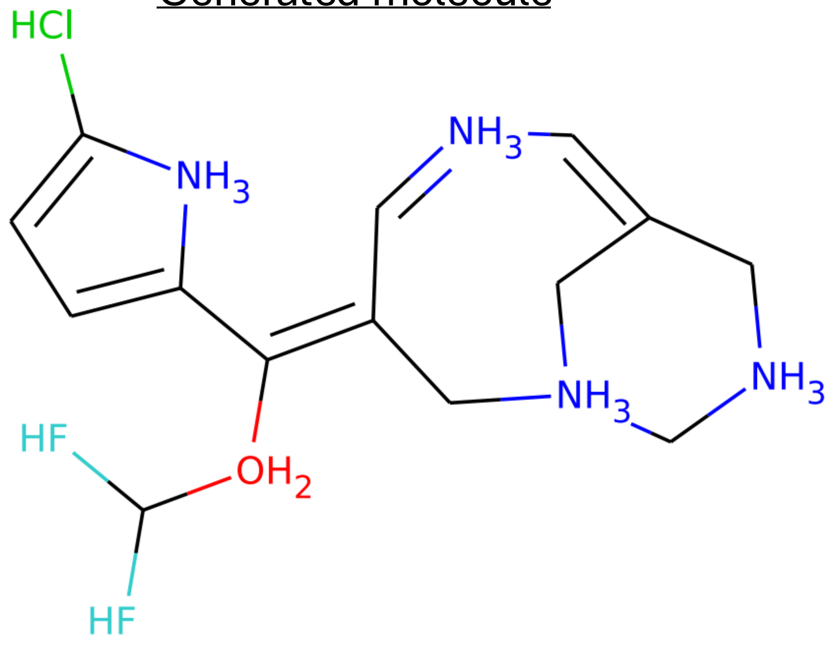
4) Graph generation

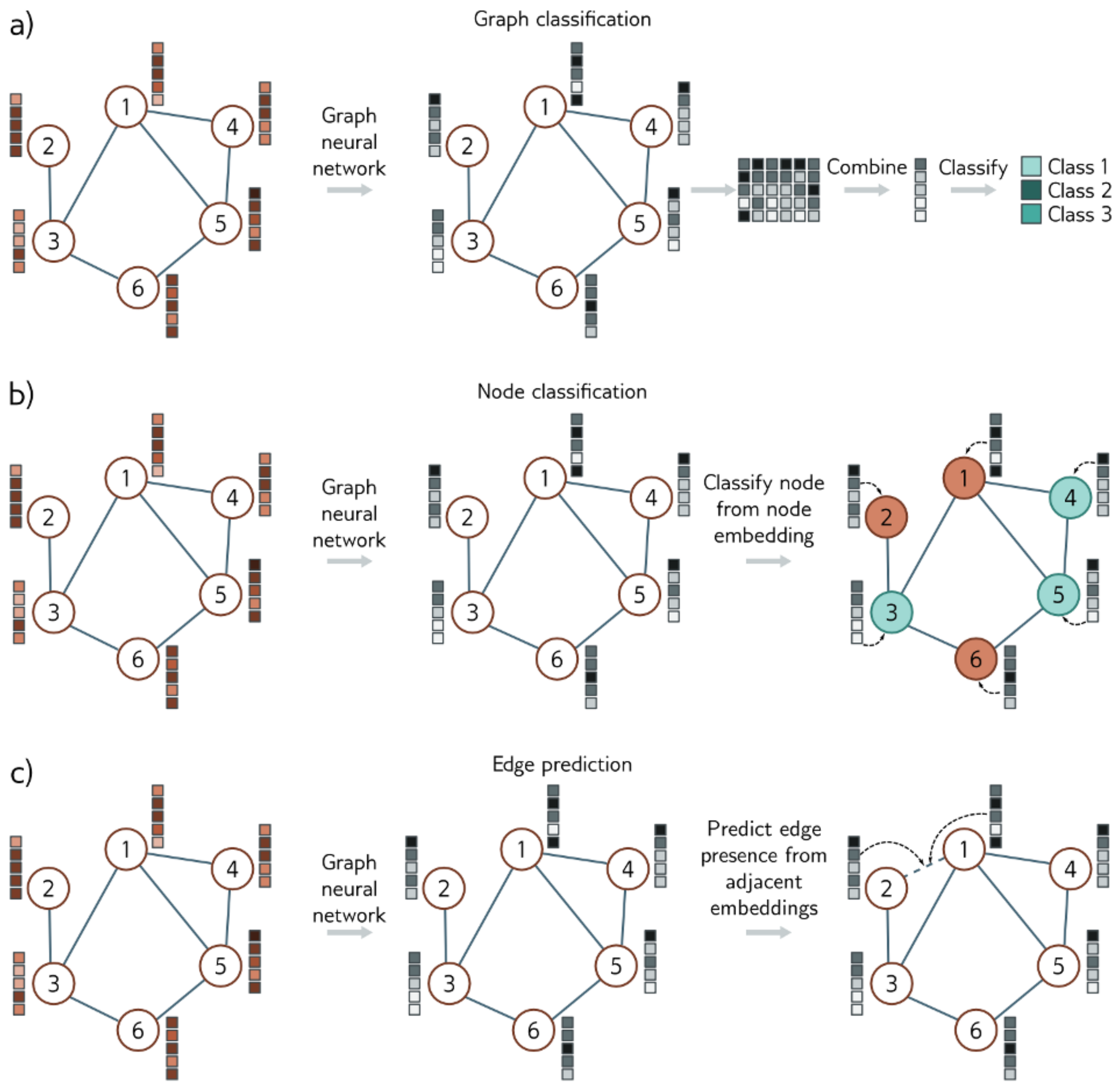
Similar in spirit to image/text generation, topic of next week.

Example molecule



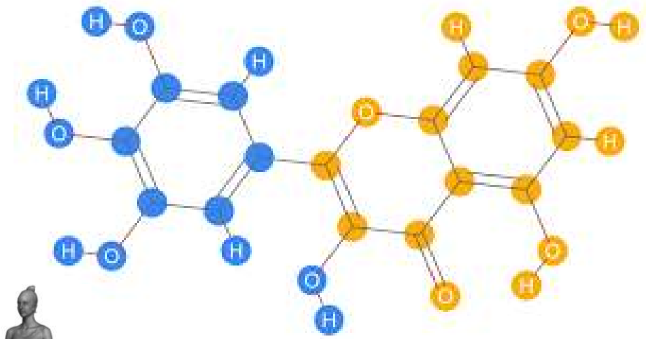
Generated molecule



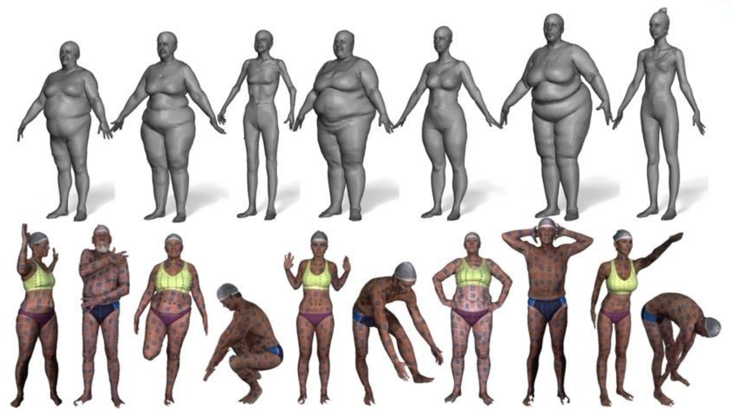

a)
Graph classification
Graph neural network
Combine
Classify
Class 1
Class 2
Class 3
b)
Node classification
Graph neural network
Classify node from node embedding
c)
Edge prediction
Graph neural network
Predict edge presence from adjacent embeddings

Graphs can be dynamic

Graphs have fixed structures.



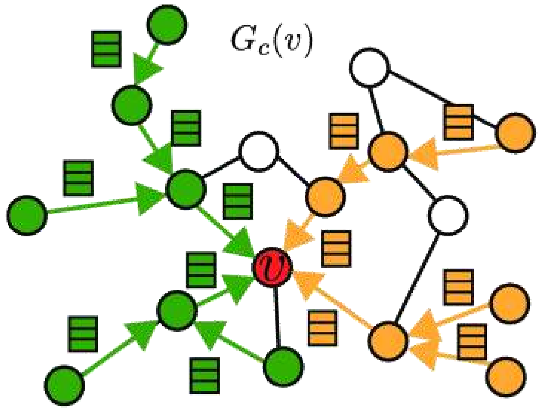
But many are subject to change.



In practice, this change can even be gradual and continuous.

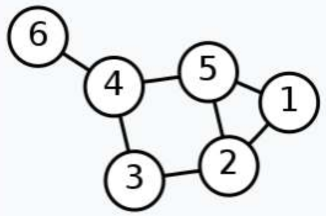
Regular structures and graphs

Regular structures are a subset of graphs. I.e., images are grid graphs.



- Convolution + pooling
- Local neighborhood: fixed window
- Constant number of neighbors
- With fixed ordering
- Translation equivariance

- Message passing + coarsening
- Local neighborhood: 1-hop
- Different number of neighbors
- No ordering of neighbors
- Local permutation equivariance

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

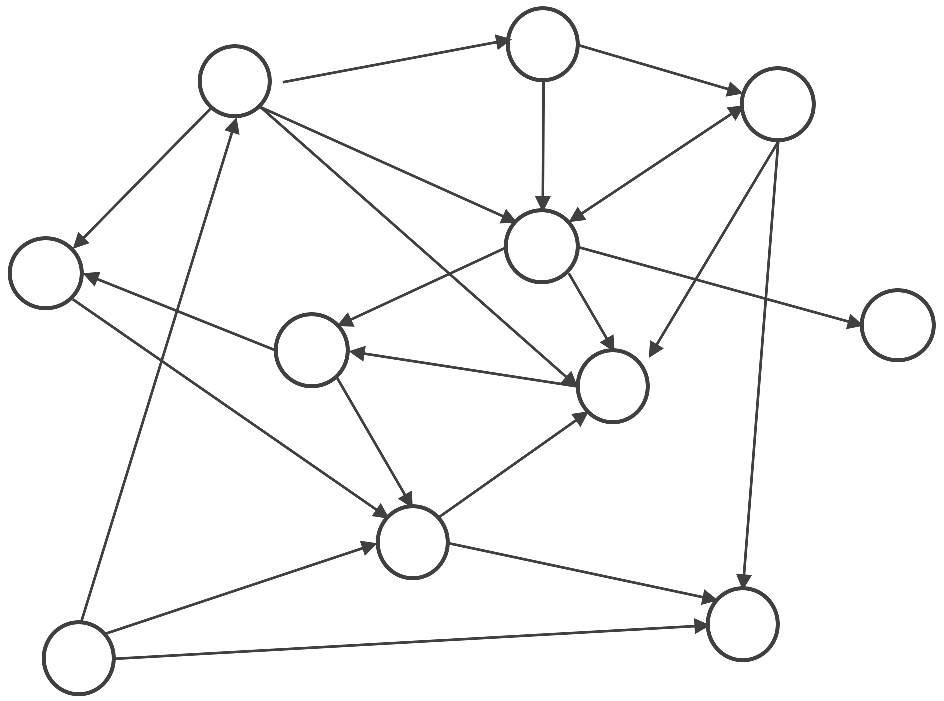
Definition of a graph

(in deep learning)

Directed graphs

Vertices $\mathcal{V} = \{1, \ldots, n\}$, also called “nodes”

Edges $\mathcal{E} = \{(i, j) \colon i, j \in \mathcal{V}\} \subseteq \mathcal{V} \times \mathcal{V}$ (directed)

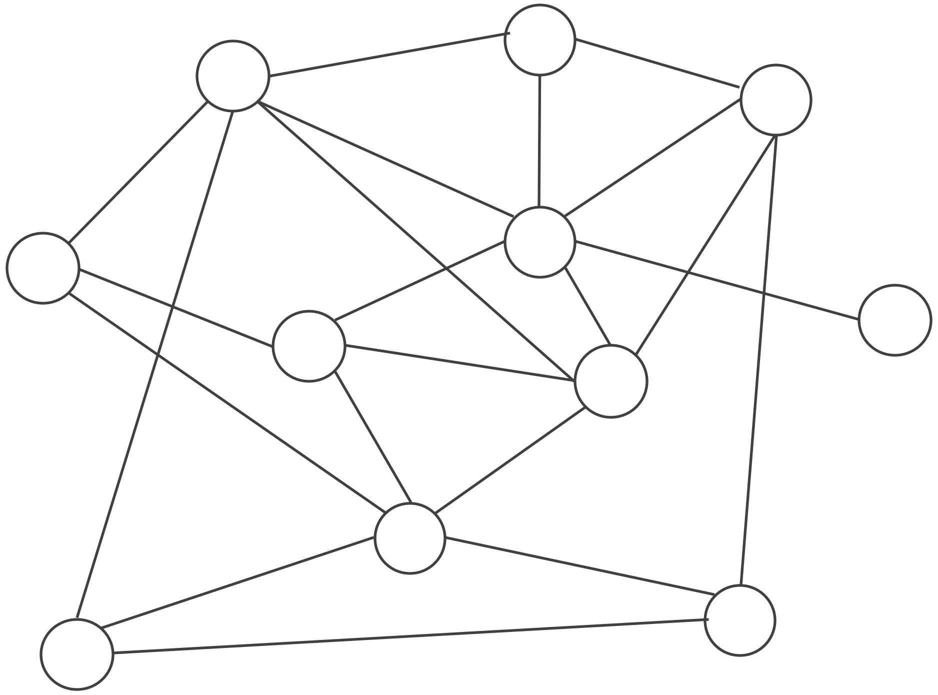


Undirected graphs

Vertices $\mathcal{V} = \{1, \dots, n\}$

Edges $\mathcal{E} = \{(i, j): i, j \in \mathcal{V}\} \subseteq \mathcal{V} \times \mathcal{V}$ (directed)

Edges $\mathcal{E} = \{\{i, j\}: i, j \in \mathcal{V}\} \subseteq \mathcal{V} \times \mathcal{V}$ (undirected)



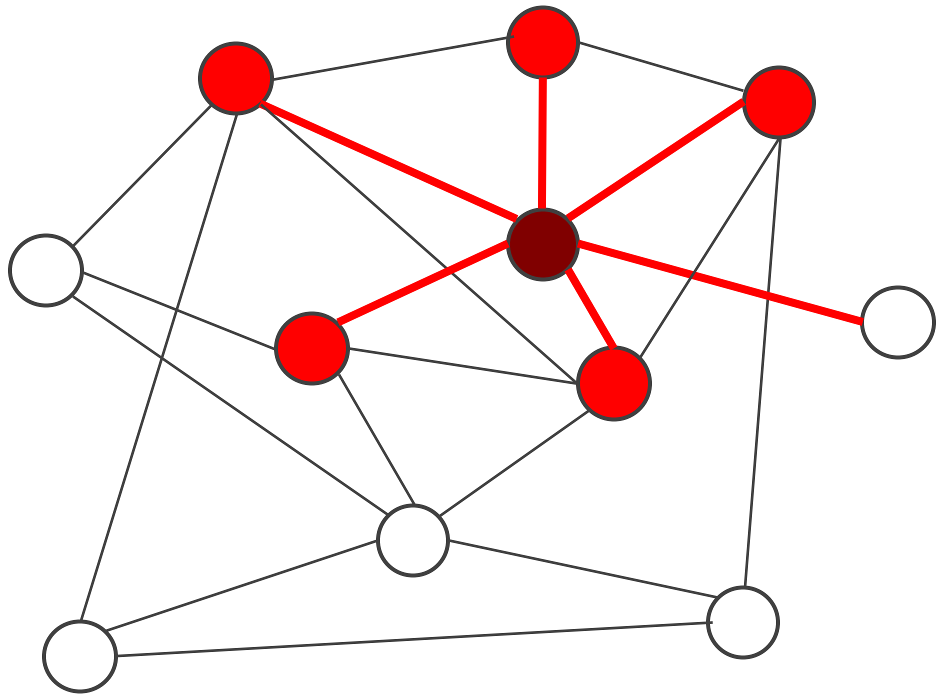
Graph neighborhood

The neighborhood of a node consists of all nodes directly connected to it

$$\mathcal{N}(i) = \{j: (i,j) \in \mathcal{E}\}$$

The **degree** of a node is the number of neighbors: $d_i = |\mathcal{N}(i)|$

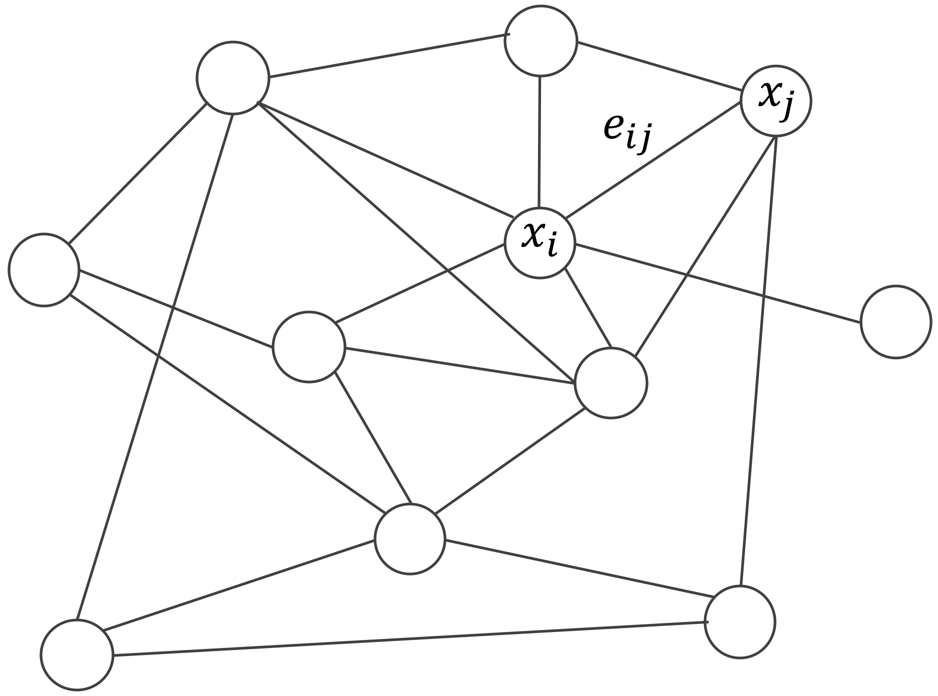
The diagonal matrix D contains all degrees per node



Attributes

Node features $\boldsymbol{x}$: $\mathcal{V} \to \mathbb{R}^d$, $X = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$

Edge features $\boldsymbol{e}_{ij}$: $\mathcal{E} \to \mathbb{R}^{d'}$

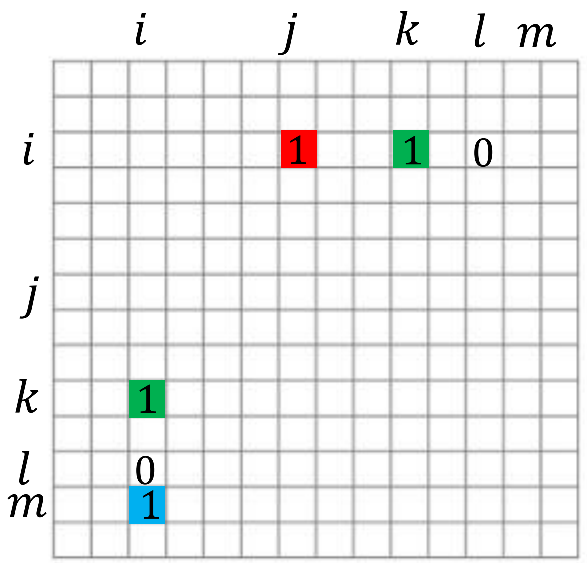


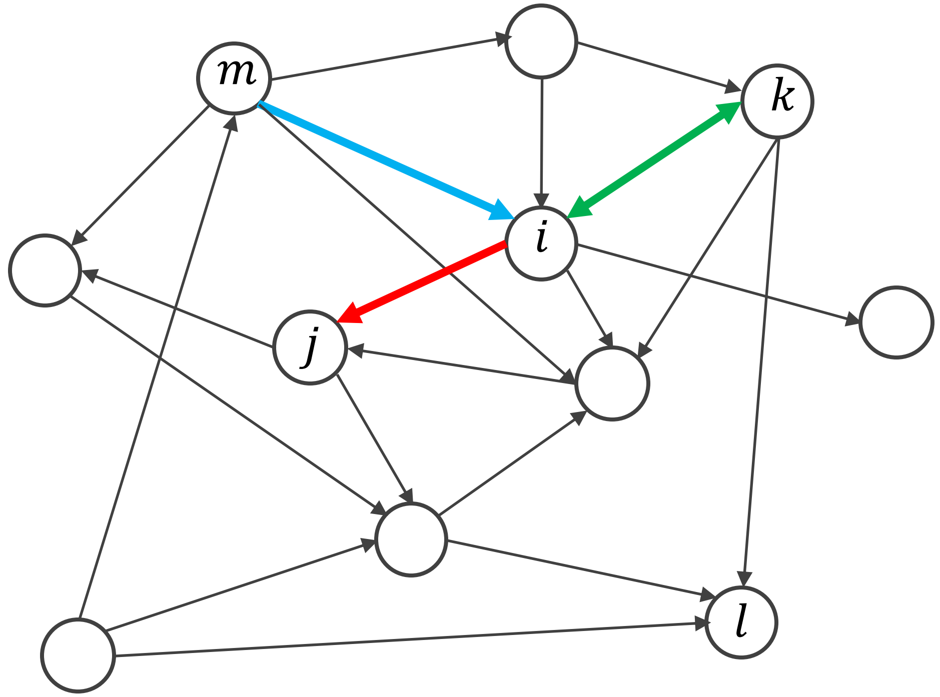
Adjacency matrix

An $n \times n$ matrix A, for n nodes

$$A_{ij} = \begin{cases} 1 \text{ if } (\mathrm{i}, \mathrm{j}) \in \mathcal{E} \\ 0 \text{ if } (\mathrm{i}, \mathrm{j}) \notin \mathcal{E} \end{cases}$$

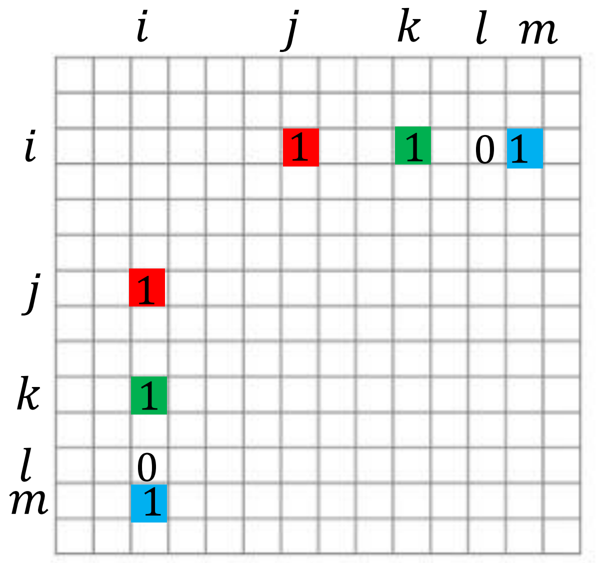
$(A^z)_{ij}$: number of paths that go from i to j in z steps

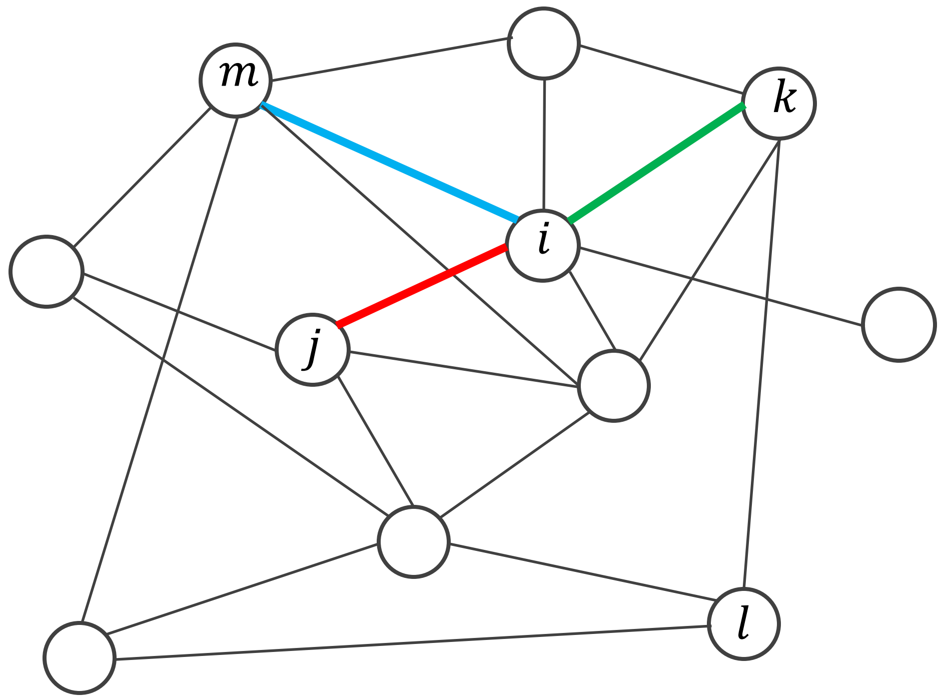




Adjacency matrix for undirected graphs

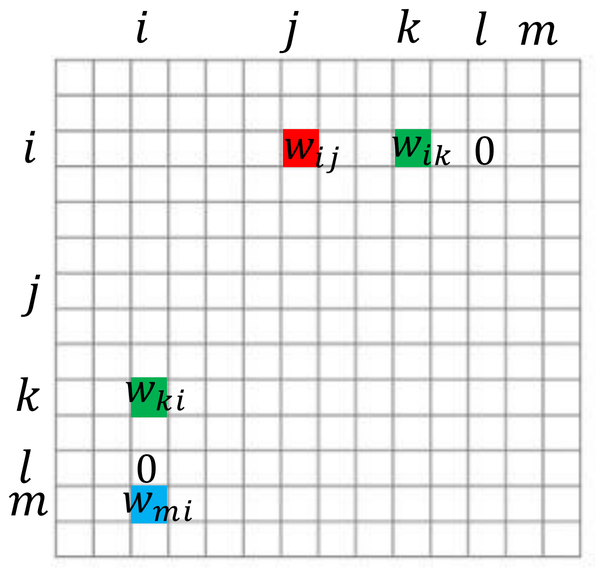
The adjacency matrix is symmetric for undirected graphs.

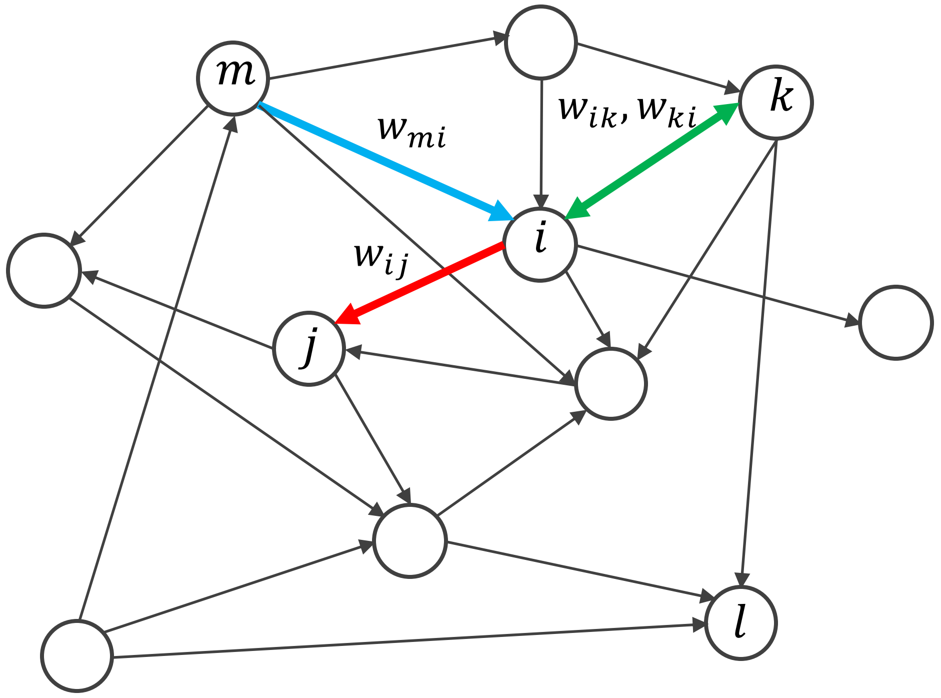




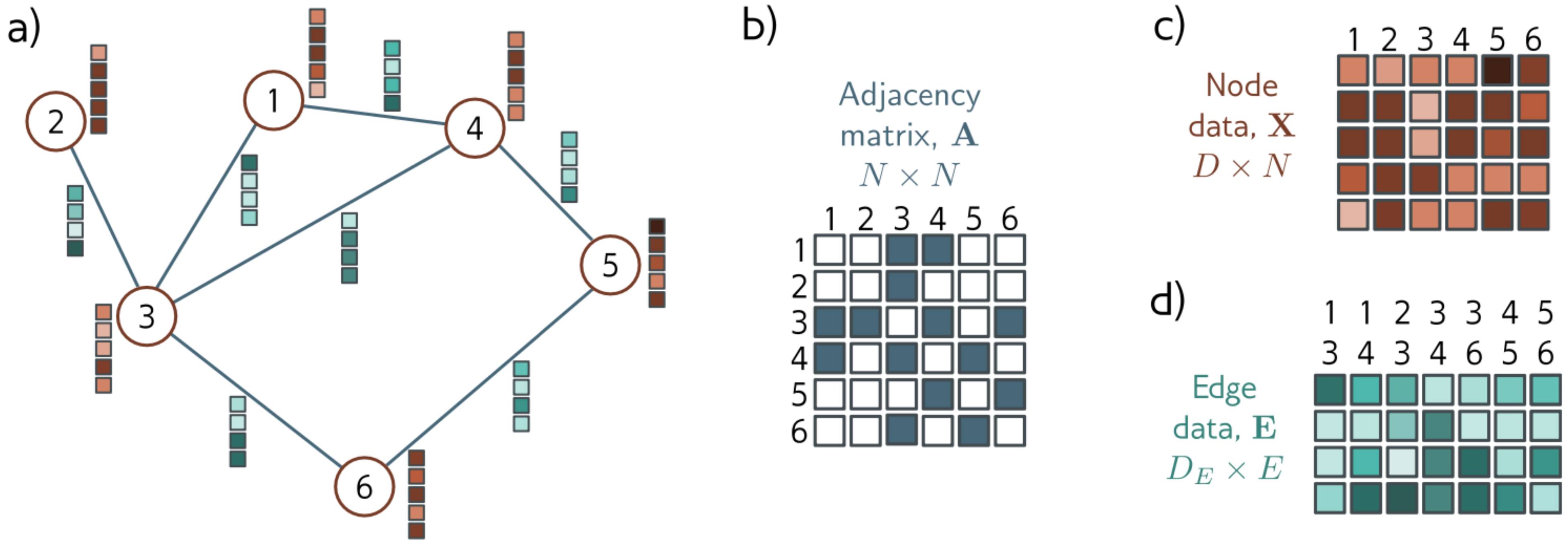
Weighted adjacency matrix

When the edges have weights, so does the adjacency matrix.





Final graph input representation

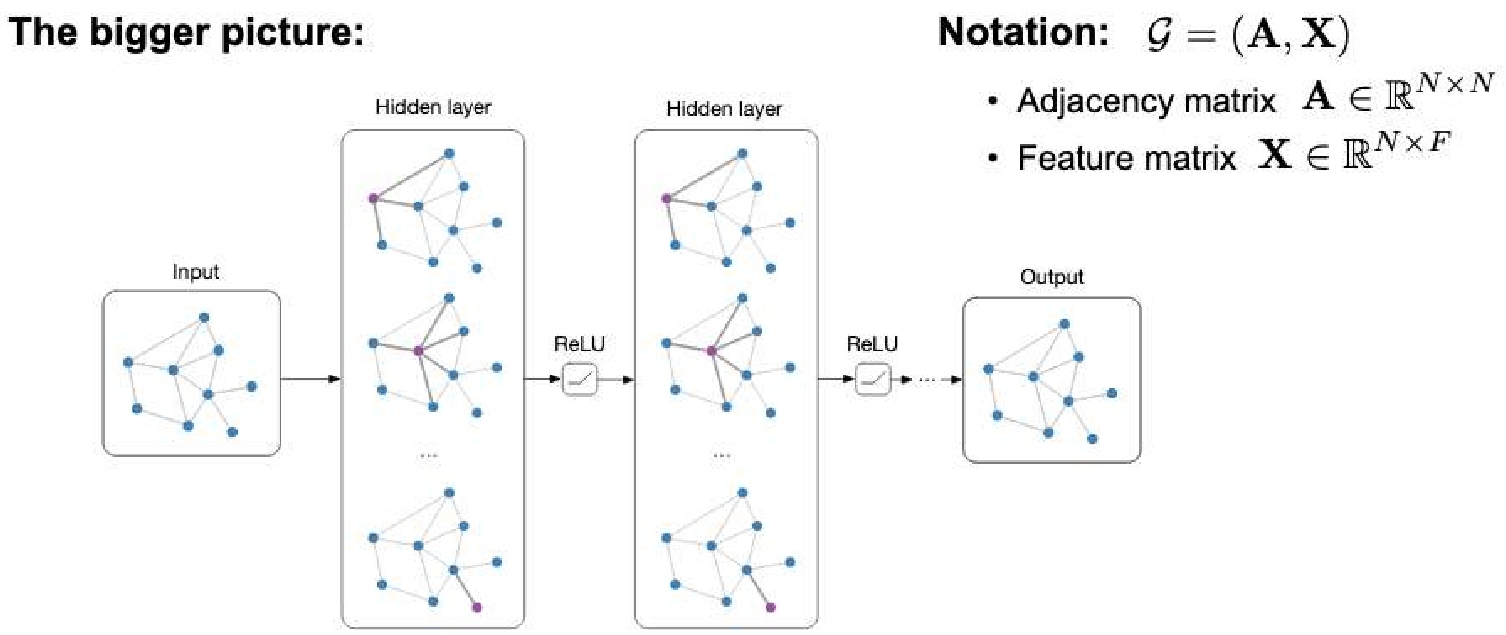


Back to graph networks

We can do a lot of processing on this data structure.

But the pre-defined features are raw inputs.

Graph networks do 1 thing: transform the feature vector per node over layers.

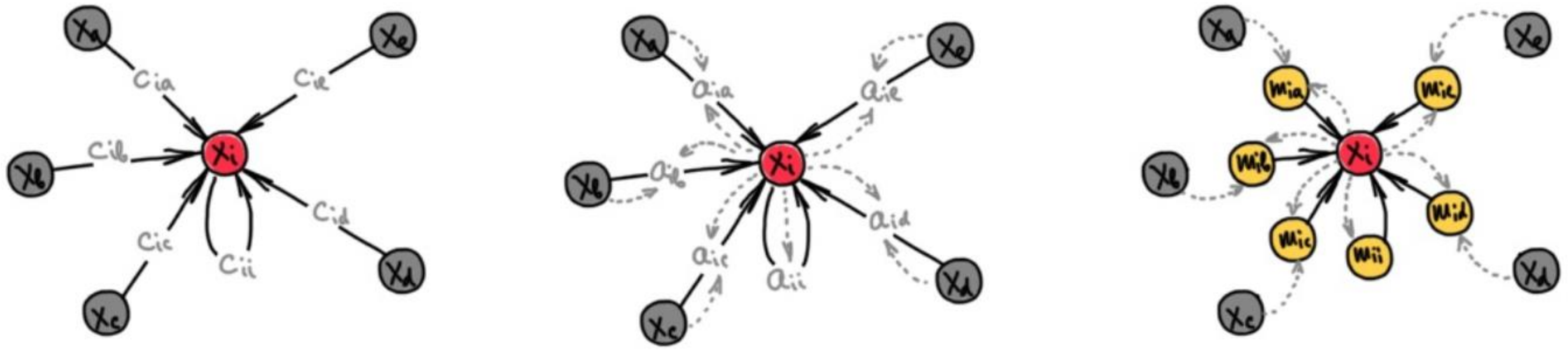


The bigger picture:

Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$

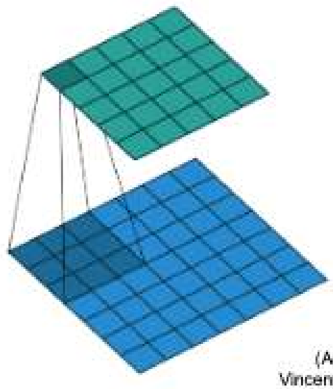
Three perspectives to graph networks



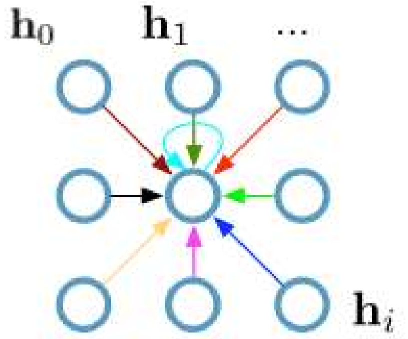
Three “flavours” of GNNs, left-to-right: convolutional, attentional, and general nonlinear message passing flavours. All are forms of message passing. Figure adapted from P. Veličković.

Graph layer as a convolution layer

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

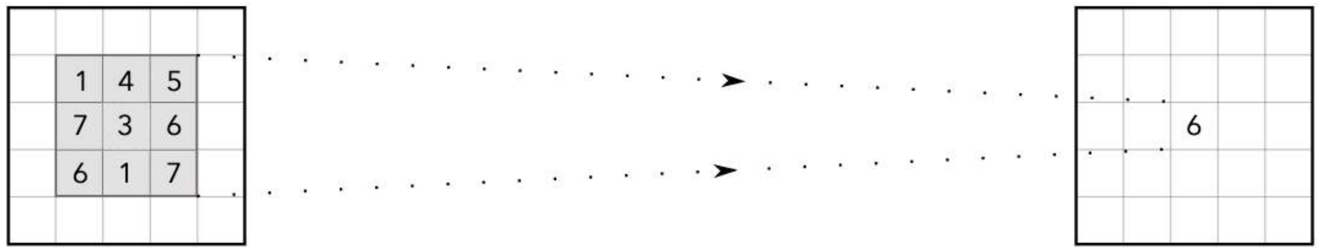
$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

Which assumptions from images are no longer valid?

Number of neighbors per node no longer fixed.

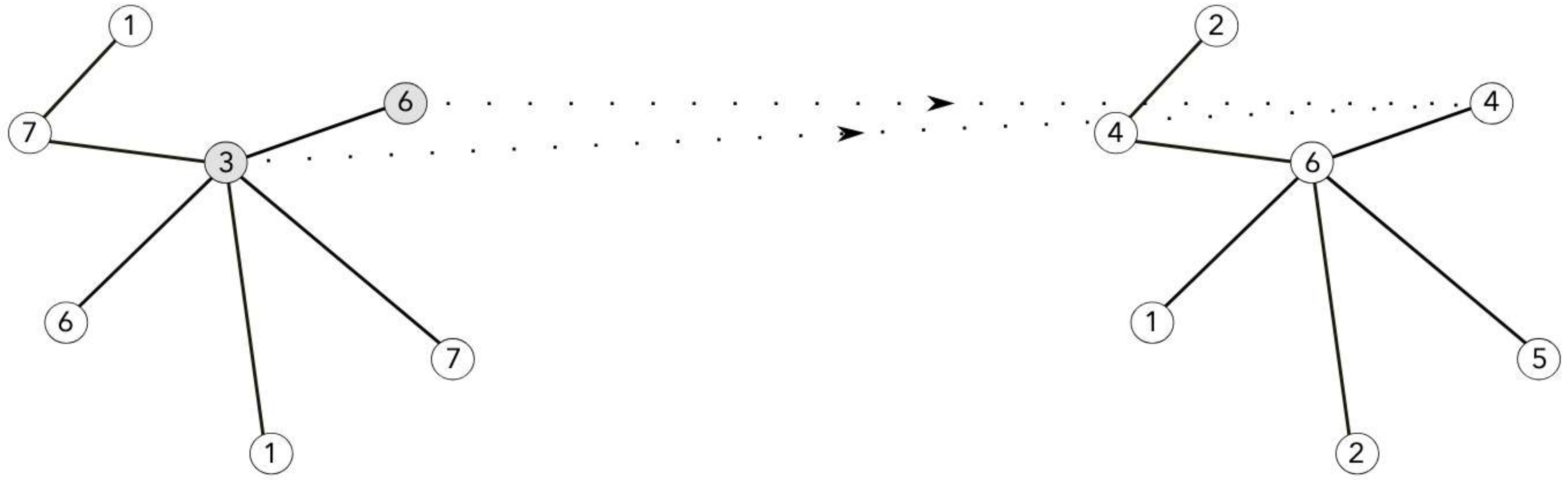
No more ordering between neighbours.

Extending convolutions to graphs



Convolution in CNNs

Convolutions in CNNs are inherently localized. Neighbours participating in the convolution at the center pixel are highlighted in gray.

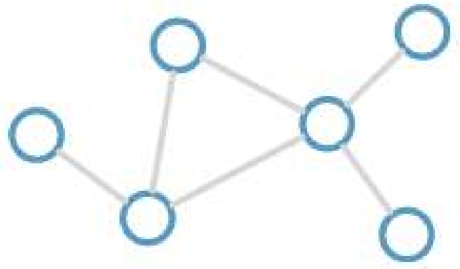


Localized Convolution in GNNs

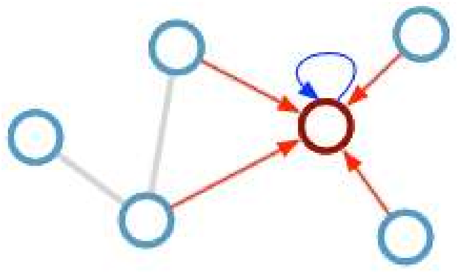
GNNs can perform localized convolutions mimicking CNNs. Hover over a node to see its immediate neighbourhood highlighted on the left. The structure of this neighbourhood changes from node to node.

Graph convolution layer

Consider this undirected graph:



Calculate update for node in red:



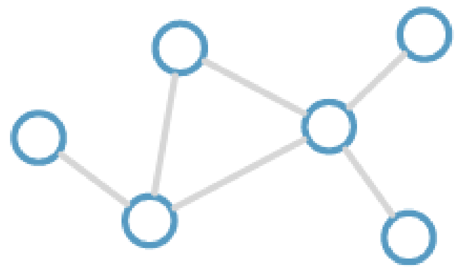
Stacking graph convolution layers

Each layer aggregates information from their direct neighbors.

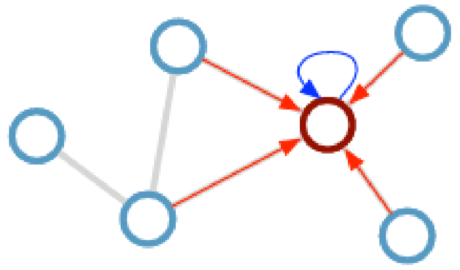
At the end of each layer, we add a non-linearity such as a ReLU.

We can increase complexity and receptive field simply by stacking multiple layers.

Consider this undirected graph:



Calculate update for node in red:



Update rule: $$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

https://mbernste.github.io/posts/gcn/

Graph convolution layer in matrix form

$$f(\boldsymbol{X}, \boldsymbol{A}) := \sigma\left(\boldsymbol{D}^{-1/2}(\boldsymbol{A} + \boldsymbol{I})\boldsymbol{D}^{-1/2}\boldsymbol{X}\boldsymbol{W}\right)$$

$\boldsymbol{A} \in \mathbb{R}^{n \times n}$	:= The adjacency matrix
$\boldsymbol{I} \in \mathbb{R}^{n \times n}$	:= The identity matrix
$\boldsymbol{D} \in \mathbb{R}^{n \times n}$	:= The degree matrix of $\boldsymbol{A} + \boldsymbol{I}$
$\boldsymbol{X} \in \mathbb{R}^{n \times d}$	:= The input data (i.e., the per-node feature vectors)
$\boldsymbol{W} \in \mathbb{R}^{d \times w}$	:= The layer's weights
$\sigma(.)$	:= The activation function (e.g., ReLU)

Kipf and Welling (2017)

Let’s break it down

$$f(\boldsymbol{X},\boldsymbol{A}) \coloneqq \sigma\left(\underbrace{\underbrace{\underbrace{\boldsymbol{D}^{-1/2}\underbrace{(\boldsymbol{A}+\boldsymbol{I})}_{\text{Add self-loops}}\boldsymbol{D}^{-1/2}}_{\text{Normalize adjacency matrix}}\boldsymbol{X}}_{\text{Aggregate}}\boldsymbol{W}}_{\text{Update}}\right)$$

Let’s break it down

Add ones to diagonal, needed because each node should pass its own vector through.

$$f(\boldsymbol{X}, \boldsymbol{A}) \coloneqq \sigma\Big(\underbrace{\underbrace{\underbrace{\boldsymbol{D}^{-1/2}(\underbrace{\boxed{\boldsymbol{A} + \boldsymbol{I}}}_{\text{Add self-loops}})\boldsymbol{D}^{-1/2}}_{\text{Normalize adjacency matrix}}\boldsymbol{X}}_{\text{Aggregate}}\boldsymbol{W}}_{\text{Update}}\Big)$$

Let’s break it down

This step essentially normalizes the adjaceny matrix. I will show how in a few slides.

$$f(\boldsymbol{X}, \boldsymbol{A}) \coloneqq \sigma\Big(\underbrace{\underbrace{\underbrace{\boxed{\boldsymbol{D}^{-1/2}}\underbrace{(\boldsymbol{A}+\boldsymbol{I})}_{\text{Add self-loops}}\boxed{\boldsymbol{D}^{-1/2}}}_{\text{Normalize adjacency matrix}}\boldsymbol{X}}_{\text{Aggregate}}\boldsymbol{W}}_{\text{Update}}\Big)$$

Let’s break it down

Just a standard linear layer and a non-linearity.

$$f(\boldsymbol{X},\boldsymbol{A}) \coloneqq \boxed{\sigma}\Big(\underbrace{\underbrace{\underbrace{\boldsymbol{D}^{-1/2}(\underbrace{\boldsymbol{A}+\boldsymbol{I}}_{\text{Add self-loops}})\boldsymbol{D}^{-1/2}}_{\text{Normalize adjacency matrix}}\boldsymbol{X}}_{\text{Aggregate}}\boldsymbol{W}}_{\text{Update}}\Big)$$

Let’s break it down

Just a standard linear layer and a non-linearity.

$$f(\boldsymbol{X},\boldsymbol{A}) \coloneqq \sigma\left(\boxed{\underbrace{\underbrace{\underbrace{\boldsymbol{D}^{-1/2}\underbrace{(\boldsymbol{A}+\boldsymbol{I})}_{\text{Add self-loops}}\boldsymbol{D}^{-1/2}}_{\text{Normalize adjacency matrix}}\boldsymbol{X}}_{\text{Aggregate}}\boldsymbol{W}}_{\text{Update}}}\right)$$

Break

Let’s break it down

Just a standard linear layer and a non-linearity.

$$f(\boldsymbol{X}, \boldsymbol{A}) \coloneqq \sigma\left(\boxed{\underbrace{\underbrace{\underbrace{\boldsymbol{D}^{-1/2}\underbrace{(\boldsymbol{A}+\boldsymbol{I})}_{\text{Add self-loops}}\boldsymbol{D}^{-1/2}}_{\text{Normalize adjacency matrix}}\boldsymbol{X}}_{\text{Aggregate}}\boldsymbol{W}}_{\text{Update}}}\right)$$

Rewriting into 2 steps

$$\tilde{A} := D^{-1/2}(A + I)D^{-1/2}$$

$$D := \begin{bmatrix} d_{1,1} & 0 & 0 & \dots & 0 \\ 0 & d_{2,2} & 0 & \dots & 0 \\ 0 & 0 & d_{3,3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_{n,n} \end{bmatrix}$$

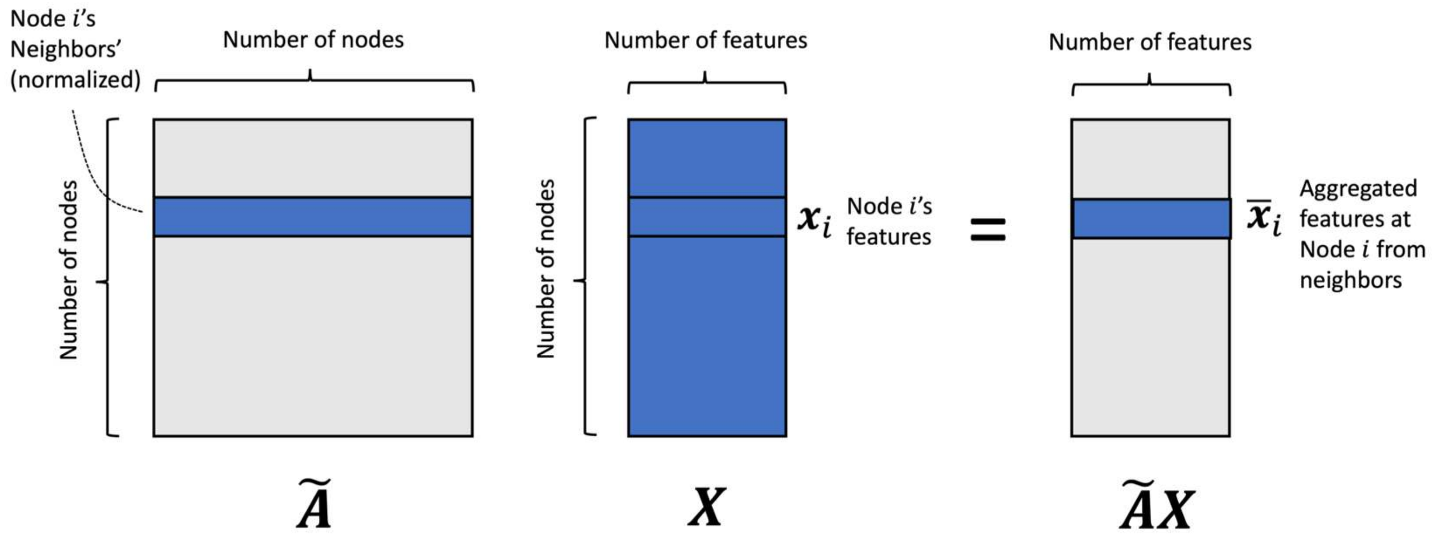
$$\tilde{A}_{i,j} := \begin{cases} \frac{1}{\sqrt{d_{i,i}d_{j,j}}}, & \text{if there is an edge between node } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$$

$$f(X, A) := \sigma\left(\tilde{A}XW\right)$$

$$D^{-1/2} := \begin{bmatrix} \frac{1}{\sqrt{d_{1,1}}} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{d_{2,2}}} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\sqrt{d_{3,3}}} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{\sqrt{d_{n,n}}} \end{bmatrix}$$

$$f(X, A) := \sigma\left(\boxed{\tilde{A}X}W\right)$$

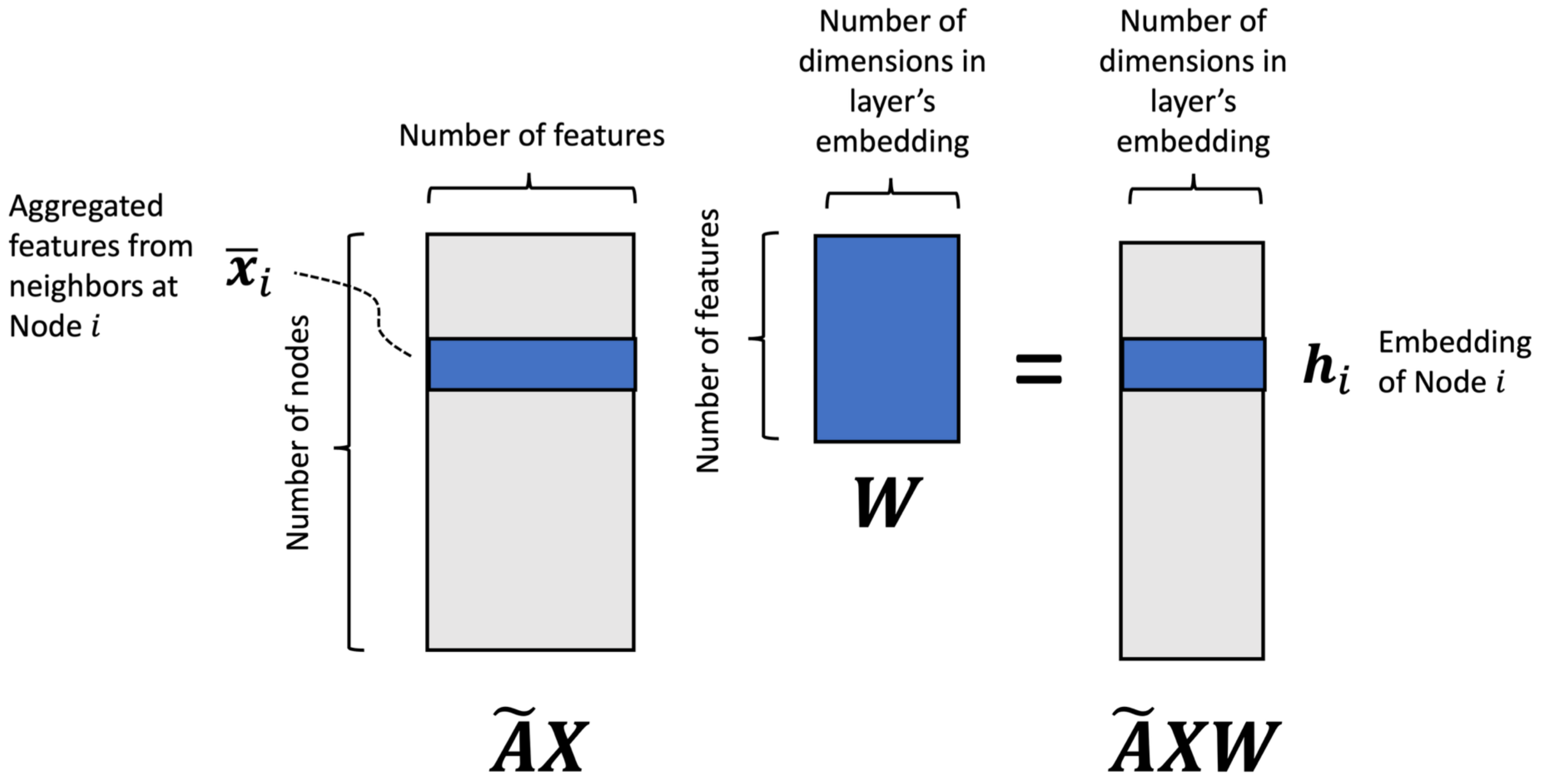
Left side of the equation


Node i's Neighbors' (normalized)
Number of nodes
Number of nodes
Number of features
Number of nodes
$\boldsymbol{x}_i$ Node i's features
=
Number of features
$\bar{\boldsymbol{x}}_i$ Aggregated features at Node i from neighbors
$\tilde{\boldsymbol{A}}$
$\boldsymbol{X}$
$\tilde{\boldsymbol{A}}\boldsymbol{X}$

$$\begin{aligned}\bar{\boldsymbol{x}}_i &= \sum_{j=1}^{n} \tilde{a}_{i,j}\boldsymbol{x}_j \\ &= \sum_{j \in \mathrm{Neigh}(i)} \tilde{a}_{i,j}\boldsymbol{x}_j \\ &= \sum_{j \in \mathrm{Neigh}(i)} \frac{1}{\sqrt{d_{i,i}d_{j,j}}}\boldsymbol{x}_j\end{aligned}$$

$$f(X, A) := \sigma\left(\tilde{A}\boxed{XW}\right)$$

Right side of the equation



Number of features
Aggregated features from neighbors at Node i
$\overline{x}_i$
Number of nodes
$\tilde{A}X$
Number of dimensions in layer's embedding
Number of features
W
=
Number of dimensions in layer's embedding
h_i
Embedding of Node i
$\tilde{A}XW$

Why add a normalization step?

Do we even need it? Let’s see what happens without it:

$$\hat{\boldsymbol{A}} := \boldsymbol{A} + \boldsymbol{I}$$

Normalization dropped, only self-loop retrained

$$f_{\text{unnormalized}}(\boldsymbol{X}, \boldsymbol{A}) := \sigma(\hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W})$$

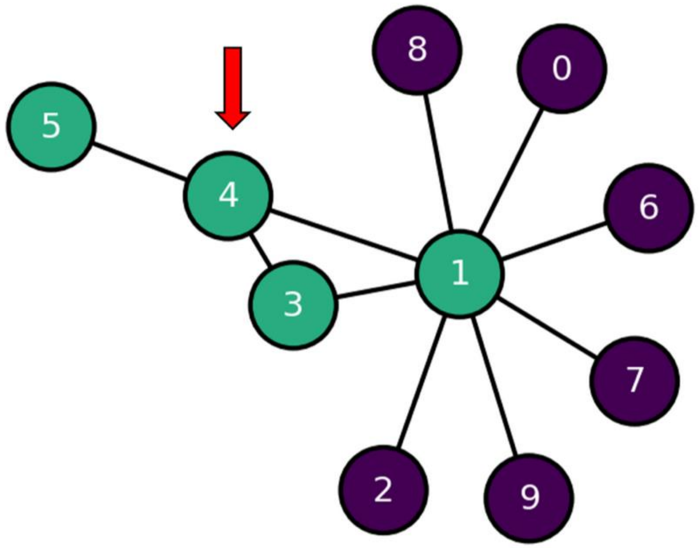
Layer update remains the same

$$\begin{aligned}\bar{\boldsymbol{x}}_i &= \sum_{j=1}^{n} \hat{a}_{i,j}\boldsymbol{x}_j \\ &= \sum_{j=1}^{n} \mathbb{I}(j \in \text{Neigh}(i))\boldsymbol{x}_j \\ &= \sum_{j \in \text{Neigh}(i)} \boldsymbol{x}_j\end{aligned}$$

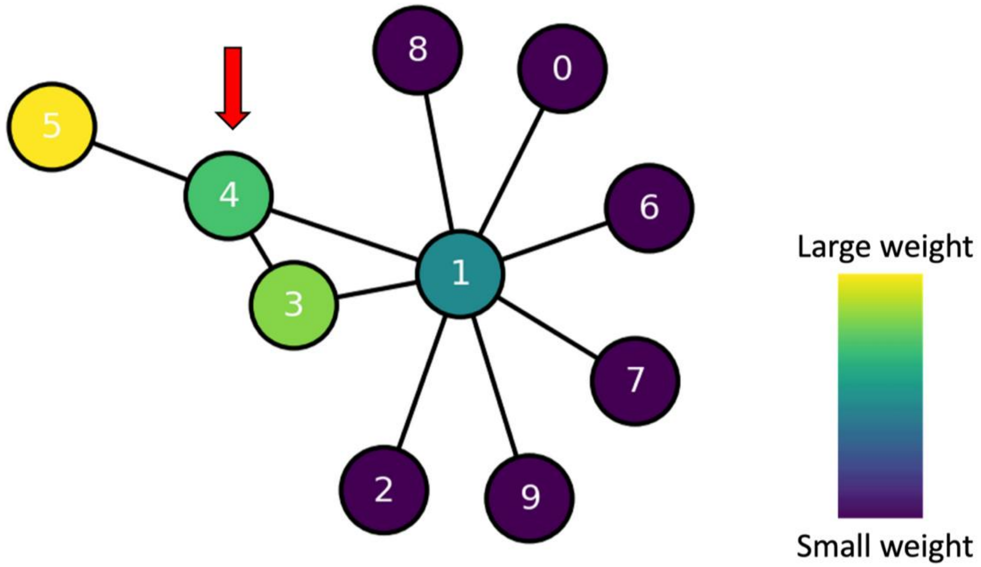
Problem! More neighbors = bigger sum.
Huge bias when training graph networks.

Why not simply divide by the node degree?

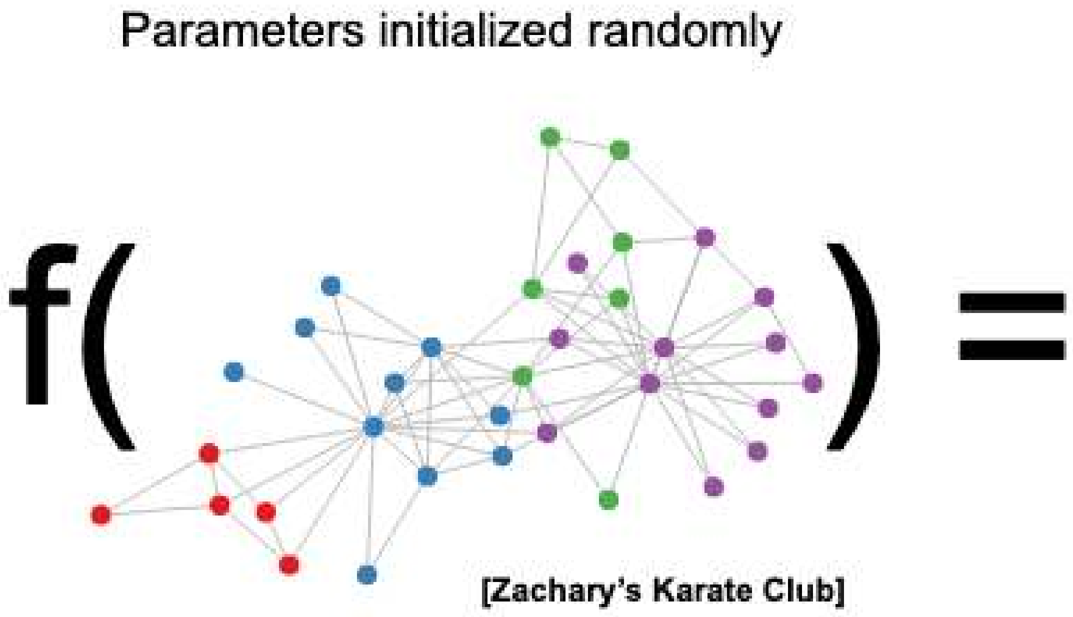
$D^{-1}\widehat{A}$

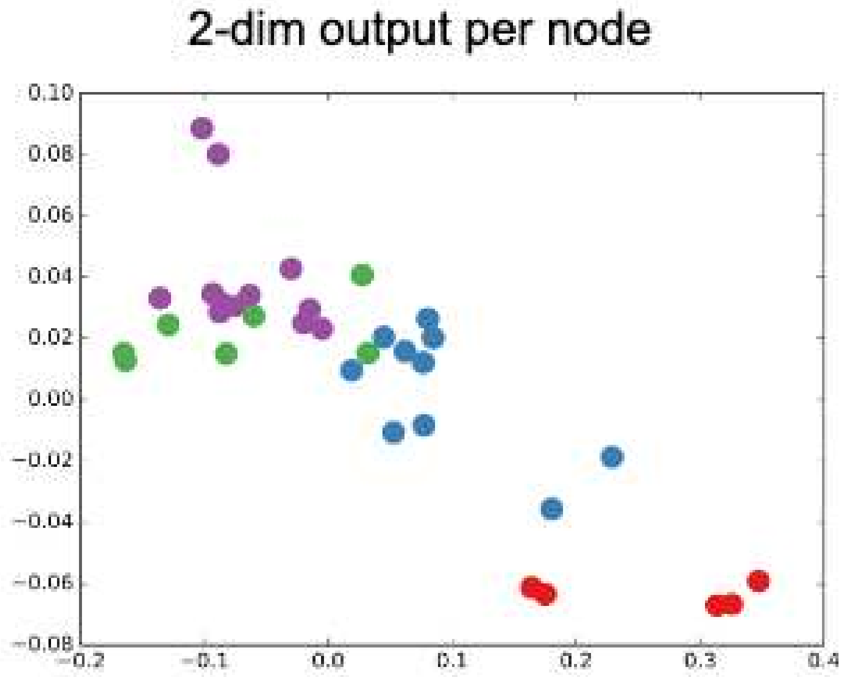


$D^{-1/2}\widehat{A}D^{-1/2}$



Visualizing node representations





Alternative: graph layer as attention

Similar but including attention as *aggregation*: $y_i = h\left(\sum_{j \in \mathcal{N}(i)} a_{ij} \boldsymbol{z_j}\right)$

Using self-attention:

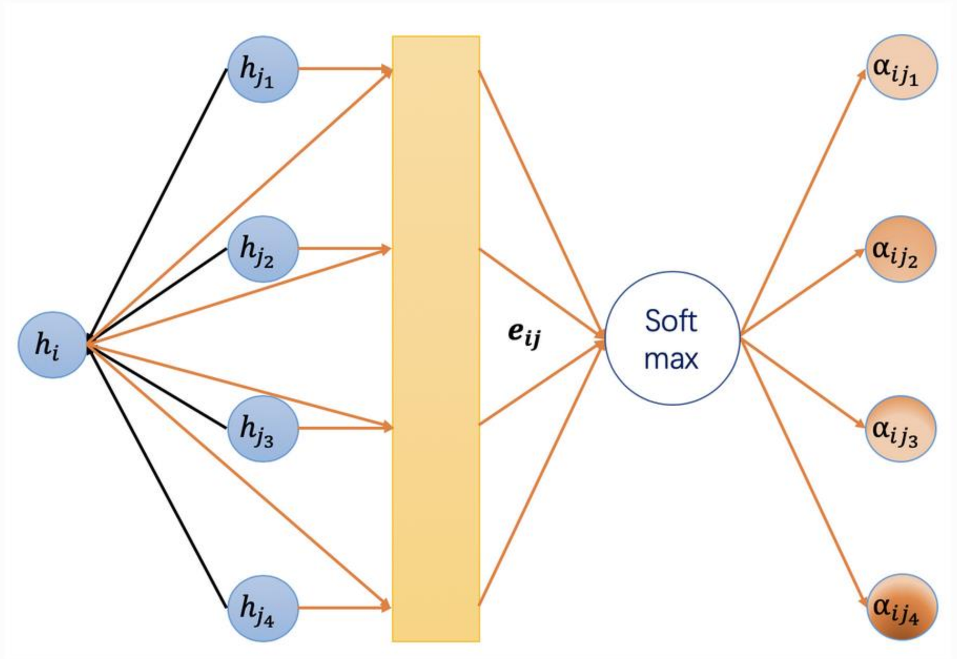
$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ij})},$$

where e_{ij} are the self-attention weights (like query == key)

$$e_{ij} = \mathrm{LeakyRELU}\left(\left[\boldsymbol{x}_i \boldsymbol{W}, \boldsymbol{x}_j \boldsymbol{W}\right] \cdot u\right)$$

u is a weight vector.

The four steps of a graph attention layer



$$z_i^{(l)} = W^{(l)} h_i^{(l)}, \tag{1}$$

$$e_{ij}^{(l)} = \text{LeakyReLU}(\vec{a}^{(l)^T}(z_i^{(l)} || z_j^{(l)})), \tag{2}$$

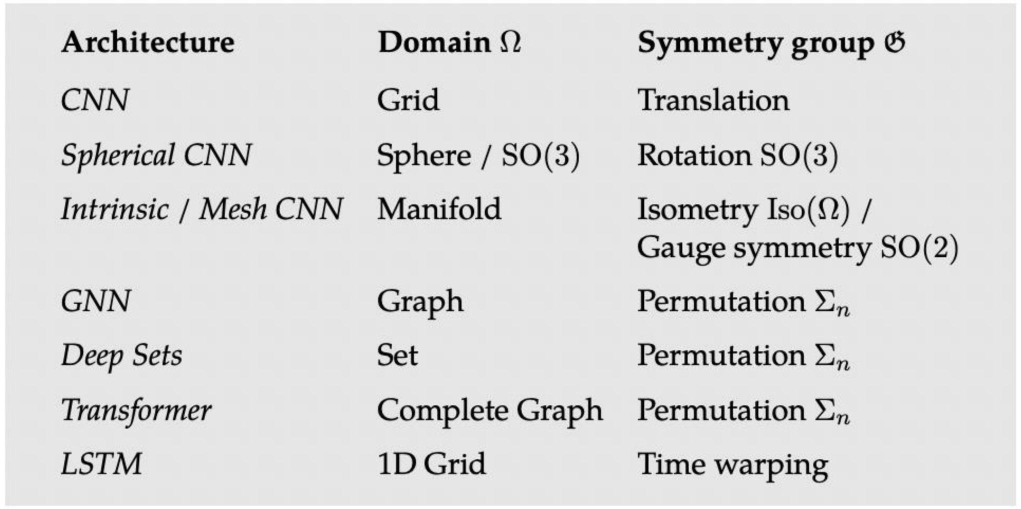
$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik}^{(l)})}, \tag{3}$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)}\right), \tag{4}$$

Connecting graphs, convolutions, and transformers

Transformers operate on a complete graph (adjacency matrix with all 1’s).

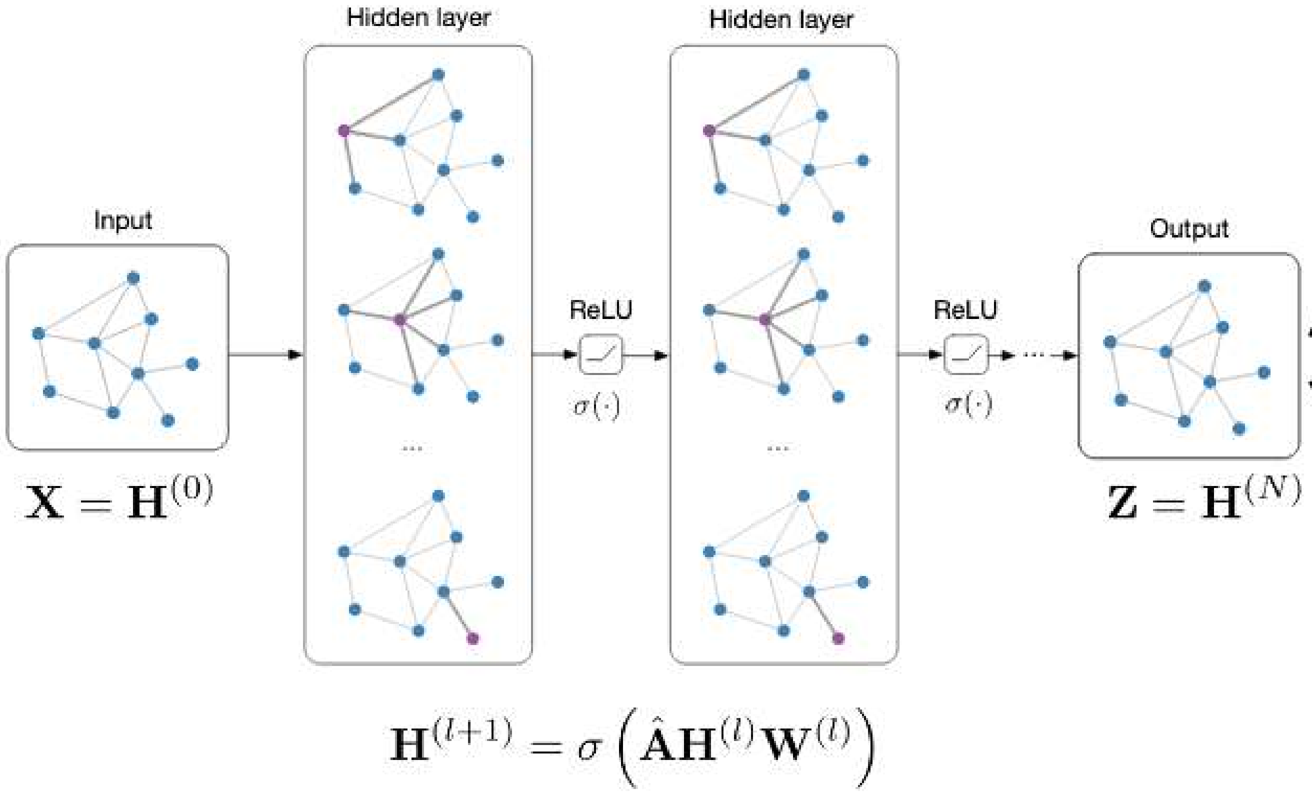
With attention-based GCN, we recover the Transformer.



Architecture	**Domain Ω**	**Symmetry group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere / SO(3)	Rotation SO(3)
Intrinsic / Mesh CNN	Manifold	Isometry Iso(Ω) / Gauge symmetry SO(2)
GNN	Graph	Permutation Σ_n
Deep Sets	Set	Permutation Σ_n
Transformer	Complete Graph	Permutation Σ_n
LSTM	1D Grid	Time warping

Optimizing graph networks

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



Input
Hidden layer
Hidden layer
ReLU
ReLU
$\sigma(\cdot)$
$\sigma(\cdot)$
Output
$\mathbf{X} = \mathbf{H}^{(0)}$
$\mathbf{Z} = \mathbf{H}^{(N)}$
$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$

Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

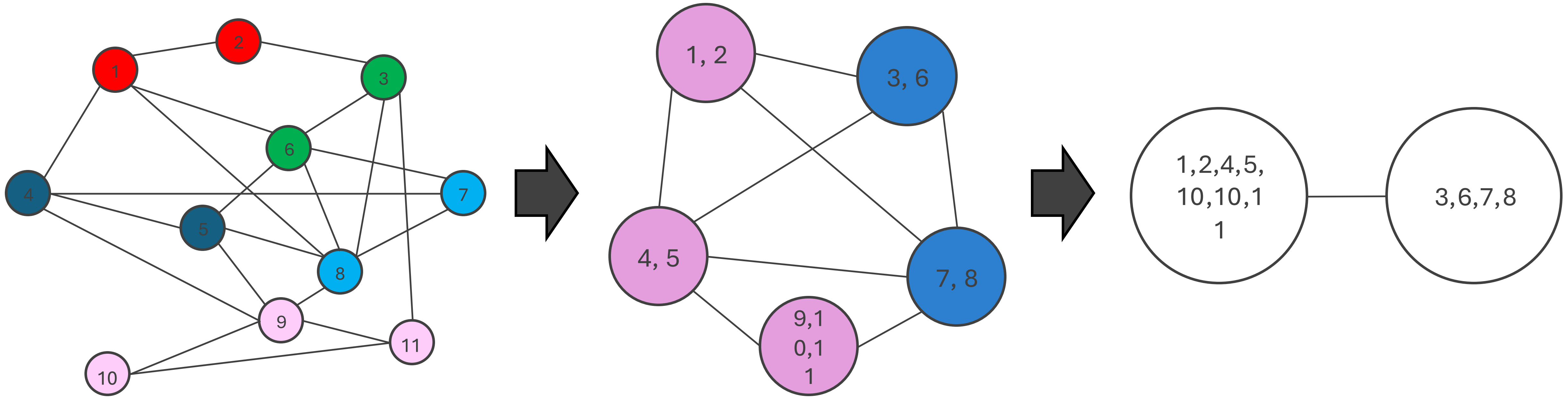
$p(A_{ij}) = \sigma(\mathbf{z_i}^\mathbf{T}\mathbf{z_j})$

Kipf & Welling (NIPS BDL 2016)
"Graph Auto-Encoders"

Pooling in graph networks

Specifically for graph classification, pooling is an optional operators.

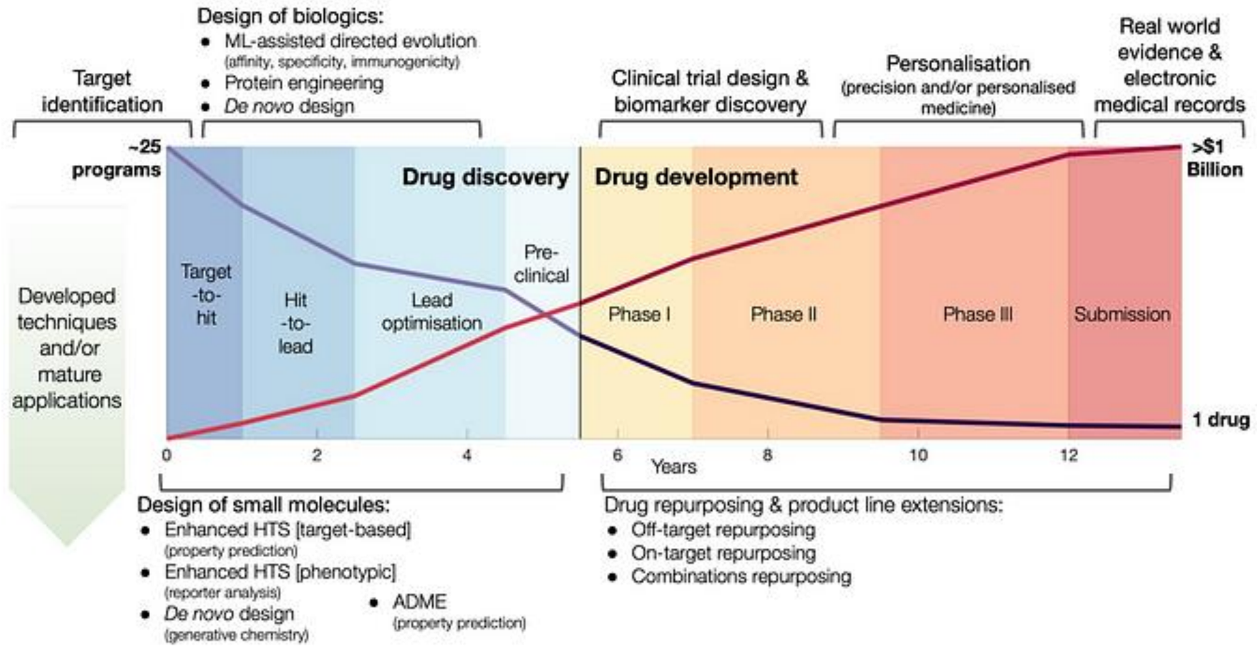
Pool nodes together to save compute, requires updating the adjacency matrix.

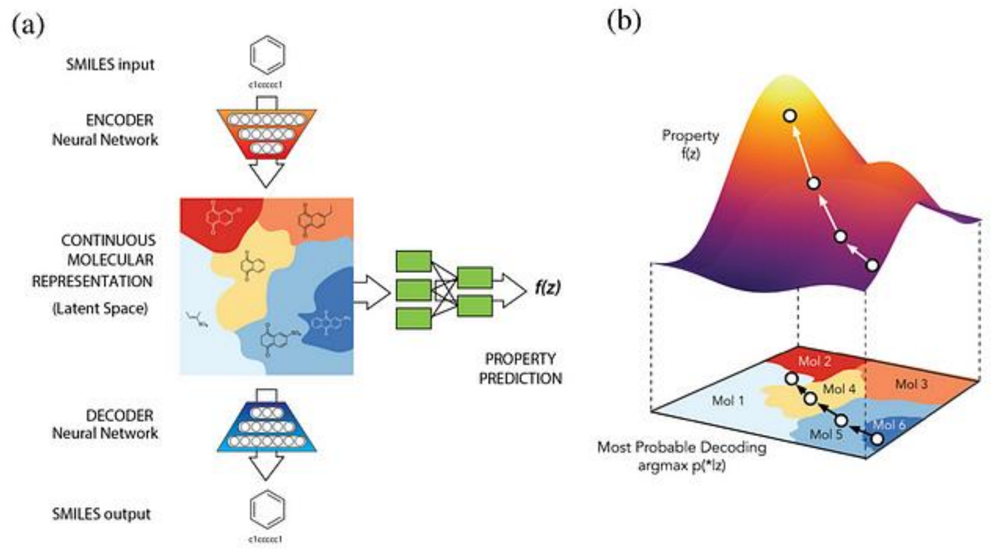


https://jonathan-hui.medium.com/applications-of-graph-neural-networks-gnn-d487fd5ed17d

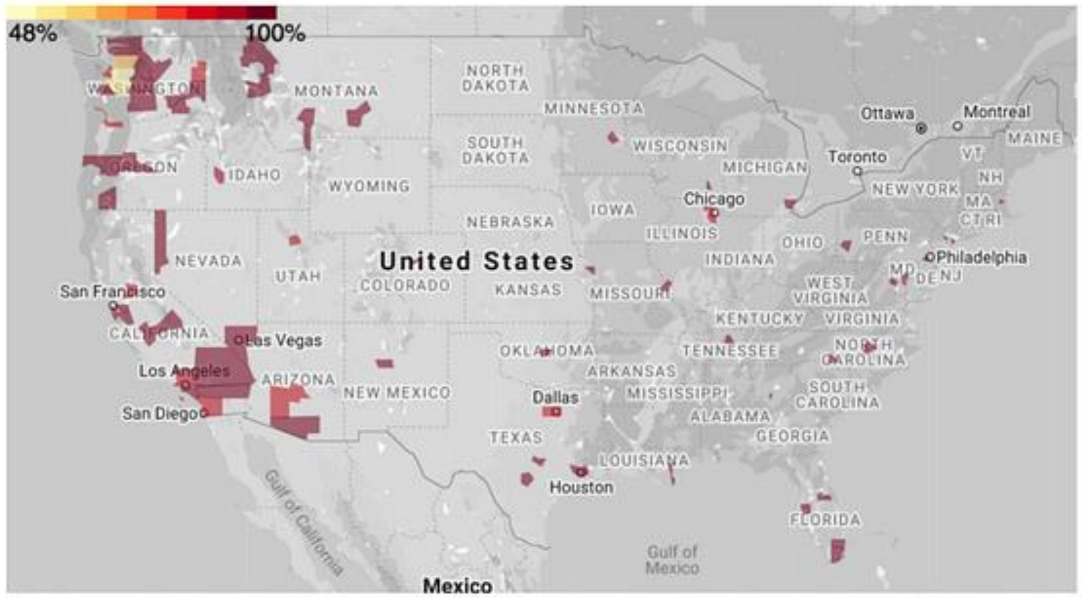
Applications of graph networks

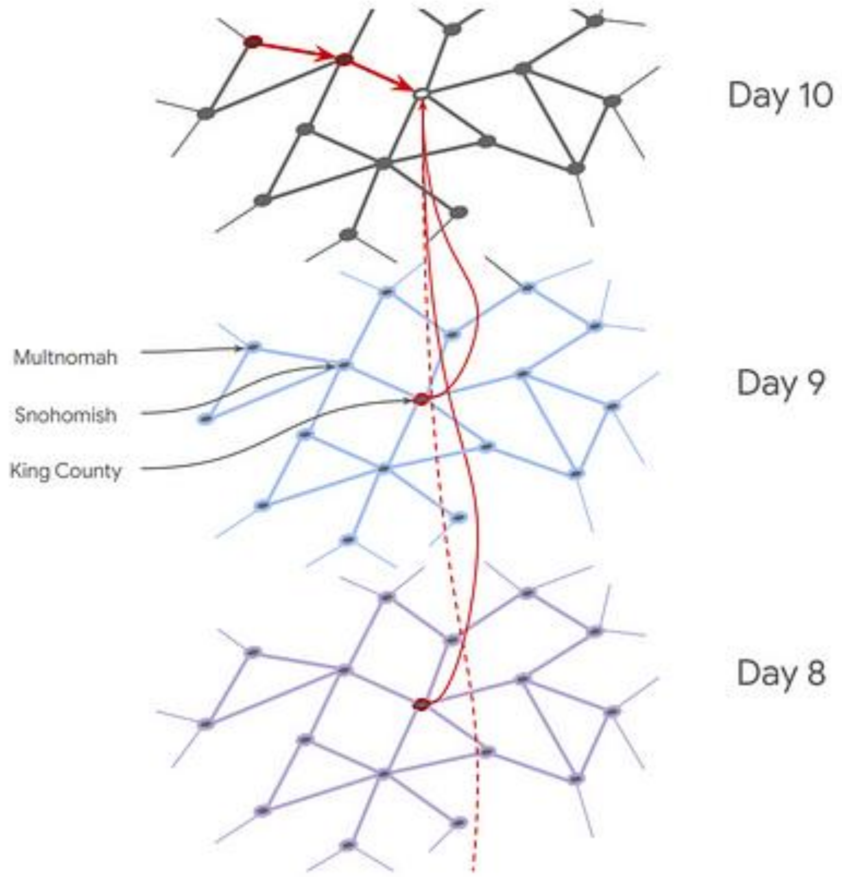
Drug discovery



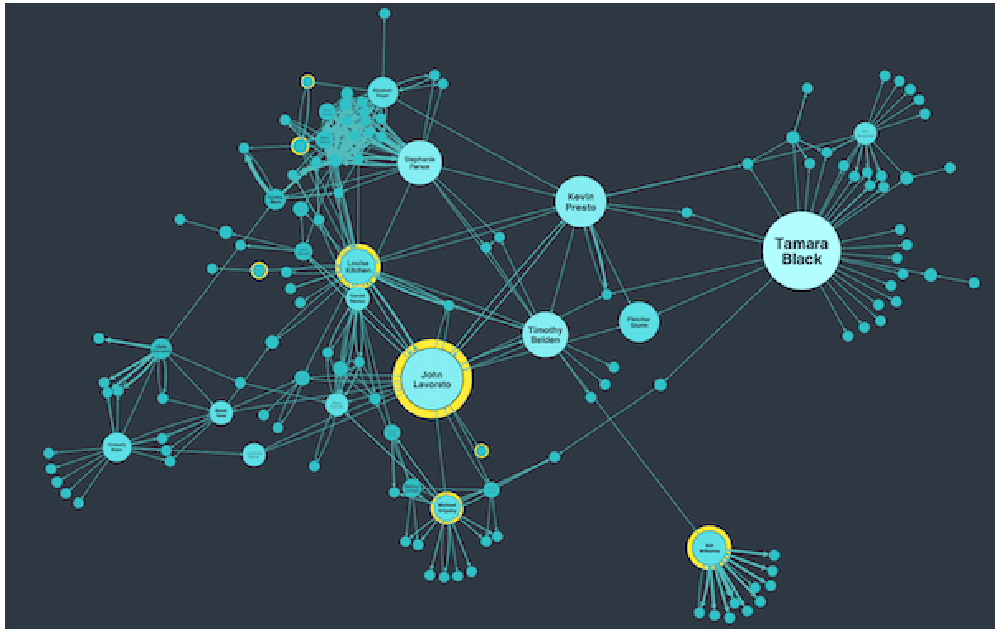


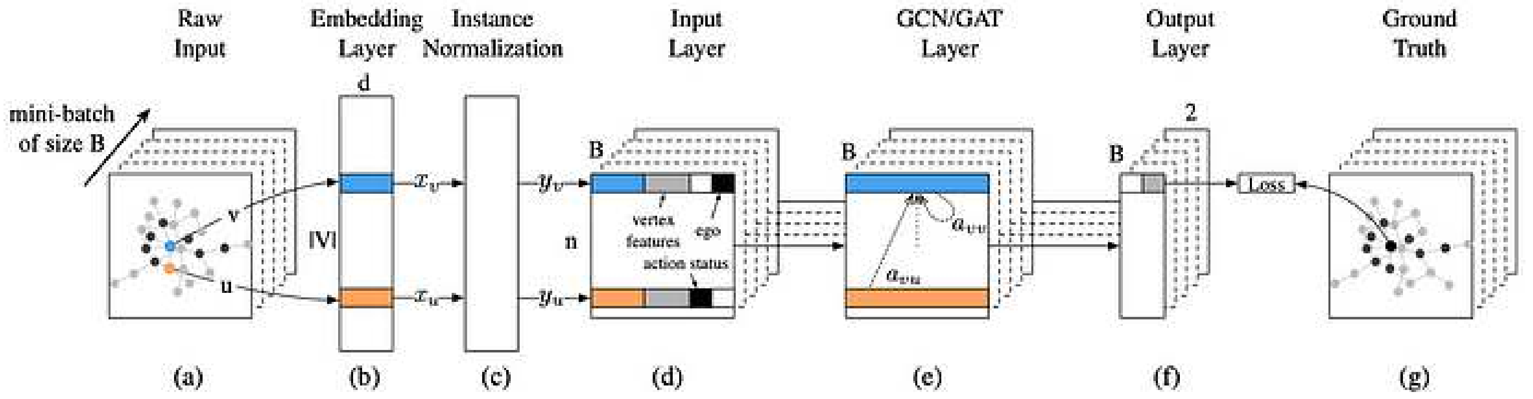
Modeling the spread of deceases



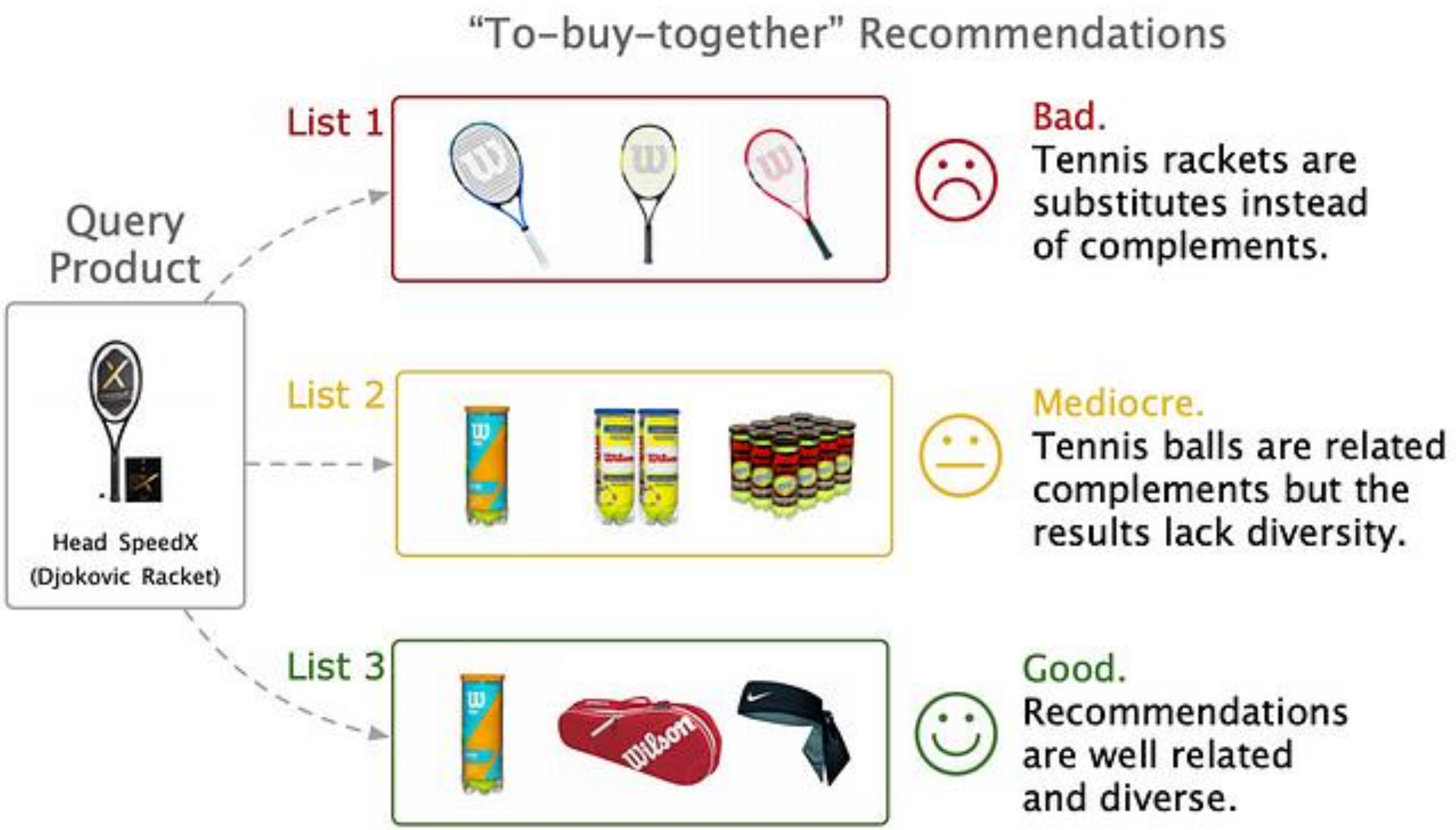


Social networks

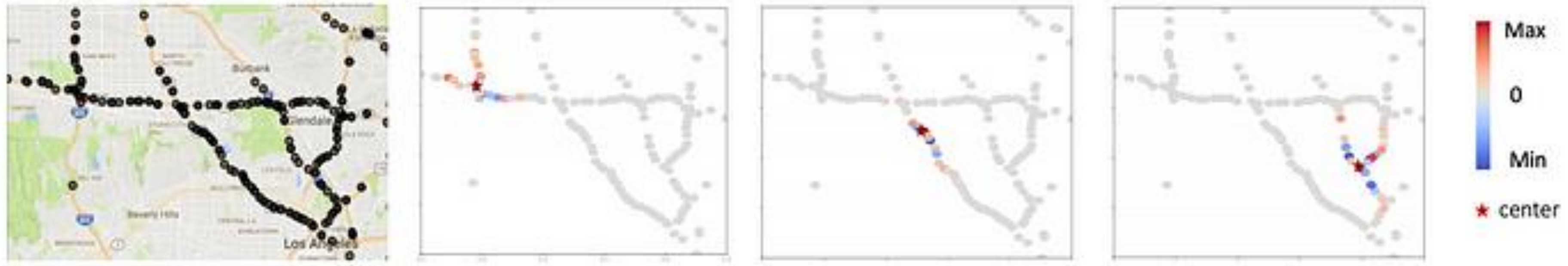


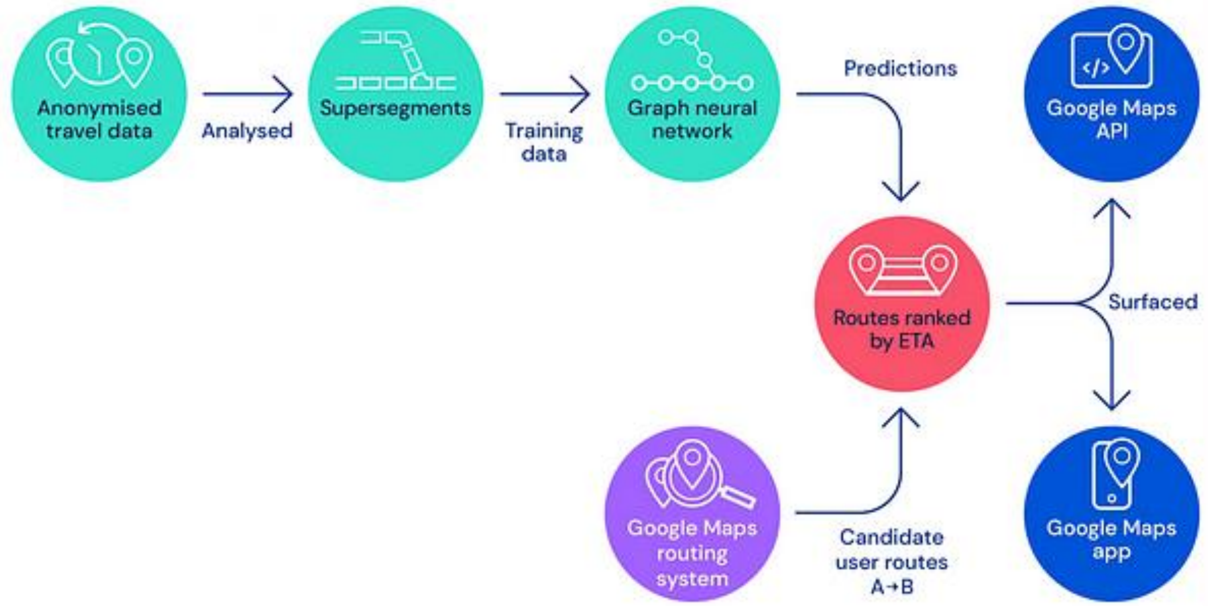


Recommendation


"To-buy-together" Recommendations
Query Product
Head SpeedX (Djokovic Racket)
List 1
List 2
List 3
Bad.
Tennis rackets are substitutes instead of complements.
Mediocre.
Tennis balls are related complements but the results lack diversity.
Good.
Recommendations are well related and diverse.

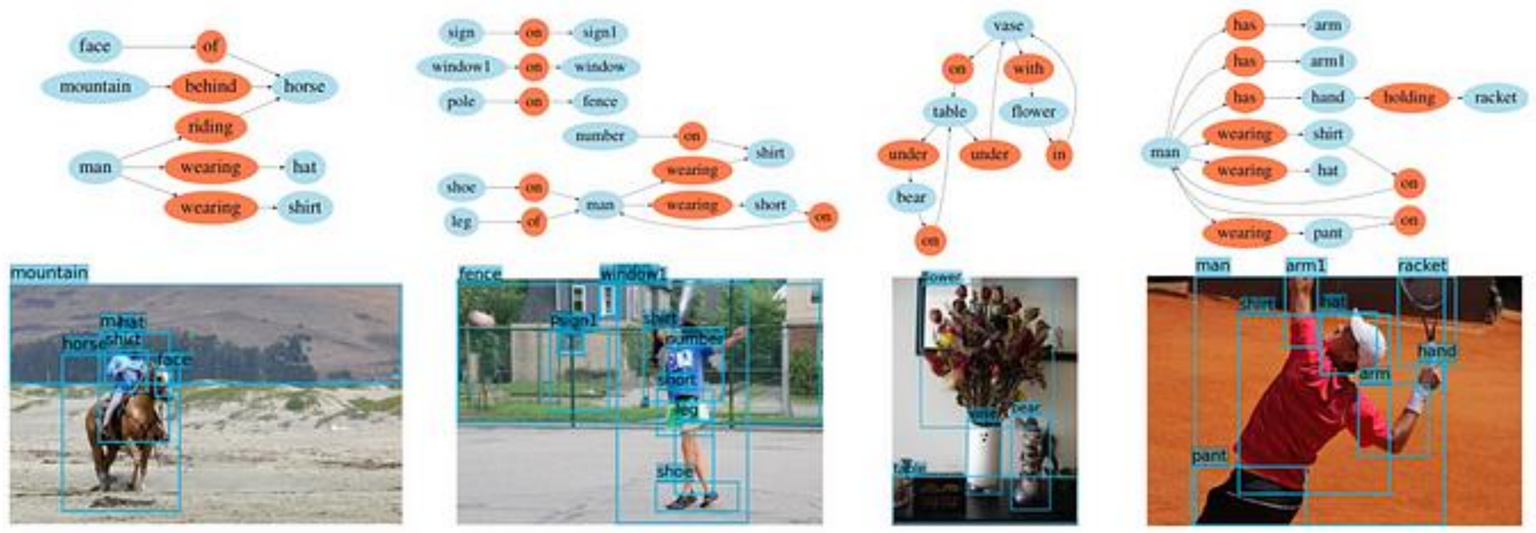
Traffic forecasting

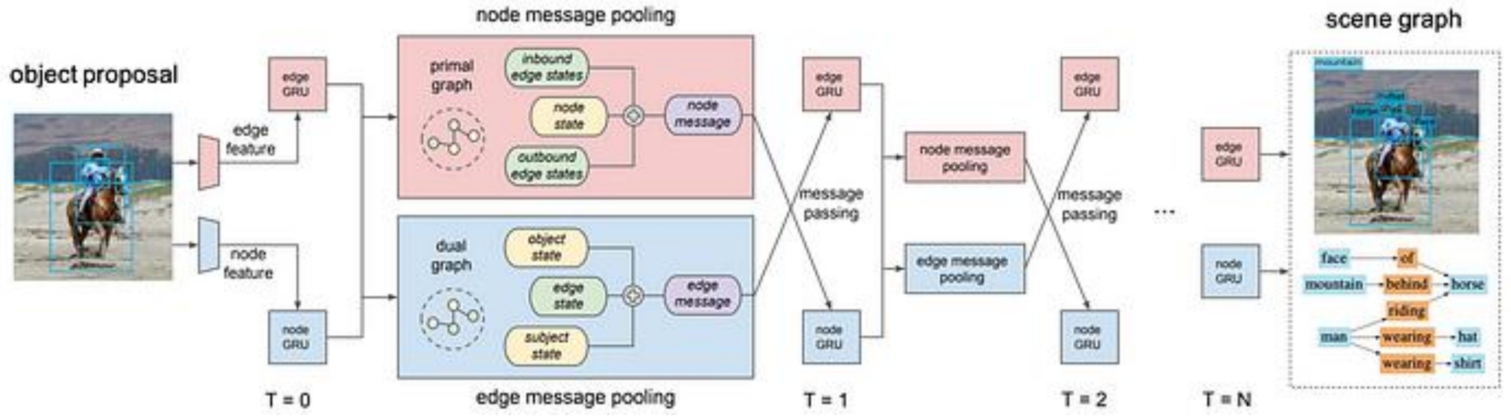




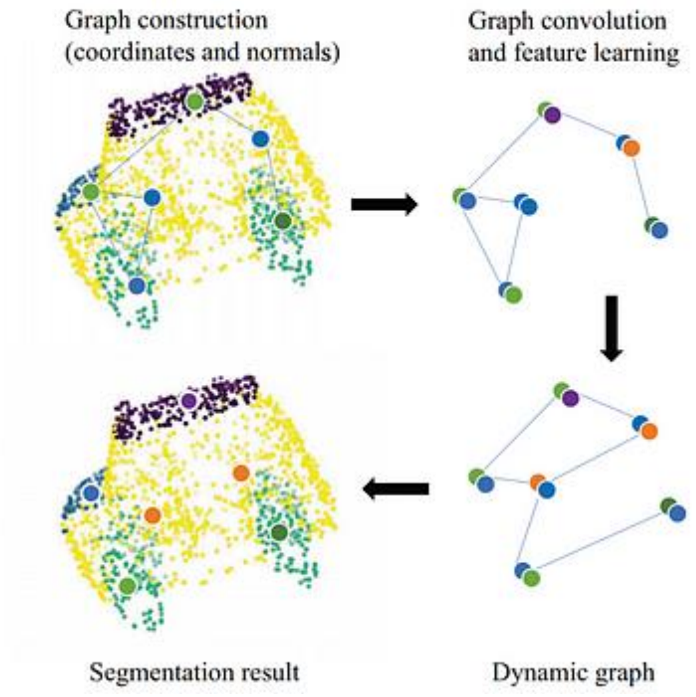
The model architecture for determining optimal routes and their travel time.

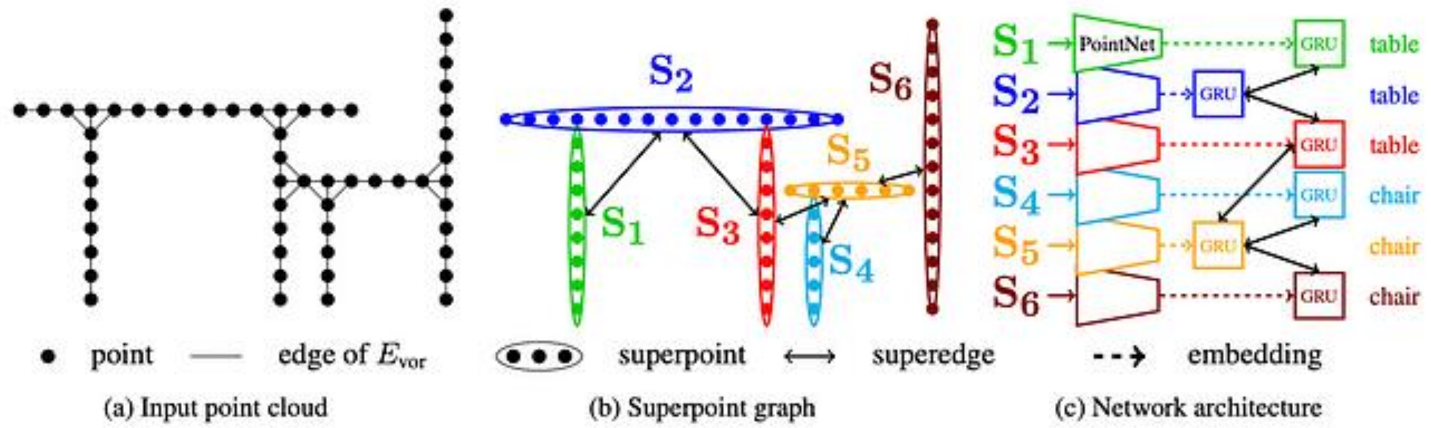
Scene graph generation of visual data



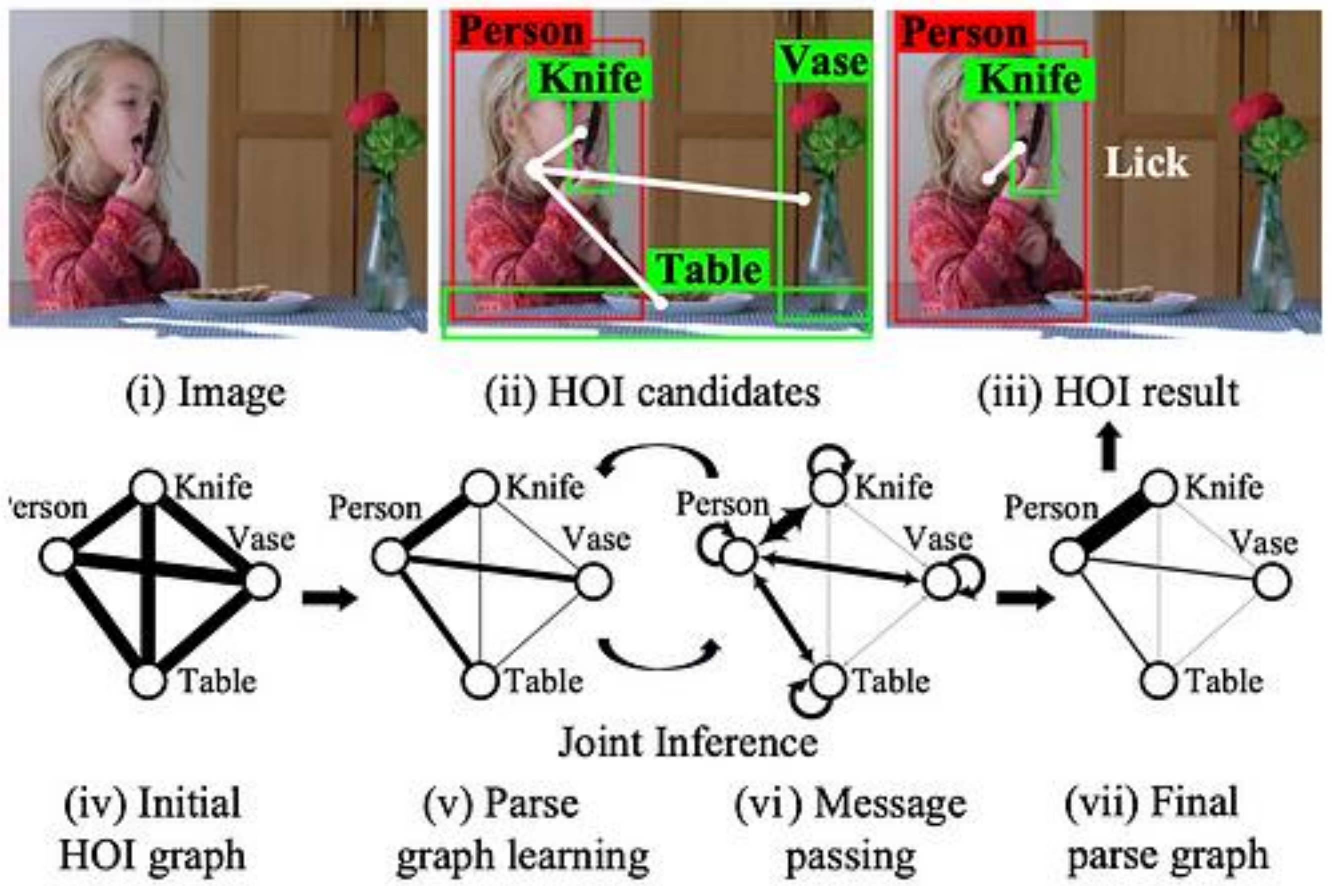


Point cloud classification

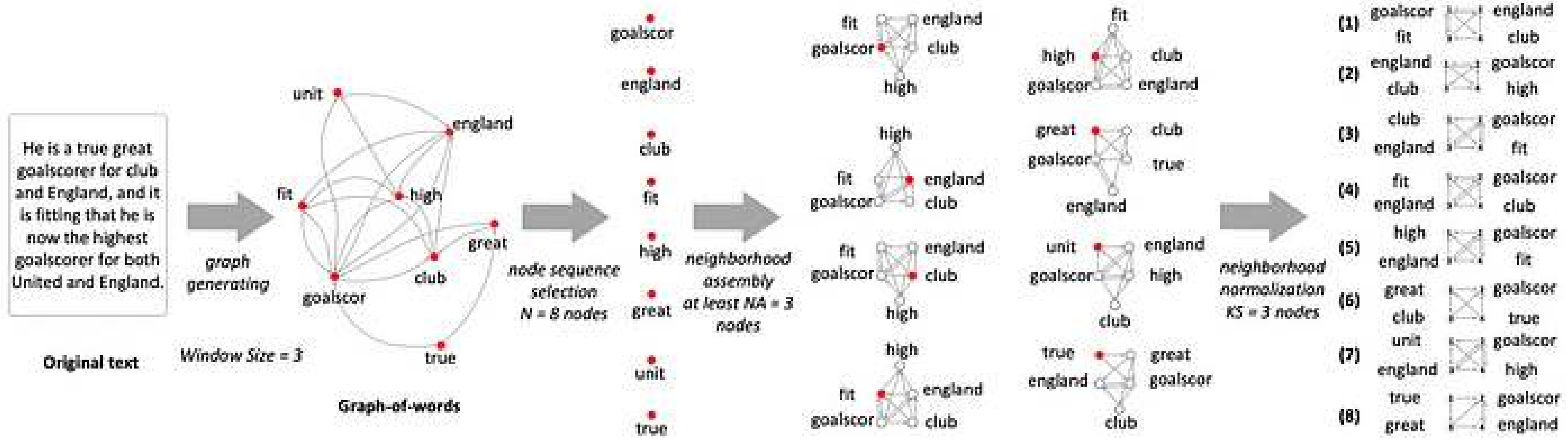

Graph construction (coordinates and normals)
Graph convolution and feature learning
Dynamic graph
Segmentation result



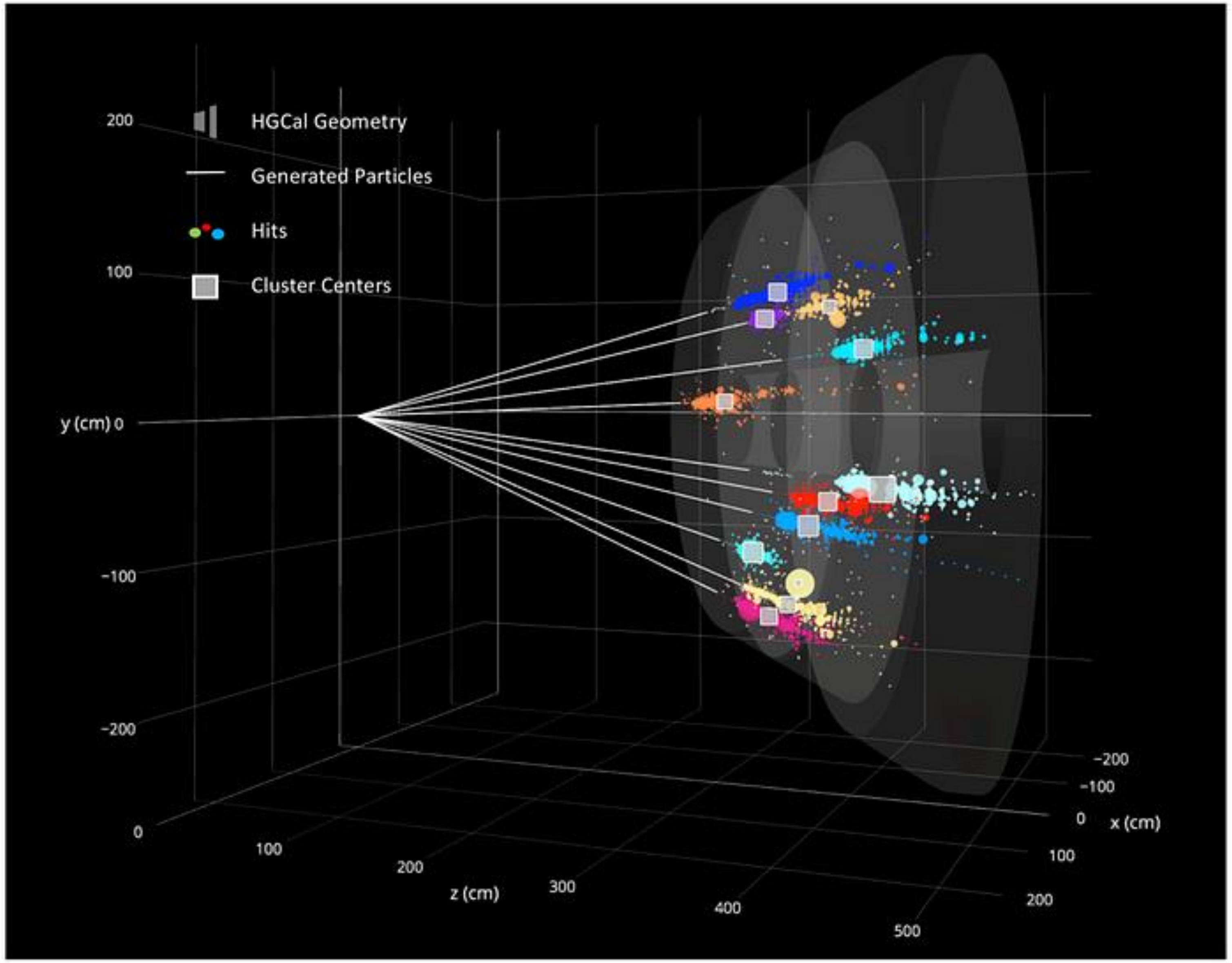
Object interactions



Text classification


He is a true great goalscorer for club and England, and it is fitting that he is now the highest goalscorer for both United and England.
Original text
graph generating
Window Size = 3
Graph-of-words
unit
england
fit
high
great
club
goalscor
true
node sequence selection N = 8 nodes
neighborhood assembly at least NA = 3 nodes
neighborhood normalization KS = 3 nodes
(1) goalscor fit england club
(2) england club goalscor high
(3) club england goalscor fit
(4) fit england goalscor club
(5) high england goalscor fit
(6) great club goalscor true
(7) unit england goalscor high
(8) true great goalscor england

Particle physics



Next lecture

Lecture	Title	Lecture	Title
1	Intro and history of deep learning	2	Manually forward, automatically backward
3	Deep learning optimization I	4	Deep learning optimization II
5	Convolutional Neural Networks I	6	Convolutional Neural Networks II
7	Attention	8	Graph Neural Networks
9	Self-supervised and vision-language learning	10	Auto-encoding and generation
11	The oddities of deep learning	12	Non-Euclidean deep learning
13	Deep learning for videos	14	Q&A

Thank you!