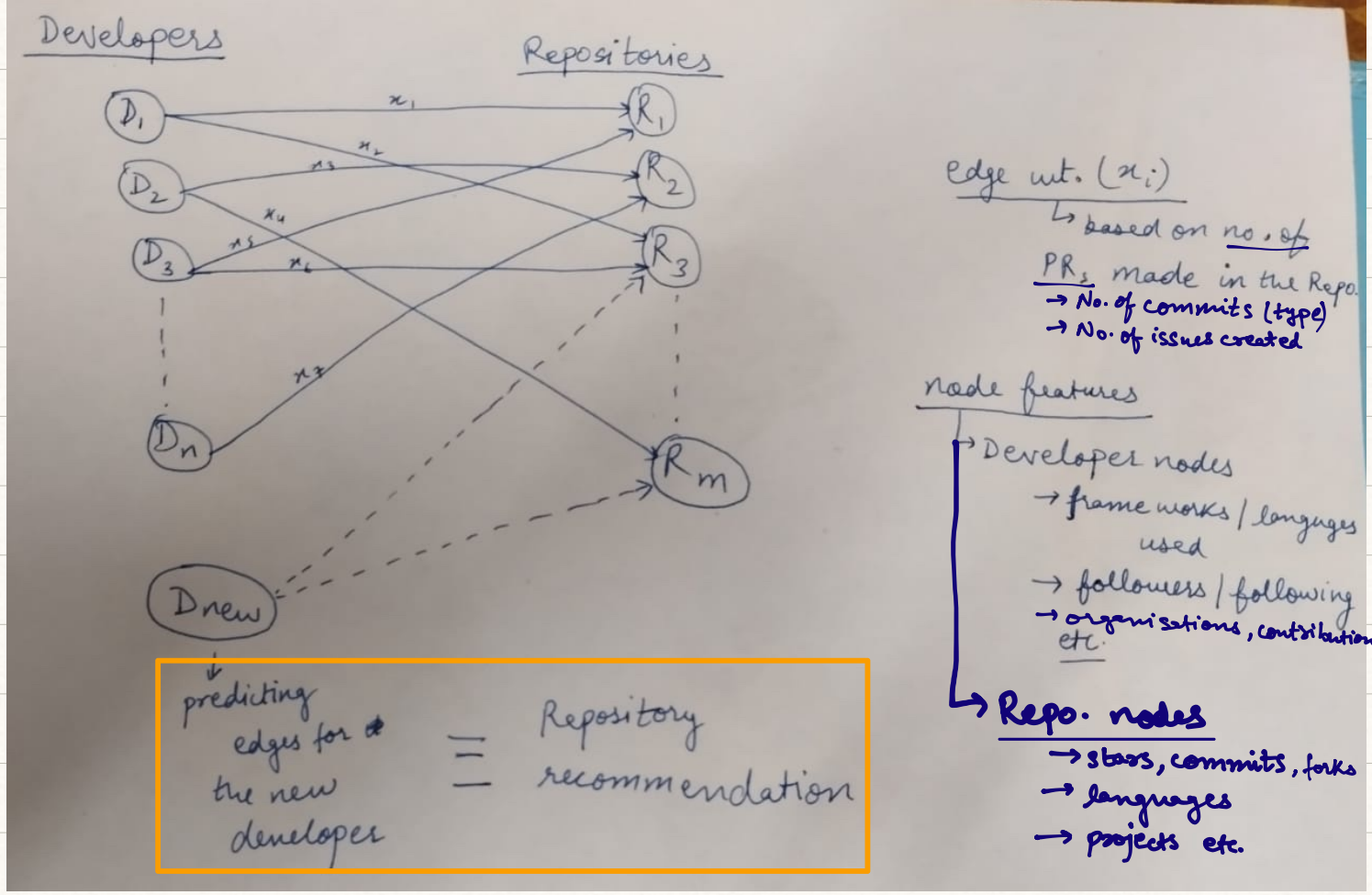
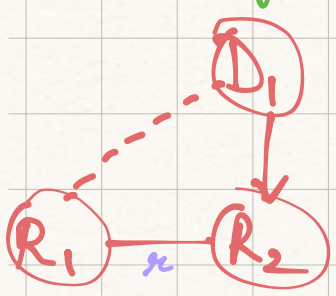


- developer features : organisations, starred repos, followers, following, num_of_contributions(in last year), languages used, Readme.md, achievements
- Repository features : fork count, star count, commit count, open PRs, readme, license, issue count, tags, languages used, num_of_contributors

Approaches

content-based filtering

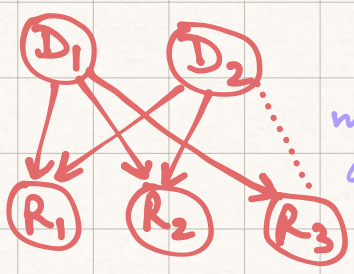


Define Similarity measure for Repositories

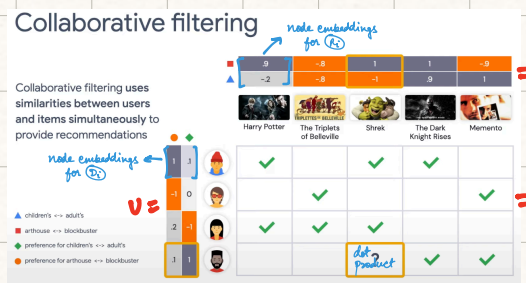
Cosine similarity or correlation

$$sim(\mathbf{u},\mathbf{v}) = \frac{\sum_{i \in C}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in C}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i \in C}(r_{v,i} - \bar{r}_v)^2}}$$

Collaborative filtering



Similarity measure for developers and Repositories or matrix factorisation



$A \approx U \times V^T$

Collaborative filtering using Matrix Factorization

$A \approx U \times V^T$

$\min_{U,V} \Sigma_{(i,j)} (A_{ij} - U_i \cdot V_j)^2$

Solve this using:

- Stochastic Gradient Descent (SGD)
- Weighted Alternating Least Squares (WALS)

WALS converge faster than SGD but SGD can handle different loss functions

Observed only MF

1		1	1	
	1			1
1	1	1		
			1	1

$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2$

SVD

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

$|A - UV^T|_F{}^2$

$= \Sigma_{(i,j)} (A_{ij} - U_i \cdot V_j)^2$

bad generalisation (A is sparse matrix)

Weighted MF

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

$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2 + W_0 \Sigma_{(i,j) \notin obs} (0 - U_i \cdot V_j)^2$

hyper parameter

Neighborhood formation phase

- Let the record (or profile) of the target user be **u** (represented as a vector), and the record of another user be **v** ($\mathbf{v} \in T$).
- The similarity between the target user, **u**, and a neighbor, **v**, can be calculated using the **Pearson's correlation coefficient**:

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i \in I_{\mathbf{u}} \cap I_{\mathbf{v}}} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sqrt{\sum_{i \in I_{\mathbf{u}} \cap I_{\mathbf{v}}} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2} \sqrt{\sum_{i \in I_{\mathbf{u}} \cap I_{\mathbf{v}}} (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})^2}}$$

KNN

Recommendation Phase

- Use the following formula to compute the rating prediction of item i for target user **u**

$$p(\mathbf{u}, i) = \bar{r}_{\mathbf{u}} + \frac{\sum_{\mathbf{v} \in V} sim(\mathbf{u}, \mathbf{v}) \times (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sum_{\mathbf{v} \in V} |sim(\mathbf{u}, \mathbf{v})|}$$

where V is the set of k similar users, $r_{\mathbf{v},i}$ is the rating of user **v** given to item i,

Item-based CF

- The item-based approach works by comparing items based on their pattern of ratings across users. The similarity of items i and j is computed as follows:

$$sim(i, j) = \frac{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},i} - \bar{r}_i)(r_{\mathbf{u},j} - \bar{r}_j)}{\sqrt{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},i} - \bar{r}_i)^2} \sqrt{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},j} - \bar{r}_j)^2}}$$

Recommendation phase

- After computing the similarity between items we select a set of k most similar items to the target item and generate a predicted value of user **u**'s rating

$$p(\mathbf{u}, i) = \frac{\sum_{j \in J} r_{\mathbf{u},j} \times sim(i, j)}{\sum_{j \in J} sim(i, j)}$$

where J is the set of k similar items

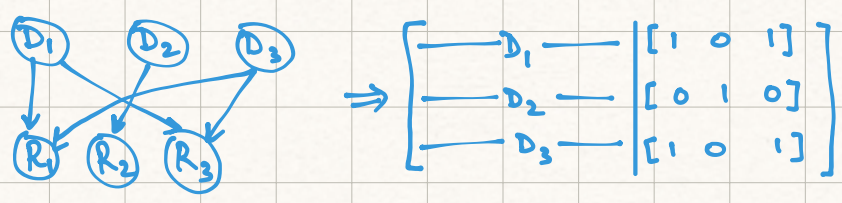
Deep learning approach

[—D₁— —R₁— | —E₁—]

Dataset $\{D_i, R_j\}$

$D_i \rightarrow$ Developer's feature vector

$R_j \rightarrow$ probability vector of repos. for each developer



Input :- Dnew (feature vector)

Output :- $[x_1 \; x_2 \; \ldots \; x_m]$, here, x_i is the probability that Dnew will contribute in R_i

weight updation → G.D.

Why use GNNs?

→ CF → only captures user-item interaction patterns and the similarities b/w users and items. (limitation - sparse datasets)

→ DL (FFNN + BP) → instead of [—D₁— | [1 0 1]] we can input edge list, like [—D₁— | —R₁— | —E₁—] but this won't give info. about the graph structure and multi-hop info. and it would be like using a usual feed forward neural network for images by passing info. about each pixel as input, instead of using CNN model which is specially designed for image data.

→ GNN → captures neighborhood info., handles heterogeneous data well, easy scalability with new data added., captures graph's structure.

A simple neural network that directly takes developer node features and predicts the probability of contributing to each repository may struggle to capture complex interactions and non-linear relationships between nodes and edges. GNNs, through their message-passing mechanism and iterative updates, can capture intricate dependencies in the graph structure and learn non-linear transformations, enabling them to model more nuanced patterns and deliver more accurate recommendations.

GNNs ⇒ like CNNs but instead of aggregating feature info. from spatially defined patches in an image, they aggregate info. based on local graph neighborhoods.

* popularity based R.S → recommend top K things to everyone

* content based RS → if user likes an item then we recommed an item similar to the first item.

* collaborative filtering based RS → based on user's preferences and behaviour of similar users.

main idea:- if 2 users have similar views/interests or tastes for certain items, then they are likely to have similar opinions about other items as well. (generalising)

Note → dot product is a good way to represent 'similarity'. (or correlation)

1. user-based C.F. here we look for users who have rated items in a similar way to target user and recommends items that those similar users have positively rated but the target user has not yet interacted with.

2. Item-based C.F. finds items that are very often rated together by users and suggests items that have not been explored by target user.

* Hybrid RS → A mix of above; used when either data is too small or too big or not a good representation of population.

⊛ dif. b/w Content based R-S. and Item based CF?

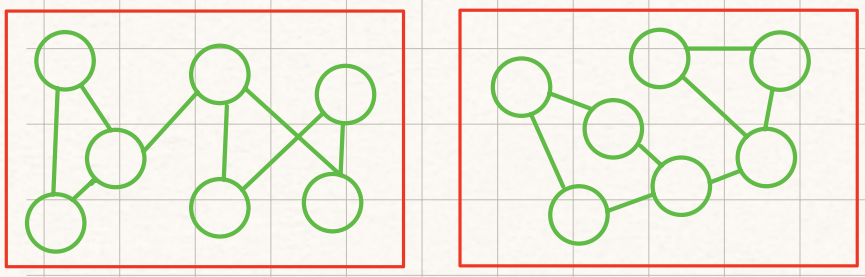
→ in Content based R.S. items are 'similar' based on item features and in Item-based C.F. items are considered similar based on user-item interaction.

2 approaches (C.F.)

1.) Just calculate similar items and recommend them
2.) Predict the missing values in the rating matrix

C.F. algos. for benchmarking

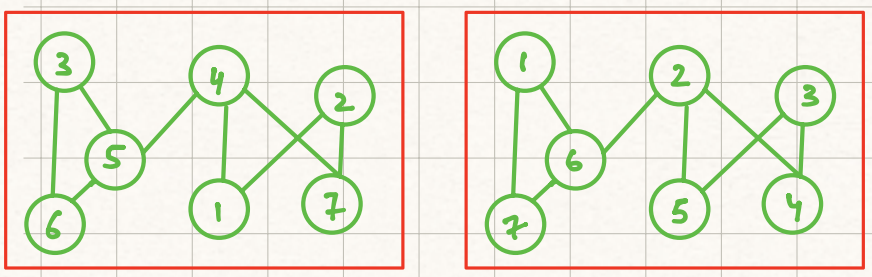
SVD, SVD++, KNN, KNN (with means), KNN (with z-score), SGD



CNNs

'person' 'car'

$$\begin{bmatrix} | & | & | \\ | & 0 & | \\ | & | & | \end{bmatrix}$$



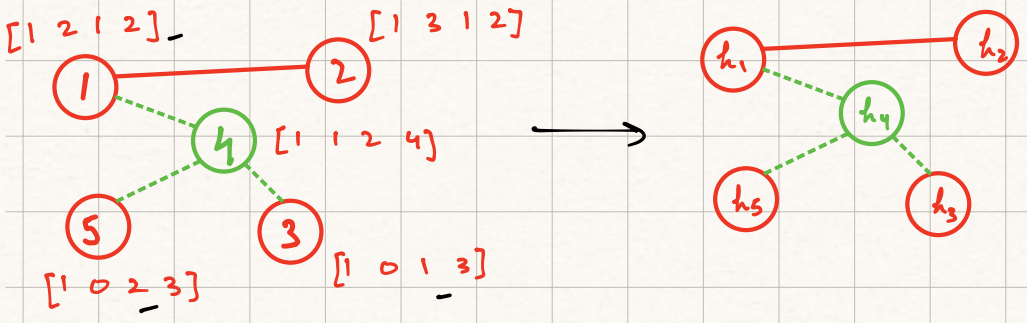
$$f\left(P\begin{bmatrix} - & x_1 & - \\ - & x_2 & - \\ & \vdots & \\ - & x_I & - \end{bmatrix}\right) = f(X) \rightarrow \begin{bmatrix} - & D_1 & - \\ & \vdots & \\ - & R_1 & - \\ & \vdots & \end{bmatrix}$$

$$f(PX, P^T A P) = f(X, A) \rightarrow \text{permutation invariance}$$

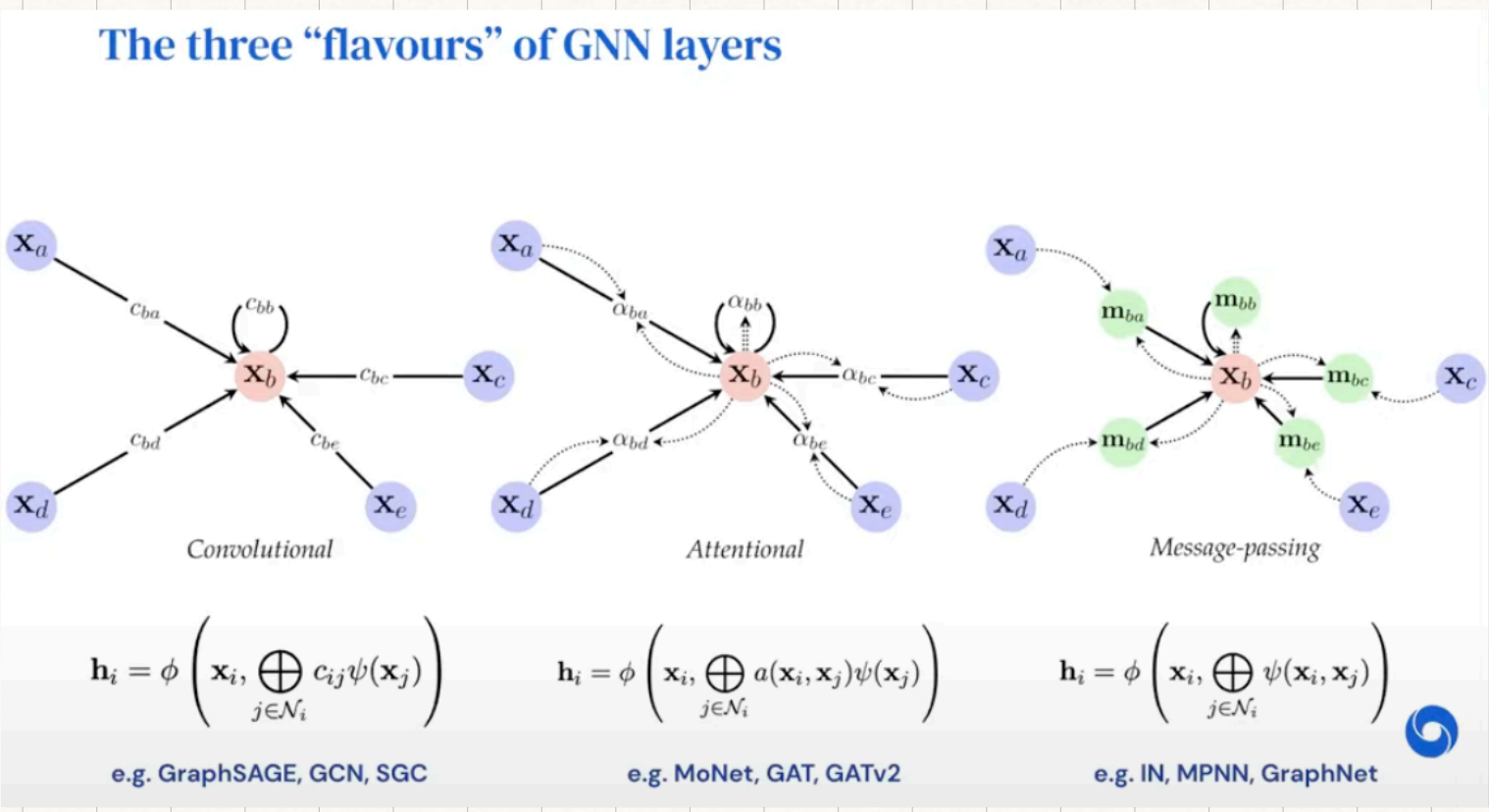
$$f(PX, P^T A P) = P f(X, A) \rightarrow \text{permutation equivariance}$$

$$f(X, A) = \begin{bmatrix} - & h_{x_1} & - \\ - & h_{x_2} & - \\ & \vdots & \end{bmatrix}$$

$$h_{x_i} = g(x_i, \Theta(N(x_i)))$$



$$h_4 = \text{update}\left([1\ 1\ 2\ 4],\ \text{agg}\left(\begin{matrix}[1\ 2\ 1\ 2]\\ [1\ 0\ 2\ 3]\\ [1\ 0\ 1\ 3]\end{matrix}\right)\right)$$



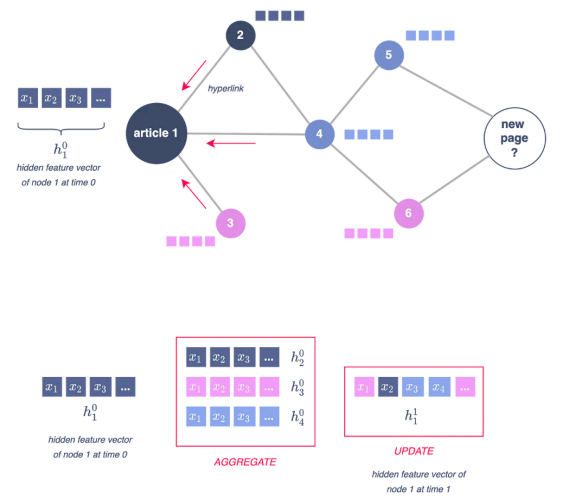


Figure 5 — Illustration of a message passing layer, illustration by Lina Faik

The message-passing update is expressed as follows:

$$h_u^{(k+1)} = UPDATE^{(k)}(h_u^{(k)}, AGGREGATE^{(k)}(h_v^{(k)}, \forall \nu \in N(u))$$

GCN

$$h_u^{(k+1)} = \sigma(W^{(k)} \sum_{v \in \tilde{N}(u)} \frac{h_v^{(k)}}{\sqrt{|\tilde{N}(u)||\tilde{N}(v)|}}), \quad \forall k \in \{0, \cdots, K-1\}$$

total number of layers

hidden feature vector of node u at time k+1

self-loop

non-linearity (e.g. ReLU)

sum of normalized neighbor embedding

Figure 6 — Message passing function, illustration by Lina Faik

GCN

2.3. Graph Attention Network (GAT)

What are the key concepts behind the model?

GCN models rely on a symmetric-normalized aggregation function that favors nodes depending on their degree. But what if the model could also learn what weight to give to each neighbor during the aggregation step? This can be done by including attention mechanisms in the learning process of the GNN.

The first GNN model to apply attention was Velickovic et al. [2018]'s Graph Attention Network (GAT), which uses attention weights to define the weighted sum of the neighbors:

$$h_u^{(k+1)} = \sigma(W^{(k)} \sum_{v \in N(u)} \alpha_{u,v} h_v^{(k)})$$

attention on neighbor v

GAT

heterogeneous graphs : graphs with dif. types of nodes

GCNs issues :

- Difficulties in learning from large networks: does not allow the model to be **trained in batches.**
- Difficulties to **generalize** to unseen nodes: But, in many real-world applications, the embeddings of unseen nodes need to be quickly generated (e.g., posts on Twitter, videos on YouTube, etc.)

GraphSAGE :

- GCN with subsampled neighbors.
- Sample and aggregate(mean, LSTM, max-pool)

$$h_u^{(k)} = \sigma(W \cdot MEAN(\{h_u^{(k-1)}\} \cup \{h_v^{(k-1)}, \forall \nu \in N(u)\})$$

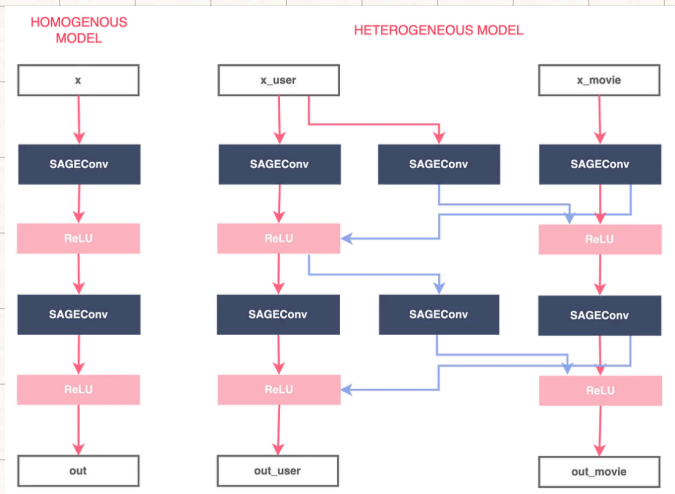


Figure 3— Conversion of a regular GNN model to a GNN model adapted to heterogeneous graphs, illustration by Lina Faik, inspired by [3]

https://pytorch-geometric.readthedocs.io/en/latest/notes/heterogeneous.html

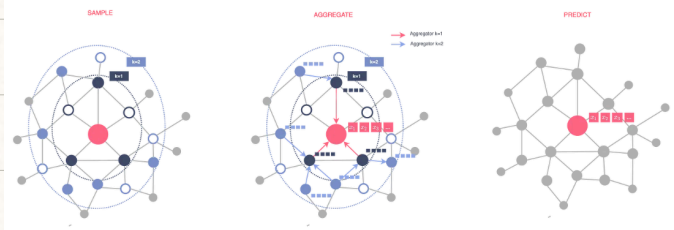


Figure 1 — Illustration of GraphSAGE approach, illustration by Lina Faik, inspired by [1]

Option 1. Train an additional linear model that takes (as an input) the concatenation of the user and movie embeddings to predict the rating. This is the approach implemented in the application part below.

Figure 5 — From node embeddings to link prediction (option 1), illustration by Lina Faik

Option 2. The other alternative consists of either:

- Computing a simple dot product, provided that the dimensions of both embeddings are the same and that the output is one-dimensional (e.g., predict whether the link exists or not):

$$\hat{y}_{um} = h_u^T \cdot h_m$$

- Learning a trainable matrix $W = (W^1, \ldots, W^{k})$ so that:

$$\forall i \in [1, k], \quad \hat{y}_{um}^i = h_u^T W^i h_m$$

✓ **Model evaluation**

As shown in Figure 5, users tend to give ratings between 3 and 4 more frequently than other ratings. To this into account when training and evaluating the model, a weight is associated with each rating as follows:

$$w_i = \frac{max_{i \in [0,5]}(c_i)}{c_i}, \forall i \in [0, 5]$$

where c_k is the number of occurrences of the rating k.

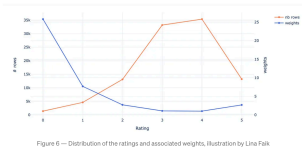


Figure 6 — Distribution of the ratings and associated weights, illustration by Lina Faik

In this context, the RMSE is weighted as follows:

$$wRMSE = \sqrt{\frac{1}{N} \sum_i w_i (r_i - \hat{r}_i)^2}$$

GCMC :

- introduces a graph-based auto-encoder framework for matrix completion

Intro. to GNNs (Part 1)

Why GNNs needed?

→ We cannot resize graphs like images, dif. sized graphs will need dif. NNs. ~~which~~ bcoz NNs have a constant size of input.

Size independence

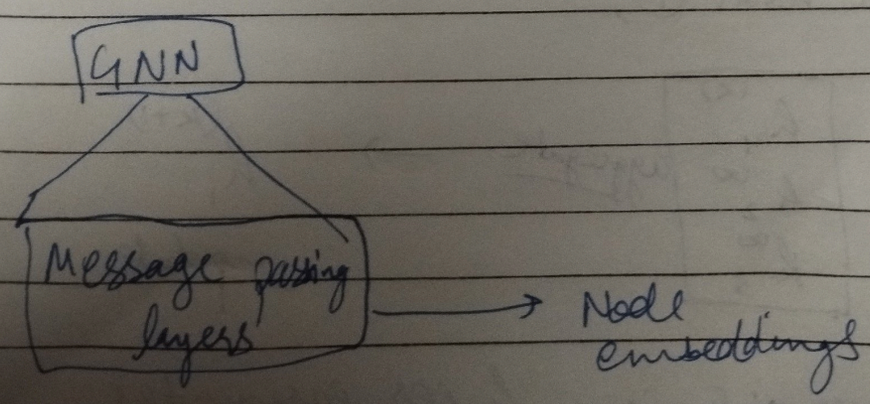
→ Isomorphism (permutation invariance: we cannot have dif. outputs for dif. node ordering or numbering; so we don't give adj. matrix as the input)

→ Graphs (unlike images) are non-euclidean (there are dif. types of geometries) so, we cannot feed them into CNNs.

(*) Grids are graphs but can be considered euclidean graphs

Graph → [GNN] → Node embeddings (NE) → Output

(*) the size of N.E. is a hyperparameter

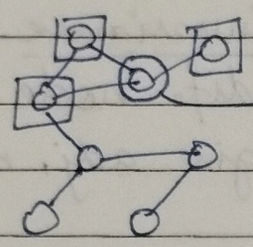


Part 2 (GNN and it's variants)

Message passing layer → combines the neighborhood info. to generate node embeddings

(This approach is called graph convolutions)

↳ it's like applying convolutions on graphs



→ This node shares embedding with the 3 neighbouring nodes

↓

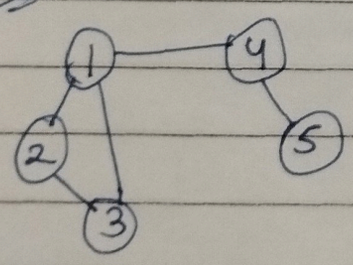
This is called sharing mechanism

↓

This can be seen as exchange of messages among the neighbouring nodes and hence the name message passing layer.

$h_1^{(k)}$ → node state in k^{th} pass of M.P.L. of node 1

eg



All these nodes are feature vectors. The NEs store the structural & feature-based info.

for node 1,

$$h_1^{(k)} \quad \boxed{\begin{matrix} h_4^{(k)} \\ h_2^{(k)} \\ h_3^{(k)} \end{matrix}} \xrightarrow{\text{aggregate}} h_1^{(k+1)}$$

update

(*) Size of these N.E. is a hyper parameter

local feature aggregation $\approx$ CNN kernels

No. of msg. passing layers = no. of neighbourhood hops info.

disadv.

→ too many M.S. layers : all node's embeddings become same (solution : Paranorm)

→ whole graph needed; it would be difficult if the graph is huge.

$$h_v^{(k+1)} = \text{UPDATE}^{(k)}\left(h_v^{(k)}, \text{AGGREGATE}^{(k)}\left(\{h_v^{(k)}, \forall v \in \underbrace{N(v)}_{\text{neighbors of } v}\}\right)\right)$$

different GNN models have different ~~UPD~~ UPDATE and AGGREGATE functions (mean, max, RNN, normalised sum etc.)

(*) These should be differentiable.

egs.

① Graph Convolutional Networks (GCN)

$$h_v^{(k)} = \sigma\left(W^{(k)} \sum_{v \in N(v) \cup \{v\}} \frac{h_v}{\sqrt{|N(v)||N(v)|}}\right)$$

(*) update and aggregate are combined here

→ self-loops for each node

→ Sum of normalised neighbor embeddings

→ σ is non-linearity, like ReLU

② MLP as Aggregator

$$\underbrace{m_{N(u)}}_{\text{aggregated msg.}} = \underbrace{MLP_\theta}_{\text{trainable}} \left(\sum_{v \in N(u)} MLP_\phi(h_v) \right)$$

→ sends node features through a MLP

③ Graph attention networks

$$m_{N(u)} = \sum_{v \in N(u)} \alpha_{u,v} \cdot h_v$$

here, $\underbrace{\alpha_{u,v}}_{\text{attention wts for nodes } u \text{ and } v} = \dfrac{\exp(a^T [Wh_u \oplus Wh_v])}{\sum_{v' \in N(u)} \exp(a^T [Wh_u \oplus Wh_{v'}])}$

→ all neighbors are given dif. $\underbrace{\text{wts.}}_{\alpha \to \text{learnable}}$ while aggregating.

→ $\oplus$ is

④ Gated Graph N.N.

$$h_u^{(k)} = GRU(h_u^{(k-1)}, m_{N(u)}^{k})$$

→ Recurrent update of the state (use RNNs)

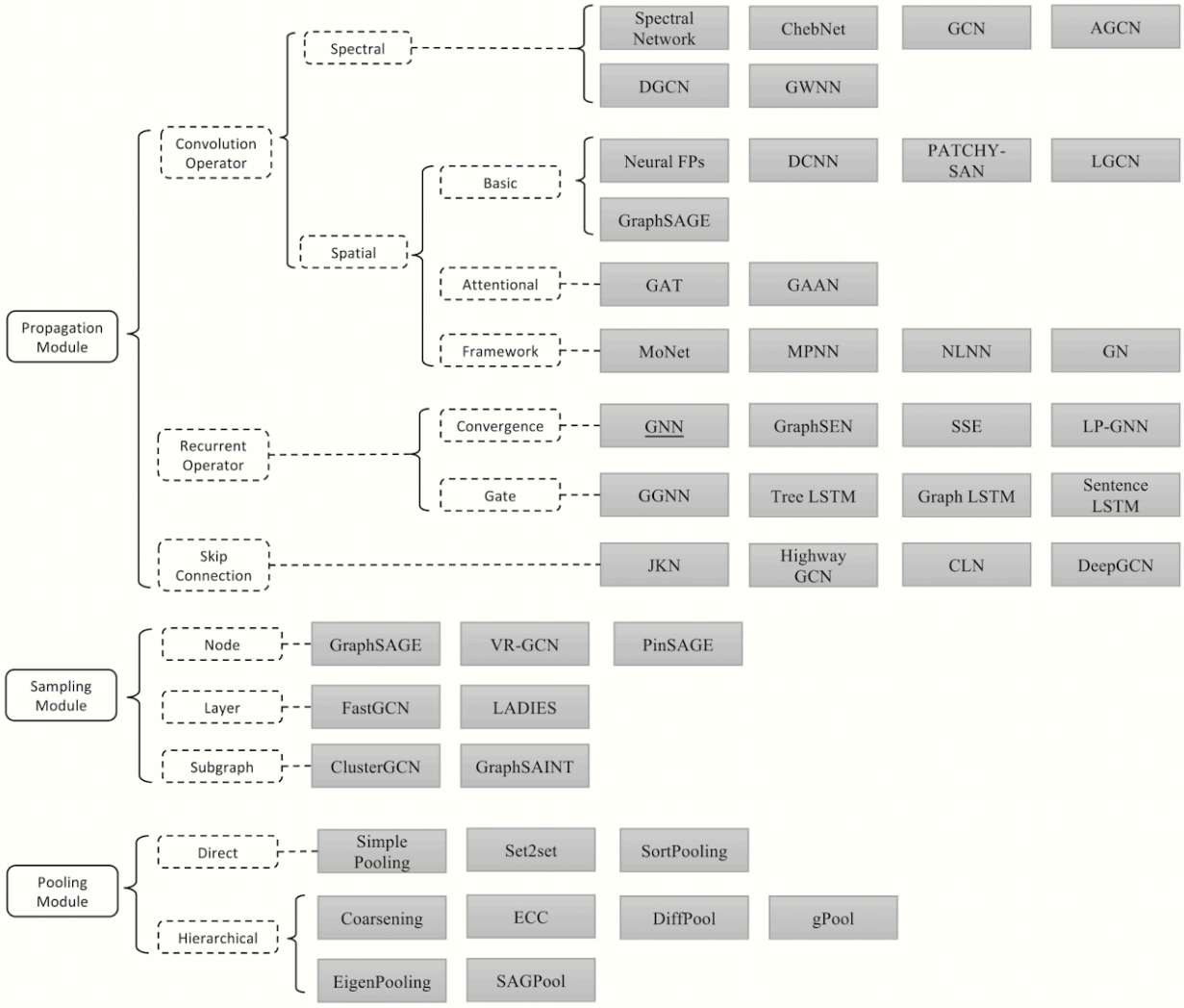


Fig. 3. An overview of computational modules.

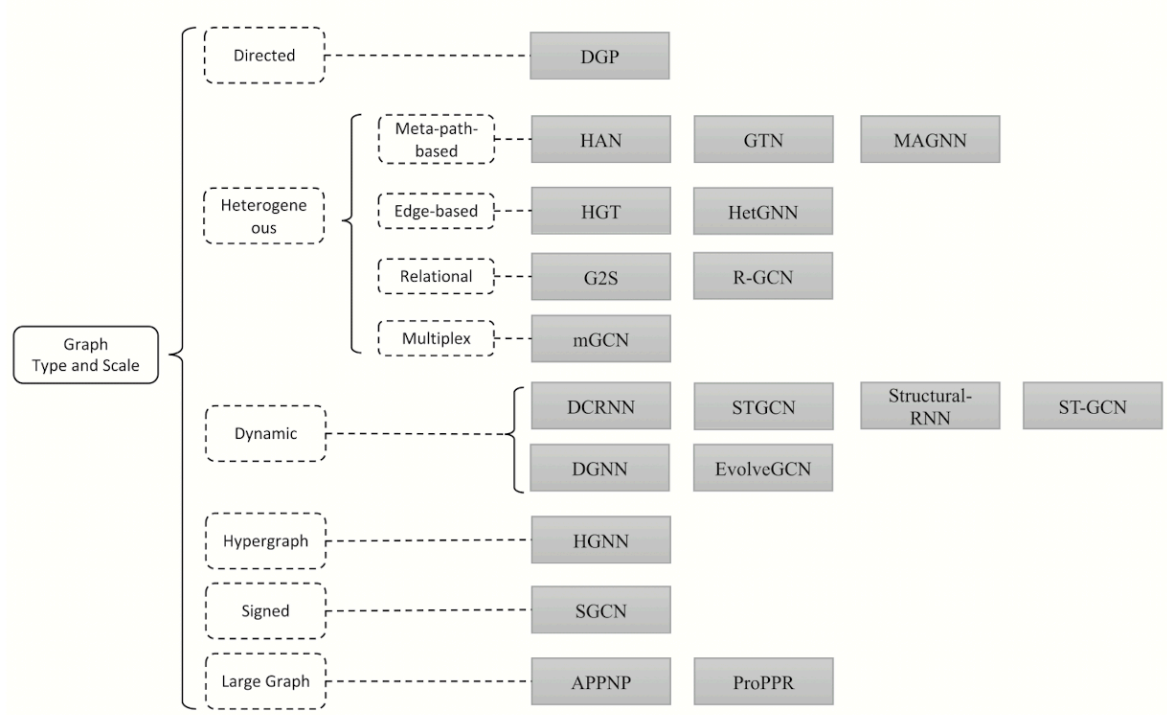


Fig. 4. An overview of variants considering graph type and scale.

Node Classification

→ label an ~~unlabel~~ unlabeled node in a network

CORA dataset → ⊛ nodes – Publication
– features : word vectors
⊛ edges – citations.
⊛ 7 labels : topic of publication

Bag of Words

for representing node features

dictionary size = 1433 words

Representation for the content of publication = [2 → Neural, 7 → Network, ⋮, 3 → ReLU]

(The word 'neural' occured 2 times in this publication)

↓

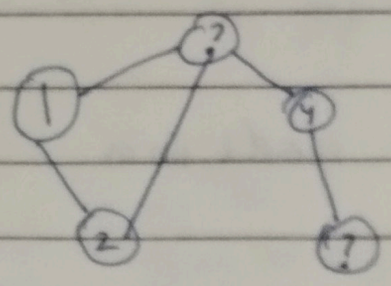
apply normalisation

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

→ downloading and preprocessing the data

Binary masks for Node-level predictions

data.train_mask



⇒ Masking ⇒

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

0 → has label
1 → don't have label

for training data we will unlabel some nodes.

Why not just use the bag of words for predicting the label?

» MLP models doesn't performs well in this case. GNNs perform better, bcoz of additional citation info.

⊛ TSEN → for seeing the model being trained in a 2D embedding space.

→ Cross validation
→ hyperparameter optimisation
→ dif layer types – GCN, GAT etc.
→ include edge features
• 0.9 accuracy: SSP model (for CORA)
(Best till now)

Edge features in GNNs

Node features of $N(v)$ → $\boxed{\text{transform}}$ (MLP layer) → high level representation → $\boxed{\text{Aggregate}}$ (norm)

↓

permutation invariant

↓

features(v) + aggregated sum

→ $\boxed{\text{update}}$ embeddings of node v

→ $\boxed{\text{calc. loss and learn}}$

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

adding self loops,

$$h_v^{k+1} = \text{AGGREGATE}_{j \in N(v)} \text{TRANSFORM}\left(h_j^{(k)}\right)$$

$$\underset{\text{new embedding}}{X'} = \underbrace{\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2}}_{\text{Normalised adjacency matrix}} \overset{\text{Node features}}{X} \underset{\text{transform}}{\Theta}$$

discrete edge types: molecules, linked by a single, double or triple bond, labeled as 1, 2, 3

Relational GCN

$$h_i^{(l+1)} = \underbrace{\sigma}_{\text{ReLU}} \left[\underbrace{\sum_{r \in R}}_{\text{different type of edges}} \underbrace{\sum_{j \in N_i^r}}_{\text{aggregate}} \underbrace{\frac{1}{c_{i,r}}}_{\text{Normalisation}} \underbrace{W_r^{(l)}}_{\text{transform on neighbor nodes}} h_j^{(l)} + \underbrace{W_0^{(l)} h_i^{(l)}}_{\text{transformation applied on our node } i} \right]$$

i.e. we apply dif. transformations for dif. types of edges, and the we are summing on all types of edges

(Edge conditioned GNN)

$W_r \rightarrow$ also have 'r' parameter

GNN with Feature-wise Linear Modulation (GNN FiLM)

$$h_v^{(t+1)} = l\left(\sum_{u \xrightarrow{l} v \in E} \sigma\left(\gamma_{l,v}^{(t)} \odot W_l h_u^{(t)} + \beta_{l,v}^{(t)} \right); \theta_l \right)$$

here, l is the types of labels of edges

Multi-dimensional edge features

$$h_v = \underset{\text{update}}{\gamma}\Big(x_v, \underset{w \in N(v)}{\bigoplus} \underset{\text{transform}}{\phi}(x_v, x_w, e_{wv})\Big)$$

aggregate

edge feature b/w nodes w and v

adj. matrix → matrix of edge feature vectors

shape ⇒ $[n, n, f(e)]$

n → no. of nodes

$f(e)$ → no. of features in an edge vector.

MP-GNN (Neural msg. passing for quantum chem. → paper)

msg function ⇒ $m_v^{t+1} = \sum_{w \in N(v)} M_t(\underbrace{h_v^t, h_w^t}_{\text{nodes features}}, \underbrace{e_{vw}}_{\text{edge feature}})$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$ → update function

transformation → MLP

PNAConv

Principal neighbourhood aggregation for GraphNets

$$X_i^{(t+1)} = U\left(X_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(X_i^{(t)}, \underbrace{E_{j\to i}}_{\text{edge features in the transformation}}, X_j^{(t)}\right)\right)$$

Crystal Graph Conv

$$z_{(i,j)_k}^{(t)} = v_i^{(t)} \oplus v_j^{(t)} \oplus \underbrace{u_{(i,j)_k}}_{\text{edge feature}}$$

$$v_i^{(t+1)} = v_i^{(t)} + \sum_{j,k} \sigma\left(z_{(i,j)_k}^{(t)} W_f^{(t)} + b_f^{(t)} \right) \odot g\left(z_{(i,j)_k}^{(t)} W_s^{(t)} + b_s^{(t)} \right)$$

learnable wt. matrix also have edge features included

Creating edge embeddings

NENN paper

paper: Incorporate Node and edge features in GNNs

"Hierarchical dual-level attention mechanism"

→ alt. layers for node and edge embedding creation (iterative updation)

→ it also have attention mech. as the name suggests (here, it's α and β)

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

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

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

Node-level attention layer

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

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

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

Edge-level attention layer

EGNN

→ Exploiting edge features for GNNs

$$X^l = \sigma\left[\Big\|_{p=1}^{P} \Big(\underbrace{\alpha^l_{\cdot\cdot p}}_{\text{attention coeff.}} (X^{l-1}, E^{l-1}_{\cdot\cdot p})\, g^l(X^{l-1})\Big)\right]$$

$$\alpha^l_{\cdot\cdot p} = DS(\hat{\alpha}^l_{\cdot\cdot p}),$$

$$\hat{\alpha}^l_{ijp} = f^l(X^{l-1}_i, X^{l-1}_j)\, E^{l-1}_{ijp}$$

$$E^l = \alpha^l$$

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

From Graph Neural Networks with Feature-wise Linear Modulation, Brockschmidt

Pytorch

GNN models with edge-weight, edge-type, edge-attr allow edge features.

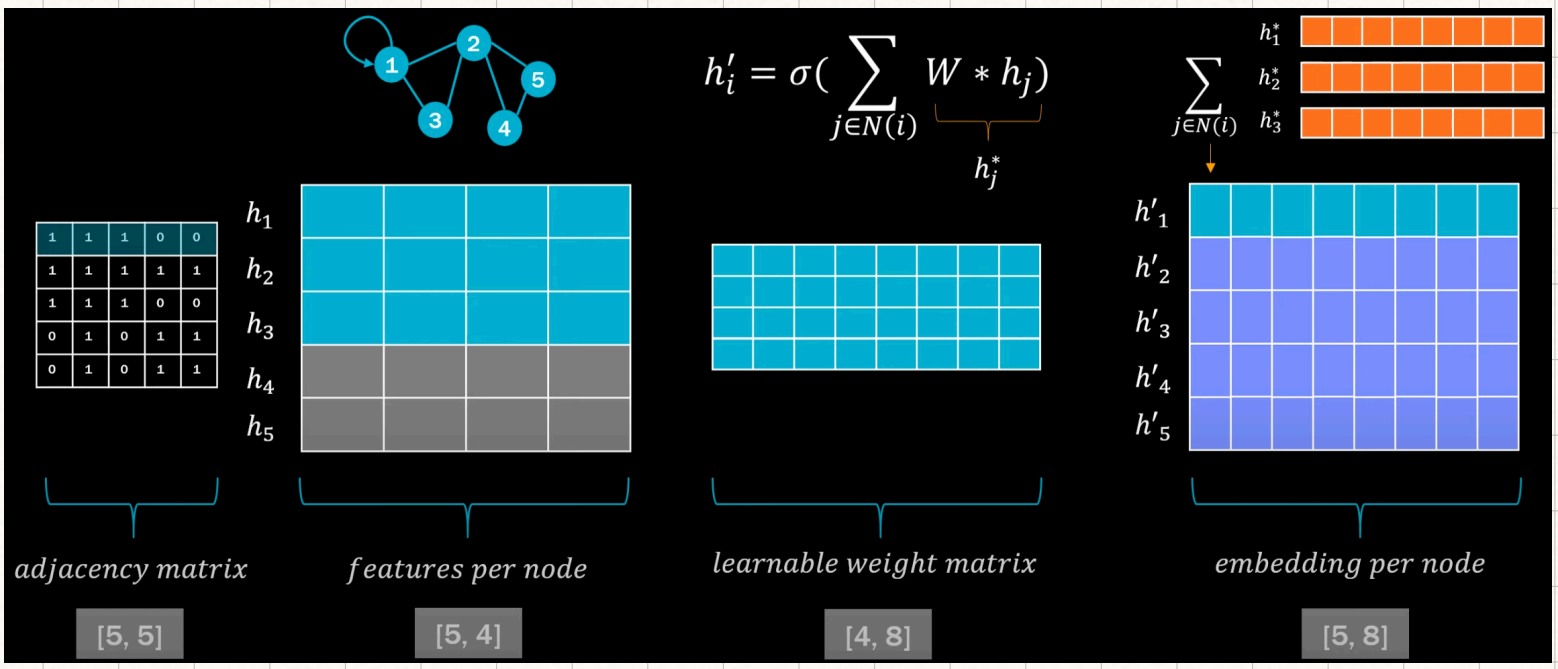
Graph attention networks (GAT)

→ each neighbor gets dif. attention while aggregating

node update Rule (General) $\Rightarrow h_i' = \sigma\left(\sum_{j \in N(i)} W * h_j \right)$

$h_i \rightarrow$ real node feature vector.

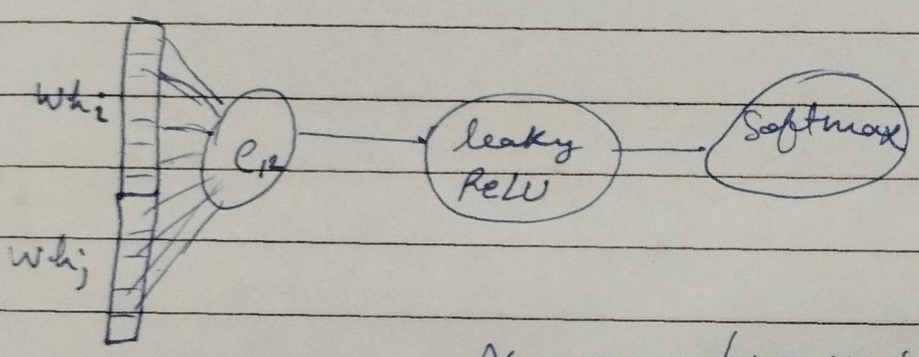
$W \rightarrow$ learnable wt. matrix



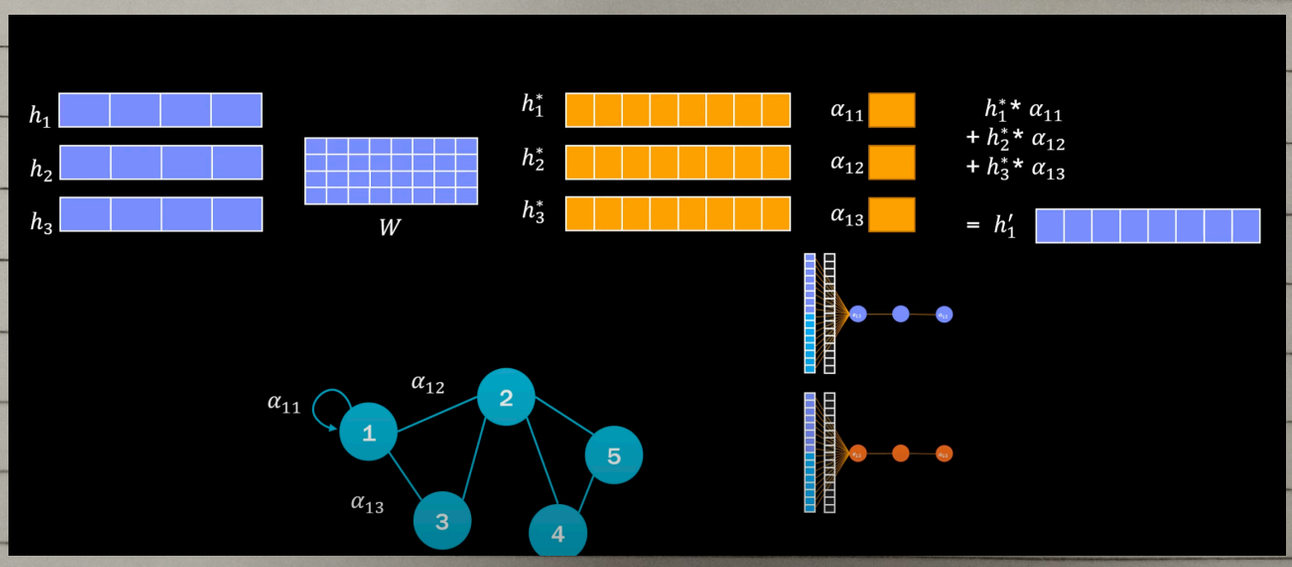
$$e_{ij} = a(Wh_i, Wh_j) \qquad \text{(attention coefficients)}$$

$$\underbrace{\alpha_{ij}}_{\text{Normalised attention coeff.}} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\underbrace{\sum_{k \in N(i)} \exp(e_{ik})}_{\text{masked attention}}} = \frac{\exp(a(Wh_i, Wh_j))}{\sum_{k \in N(i)} \exp(a(Wh_i, Wh_j))}$$

masked attention: only taking neighbor nodes and not all nodes.



$$\alpha_{ij} = \frac{\exp(\text{LReLU}(\vec{w}_a^T [Wh_i \,||\, Wh_j]))}{\sum_{k \in N(i)} \exp(\text{LReLU}(\underbrace{\vec{w}_a^T}_{\text{learnable wt. vector}} [Wh_i \,||\, Wh_j]))}$$



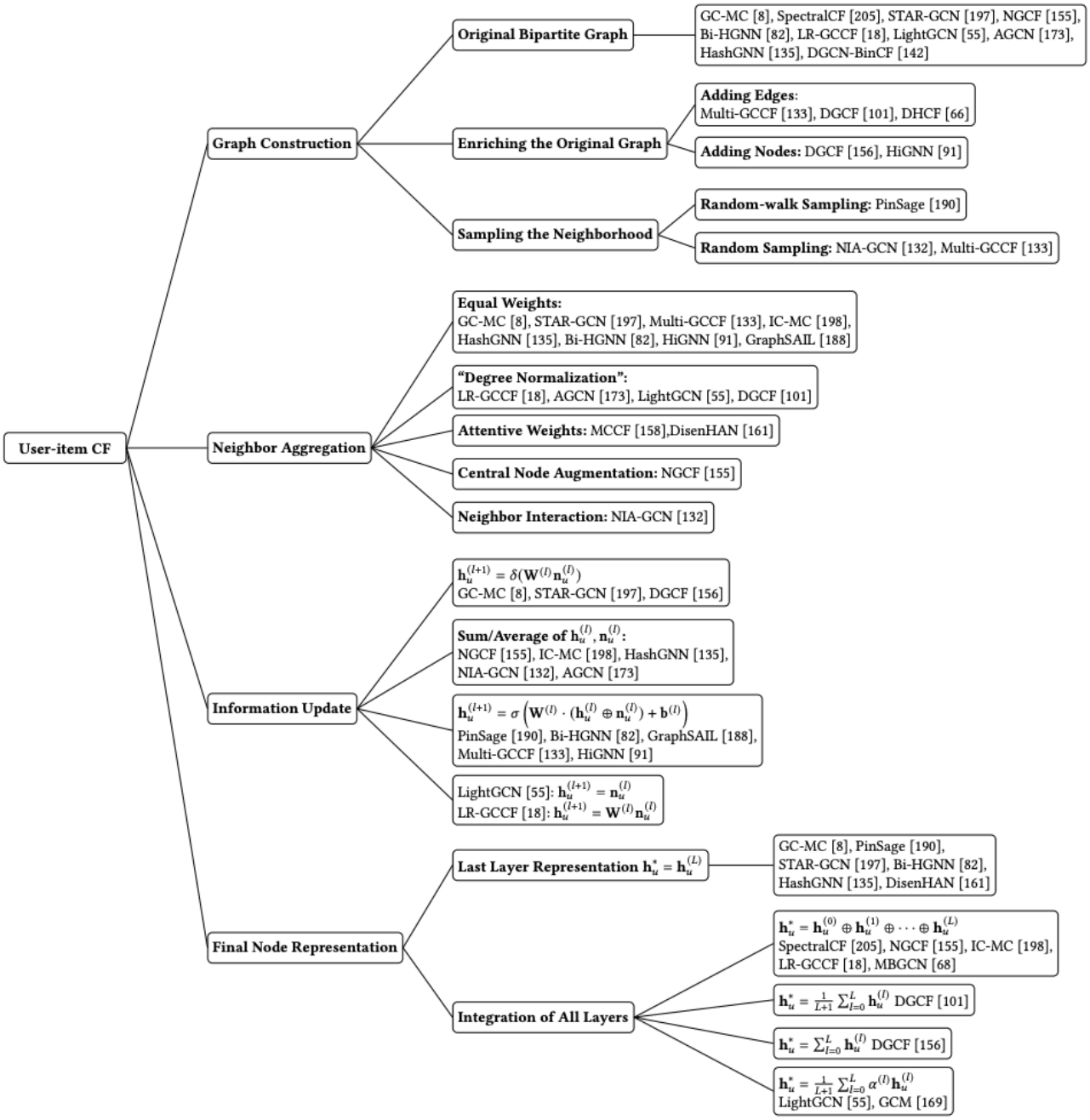


Fig. 3. Summary of representative works in user-item collaborative filtering.

Theoretical Foundations of Graph Neural Networks : https://youtu.be/uF53xsT7mjc

→ molecules as graphs (predict : potent drug)

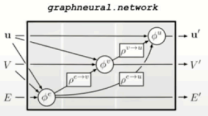
Transportation maps :- Run GNN on supersegment graph to estimate time of arrival.

Rich ecosystem of libraries

PyTorch geometric — github.com/rusty1s/pytorch_geometric

Spektral — graphneural.network

DGL — dgl.ai



github.com/deepmind/graph_nets

github.com/deepmind/jraph

Rich ecosystem of datasets

OGB — ogb.stanford.edu

PyTorch geometric — https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

TUDataset — graphlearning.io

Benchmarking Graph Neural Networks

github.com/graphdeeplearning/benchmarking-gnns

- One very generic form is the *Deep Sets* model (Zaheer et al., NeurIPS'17): $f(\mathbf{X}) = \phi\left(\sum_{i\in\mathcal{V}} \psi(\mathbf{x}_i)\right)$ where ψ and ϕ are (learnable) functions, e.g. MLPs.
  - The **sum** aggregation is *critical*! (other choices possible, e.g. **max** or **avg**)

??

Permutation *invariance*

- We want to design functions f(**X**) over sets that will not depend on the order
- Equivalently, applying a permutation matrix shouldn't modify the result!
- We arrive at a useful notion of **permutation invariance**. We say that f(**X**) is permutation *invariant* if, for *all* permutation matrices **P**:

$$f(\mathbf{PX}) = f(\mathbf{X})$$

- One very generic form is the *Deep Sets* model (Zaheer et al., NeurIPS'17): $f(\mathbf{X}) = \phi\left(\sum_{i\in\mathcal{V}} \psi(\mathbf{x}_i)\right)$ where ψ and ϕ are (learnable) functions, e.g. MLPs.
  - The **sum** aggregation is *critical*! (other choices possible, e.g. **max** or **avg**)

sum function : agnostic(unknown) to the ordering

Permutation *equivariance*

- Permutation-*invariant* models are a good way to obtain set-level outputs
- What if we would like answers at the **node** level?
  - We want to still be able to **identify** node outputs, which a permutation-invariant aggregator would destroy!
- We may instead seek functions that don't **change** the node order
  - i.e. if we permute the nodes, it doesn't matter if we do it **before** or **after** the function!
- Accordingly, we say that f(**X**) is **permutation equivariant** if, for all permutation matrices **P**:

$$f(\mathbf{PX}) = \mathbf{P}f(\mathbf{X})$$

General blueprint for learning on sets

- Equivariance mandates that each node's row is unchanged by f. That is, we can think of equivariant set functions as transforming each node input $\mathbf{x}_i$ into a *latent* vector $\mathbf{h}_i$:

$$\mathbf{h}_i = \boxed{\psi(\mathbf{x}_i)}$$

where ψ is any function, applied in isolation to every node. Stacking $\mathbf{h}_i$ yields **H** = f(**X**).

- We arrive at a general blueprint: (stacking) **equivariant** function(s), potentially with an **invariant** tail---yields (m)any useful functions on sets!

$$f(\mathbf{X}) = \boxed{\phi\left(\bigoplus_{i\in\mathcal{V}} \boxed{\psi(\mathbf{x}_i)}\right)}$$

Here, $\bigoplus$ is a permutation-invariant **aggregator** (such as sum, avg or max).

(remark: this is typically **as far** as we can get with sets, without assuming or inferring additional structure)

whatt are deep sets?

PERPLEXITY

Deep Sets refer to a type of neural network architecture that includes two learnable functions, ψ and ϕ, which can be implemented using MLPs (multi-layer perceptrons). The sum aggregation is a critical component of this model, although other aggregation methods such as max or avg are also possible. Deep Sets models have been used for tasks such as 3D classification and segmentation, combinatorial optimization, and ride-sharing [1].

1 Pointhop++: A Light...

Permutation invariance and equivariance on **graphs**

- The main difference: node permutations now also accordingly act on the **edges**
- We need to appropriately permute both **rows** and **columns** of **A**
  - When applying a permutation matrix **P**, this amounts to **PAP**$^\top$
- We arrive at updated definitions of suitable functions f(**X, A**) over graphs:

Invariance: $f(\mathbf{PX}, \mathbf{PAP}^\top) = f(\mathbf{X}, \mathbf{A})$

Equivariance: $f(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$

Locality on graphs: **neighbourhoods**

- On **sets**, we enforced equivariance by applying functions to every node **in isolation**
- **Graphs** give us a broader context: a node's **neighbourhood**
  - For a node *i*, its (1-hop) neighbourhood is commonly defined as follows:

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

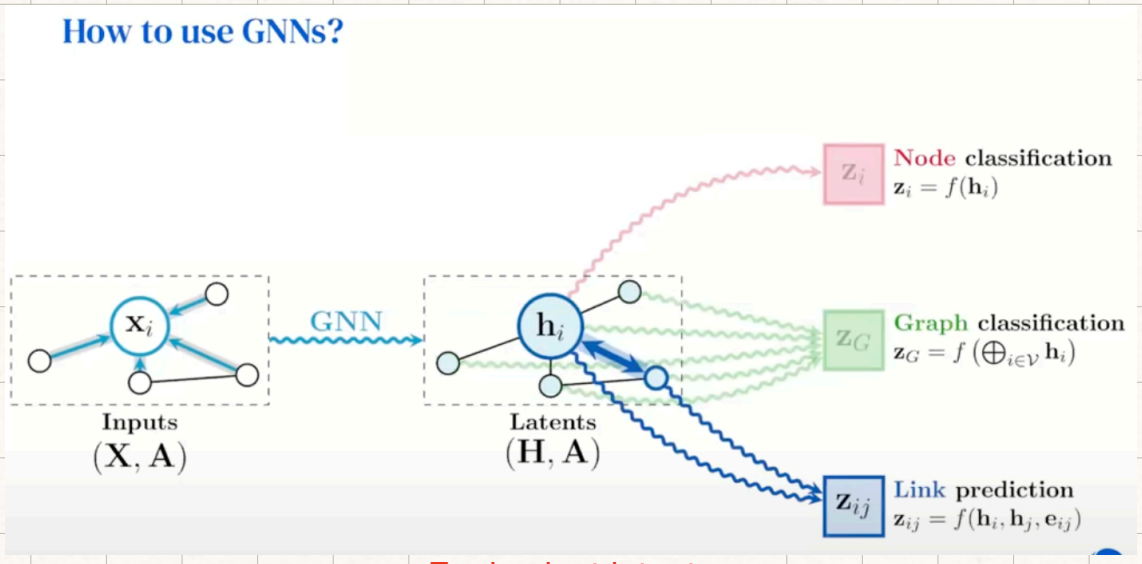
N.B. we do not explicitly consider *directed* edges, and often we assume $i \in N_i$

- Accordingly, we can extract the *multiset* of **features** in the neighbourhood

$$\boxed{\mathbf{X}_{\mathcal{N}_i} = \{\!\{\mathbf{x}_j : j \in \mathcal{N}_i\}\!\}}$$

and define a *local* function, g, as operating over this multiset: $g(\mathbf{x}_i, \mathbf{X}_{N_i})$.

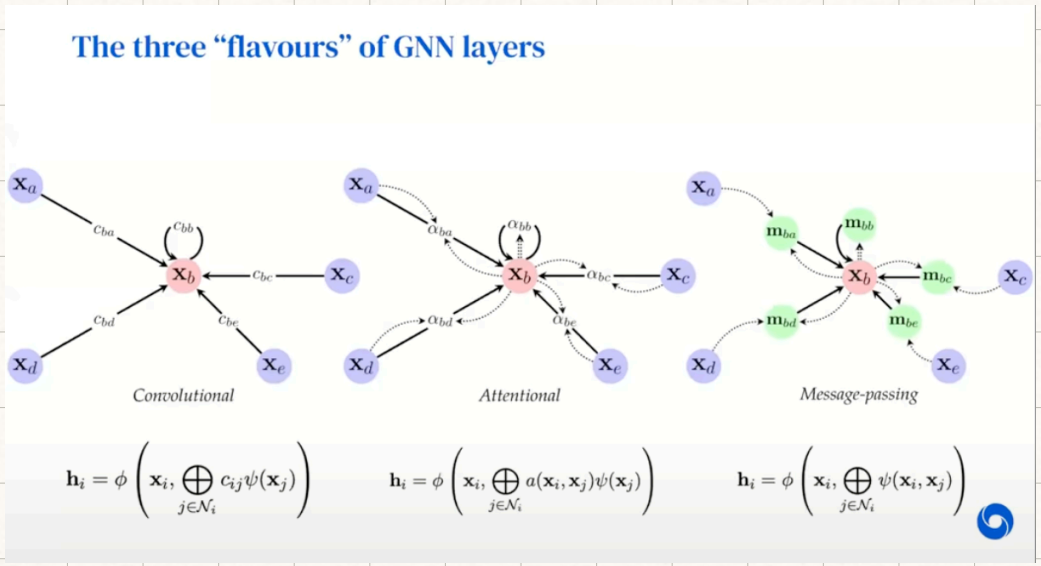
local function → 1 hop neighbourhood

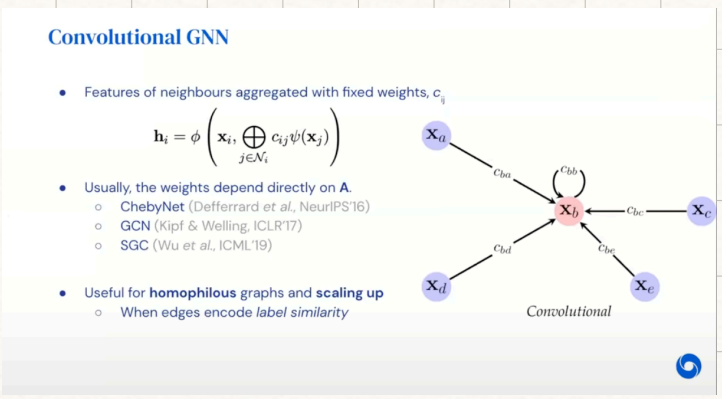


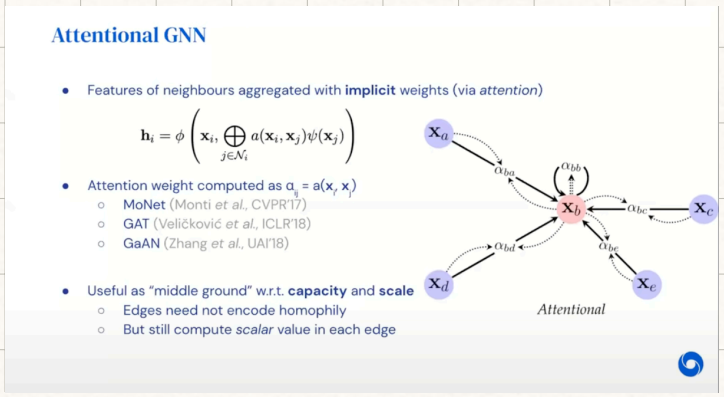
Equivariant latents

GNN ≈ diffusion
propagation
msg. passing

GNN layers' flavours
- → Convolutional
- → Attentional
- → Msg passing







c_ij : interaction constant — i.e. how much does the node i values the features of node j

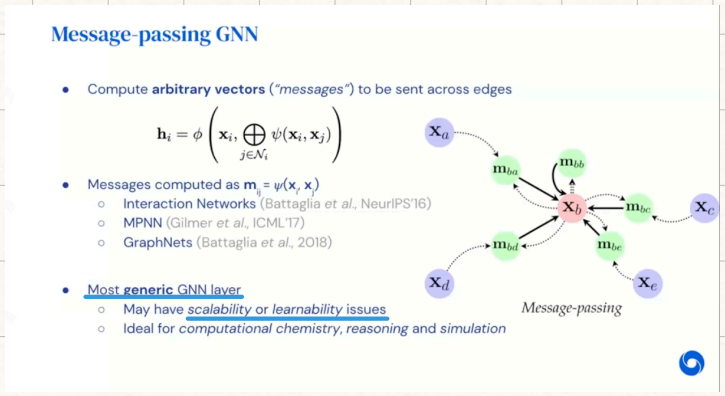
phi(inside one) function gives the locally transformed features

that circle with a plus is some permutation invariant aggregator

Instead of pre-defined constants here we learn those constants

like transformers : attention mechanism

reasonably scalable : because only 1 scalar per edge



Sender and receiver collaborate to decide what's the message is going to be

generic so - widely used but scalability issues bcoz a lot of edges(wrt no. of nodes)

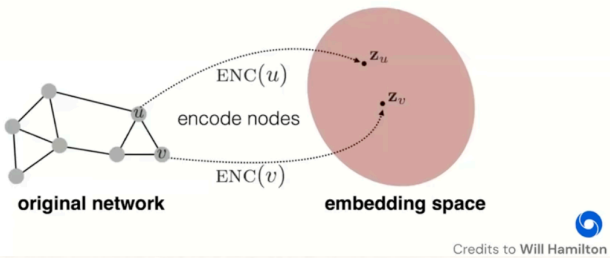
earlier : spatial Vs spectral and node2vec and deepwalk

now : this blueprint

node embeddings

Node embedding techniques

- Some of the earliest "successes" of deep learning on graphs relied on finding good ways to **embed** nodes into vectors $\mathbf{h}_u$ (below: $\mathbf{z}_u$) using an *encoder function*
  - At the time, implemented as a **look-up table**!



Credits to **Will Hamilton**

z is h — embedding

earlier all these encoder functions were lookup tables

What's in a *good* representation?

- What makes an embedding "good"?
  - Graphs carry interesting **structure**!
  - Good node representations should **preserve** it.
- Simplest notion of graph structure is an *edge*.
  - Features of nodes *i* and *j* should be predictive of existence of edge (*i*, *j*)!
- Yields a straightforward unsupervised objective
  - Optimise $\mathbf{h}_i$ and $\mathbf{h}_j$ to be **nearby** iff $(i, j) \in E$.
  - Can use standard *binary cross-entropy* loss:

$$\sum_{(i,j)\in E} \log \sigma\left(\mathbf{h}_i^\top \mathbf{h}_j\right) + \sum_{(i,j)\notin E} \log\left(1 - \sigma\left(\mathbf{h}_i^\top \mathbf{h}_j\right)\right)$$

Random-walk objectives

- This link-prediction objective is a special case of **random-walk** objectives.
- **Redefine** the condition from $(i, j) \in E$ to *i and j co-occur in a (short) random walk*.
- Dominated unsupervised graph representation learning prior to GNNs!
  - DeepWalk (Perozzi *et al.*, KDD'14)
  - node2vec (Grover & Leskovec, KDD'16)
  - LINE (Tang *et al.*, WWW'15)

Local objectives **emulate** Conv-GNNs

- Random walk objectives inherently capture **local** similarities.
- But a (convolutional) GNN *summarises local patches* of the graph!
  - Neighbouring nodes tend to highly overlap in *n*-step neighbourhoods;
  - Therefore, a conv-GNN enforces similar features for neighbouring nodes **by design**.
- From a representation perspective, DeepWalk-style models **emulate** a convolutional GNN!
- **Corollary 1:** Random-walk objectives can **fail** to provide useful signal to GNNs!
- **Corollary 2:** At times, DeepWalk can be matched by an **untrained conv-GNN**!
  - First spotted within DGI (Veličković *et al.*, ICLR'19)
  - Independently verified by SGC (Wu *et al.*, ICML'19)

A note on Transformers

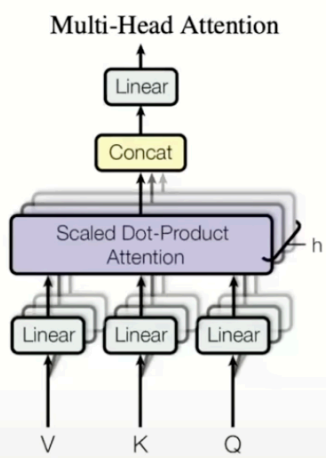
Transformers **are** Graph Neural Networks!

- Fully-connected graph
- Attentional flavour

The sequential structural information is injected through the **positional embeddings**. Dropping them yields a fully-connected GAT model.

Attention can be seen as inferring **soft adjacency**.

See Joshi (The Gradient; 2020).



Parallels with NLP : graph of words in a sentence. But how do we put edges? —> just make a complete graph and let the GNN learn weights

transformers can be considered as a special case of GNNs

https://thegradient.pub/transformers-are-graph-neural-networks/

spectral and spatial :

Look to the Fourier transform

- The **convolution theorem** defines a very attractive identity:

$$(x \star y)(\xi) = \hat{x}(\xi) \cdot \hat{y}(\xi) \qquad \hat{x}(\xi) = \int_{-\infty}^{+\infty} x(u) e^{-i\xi u} du$$

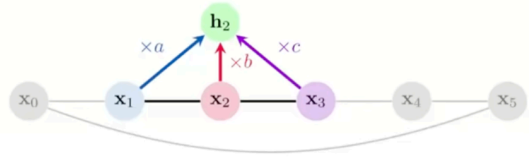
"convolution in the time domain is multiplication in the frequency domain"

- This could give us a 'detour' to defining convolutions on graphs
  - Pointwise multiplication is **easy**!
  - But what are the 'domains' in this case?
- We will first see how graphs arise in ***discrete*** sequences.

Rethinking the convolution on *sequences*

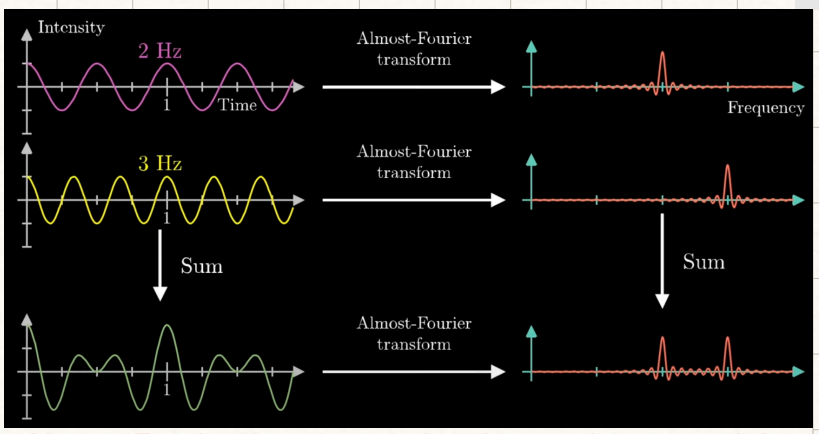
*for easier handling of boundary conditions

- We can imagine a sequence as a *cyclical* ***grid*** *graph*, and a **convolution** over it:



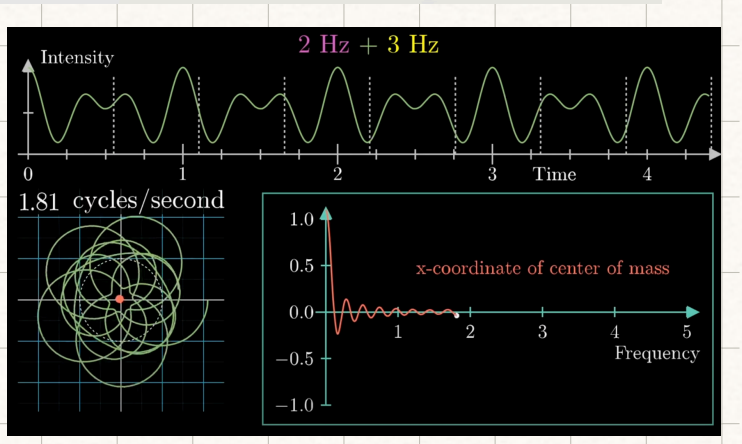
- NB this defines a **circulant** matrix **C**([*b*, *c*, 0, 0, ..., 0, *a*]) s.t. **H** = f(**X**) = **CX**

$$f(\mathbf{X}) = \begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix} \begin{bmatrix} — & \mathbf{x}_0 & — \\ — & \mathbf{x}_1 & — \\ & \vdots & \\ — & \mathbf{x}_{n-2} & — \\ — & \mathbf{x}_{n-1} & — \end{bmatrix}$$



Almost Fourier transform : signal wrapping around the circle with a given frequency

inverse Fourier transform : which function myst have produced that centre mass vs frequency graph(kinda answers that)



application : removing a high pitch in audio ; fourier transform will give the high pitch and we can smooth it out and then apply inverse fourier transform

GNNs and PGMs, more broadly

- Based on this idea, **structure2vec** (Dai *et al.*, ICML'16) embed mean-field inference within computations of a GNN.
  - Key difference: in PGMs, we expect potential functions specified and known upfront
  - Here, they are defined implicitly, within the latents of a GNN.
- The structure2vec GNN itself is not unlike a typical MPNN.
- Recently, there are other approaches that unify GNNs with PGM-like computations:
  - CRF-GNNs (Gao *et al.*, KDD'19)
  - GMNNs (Qu *et al.*, ICML'19)
  - ExpressGNN (Zhang *et al.*, ICLR'20)
  - Tail-GNNs (Spalević *et al.*, ICML'20 GRL+)

Which is best? Neither.

- There doesn't seem to be a clear single "winner" aggregator here...
- In fact, we prove in the PNA paper (Corso, Cavalleri *et al.*, NeurIPS'20) that **there isn't one**!

Theorem 1 (Number of aggregators needed). *In order to discriminate between multisets of size n whose underlying set is $\mathbb{R}$, at least n aggregators are needed.*

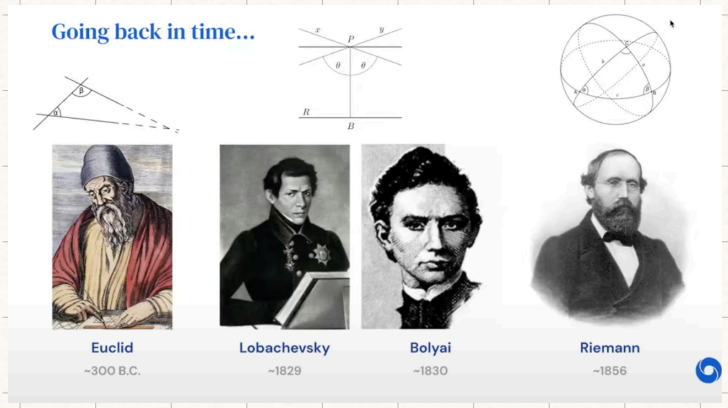
- The proof is (in my opinion) **really cool**! (relies on **Borsuk-Ulam** theorem)
- PNA proposes empirically powerful **combination** of aggregators for general-purpose GNNs:

$$\bigoplus = \underbrace{\begin{bmatrix} I \\ S(D, \alpha = 1) \\ S(D, \alpha = -1) \end{bmatrix}}_{\text{scalers}} \otimes \underbrace{\begin{bmatrix} \mu \\ \sigma \\ \max \\ \min \end{bmatrix}}_{\text{aggregators}}$$

video : https://youtu.be/aCUOAkOqNoU
slides : https://petar-v.com/talks/5G-CS224W.pdf

why to connect geometry and deep learning? what do we mean by geometry?

different types of geometries :



which is true geometry?

erlangen program : in group theory all these geometries can be grouped into a group based on their symmetries and invariances

other impacts:
1. physics : noether's theorem : all conservation laws can be derived from symmetry
2. category theory : theoretical computer science

GNN literature is very similar to the geometry in 1800s, what is the one true architecture?

learning in high dimensions is hard

1-Lipschitz

Even "simple" target functions (e.g. 1-Lipschitz) require exponential samples in nb. dimensions

A what is 1-Lipschitz in deep learning?

In deep learning, the concept of Lipschitz continuity is related to the smoothness and stability of functions. A function is said to be Lipschitz continuous if there exists a constant, known as the Lipschitz constant, that bounds the rate at which the function can change.

More specifically, a function f: X -> Y is said to be 1-Lipschitz if for any two points x1 and x2 in the input space X, the difference between the function values f(x1) and f(x2) is at most equal to the distance between x1 and x2, multiplied by 1. Mathematically, this can be expressed as:

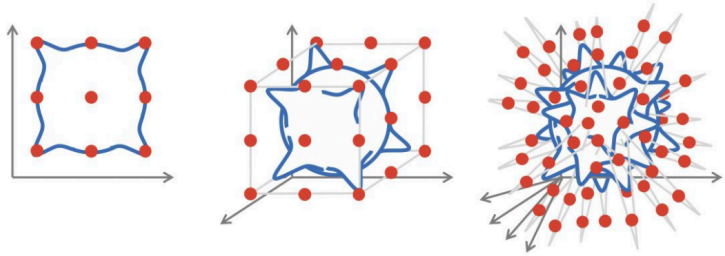
$$|f(x1) - f(x2)| \leq d(x1, x2)$$

where d(x1, x2) denotes the distance between x1 and x2.

In the context of deep learning, enforcing 1-Lipschitz continuity on a neural network can have certain benefits. One notable application is in the field of adversarial robustness, where the goal is to design models that are resistant to small perturbations in the input. By constraining the Lipschitz constant to be 1, it ensures that the function does not amplify or magnify small perturbations disproportionately, thereby providing some level of stability.

There are various techniques and approaches to enforce 1-Lipschitz continuity in deep learning models, such as Lipschitz regularization, weight clipping, or using specific activation functions like the Lipschitz-continuous activation functions. These techniques help control the Lipschitz constant and promote stability and robustness in the model's predictions.

The Curse of Dimensionality



Even "simple" target functions (e.g. 1-Lipschitz) require **exponential** samples in nb. dimensions.

is sigmoid a 1-lipschitz?

No, the sigmoid function is not 1-Lipschitz.

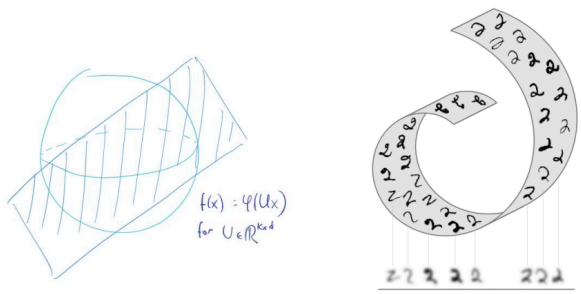
The sigmoid function is defined as f(x) = 1 / (1 + e^(-x)). It is a smooth, nonlinear function that maps the input values to a range between 0 and 1.

To determine if a function is Lipschitz, we need to find a constant that bounds the rate at which the function can change. In the case of the sigmoid function, it does not satisfy the Lipschitz condition with a Lipschitz constant of 1.

If we consider two points, x1 and x2, where x1 < x2, we can observe that the derivative of the sigmoid function is not bounded by a constant. In fact, the derivative of the sigmoid function reaches its maximum value of 0.25 at x = 0 and approaches 0 as x goes to positive or negative infinity. Therefore, there is no single constant that can be used to bound the rate of change for the sigmoid function across its entire domain.

Hence, the sigmoid function is not 1-Lipschitz.

Low-dimensional projections don't necessarily help!



Shallow MLPs can lose a lot of the **fidelity** of their inputs.
What can we do?

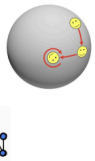
Curse of dimensionality : we require exponentially many training samples as we increase the number of dimensions we are learning in

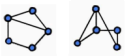
what is data manifold? - The data manifold refers to the underlying structure or shape of the data in its original high-dimensional space. It represents the intrinsic organization and relationships between data points. The manifold is a smooth, curved surface that captures the variations and configurations of the data.
Manifold learning algorithms(Isomap, t-SNE, or locally linear embedding) and dimensionality reduction methods (PCA) can be used to approximate this underlying structure by finding a lower-dimensional representation(in case of PCA) that preserves the essential characteristics of the data manifold.

Geometry to the rescue!

- We can inject further assumptions about the **geometry** of through *inductive biases*
  - Restrict the functions in our hypothesis space to ones that *respect* the geometry.
  - This can make the high-dimensional problem more tractable!

- Some popular examples:
  - **Image** data should be processed independently of **shifts**
  - **Spherical** data should be processed independently of **rotations**
  - **Graph** data should be processed independently of **isomorphism**





- We will now attempt to **formalise** this!

?

Principle 1: Symmetry groups
Principle 2: Scale separation

Symmetry is a transformation that leaves an object invariant (i.e. unchanged)
Hence they must be composable(AB=BA), invertible, contain identity…

they can be reasoned about using a very elegant mathematical object: the group.
Elements of these groups are domain transformations (e.g. some functions $g : \Omega \to \Omega$).

- We assume data lives on a ***domain***, Ω
  - e.g. for images, $u \in \Omega$ are *pixels*; for graphs they are *nodes*
- We assume a ***feature space***, $\mathbf{C}$, to be stored in elements of a domain
  - For our purposes, $\mathbf{C} = \mathbb{R}^k$
- We can then define *featurised domains* using a space of **feature functions** $\mathbf{X}(\Omega, \mathbf{C})$
  - $x \in \mathbf{X}(\Omega, \mathbf{C})$ is a function s.t. $x(u) \in \mathbf{C}$ gives features of element $u \in \Omega$
  - For discrete environments we can think of $\mathbf{X}$ as a *feature matrix* ($\mathbf{X} \in \mathbb{R}^{|\Omega| \times k}$)

Notation

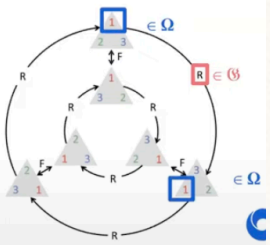
A *group* is a set $\mathfrak{G}$ along with a binary operation $\circ : \mathfrak{G} \times \mathfrak{G} \to \mathfrak{G}$ called *composition* (for brevity, denoted by juxtaposition $\mathfrak{g} \circ \mathfrak{h} = \mathfrak{g}\mathfrak{h}$) satisfying the following axioms:

Associativity: $(\mathfrak{g}\mathfrak{h})\mathfrak{k} = \mathfrak{g}(\mathfrak{h}\mathfrak{k})$ for all $\mathfrak{g}, \mathfrak{h}, \mathfrak{k} \in \mathfrak{G}$.

Identity: there exists a unique $\mathfrak{e} \in \mathfrak{G}$ satisfying $\mathfrak{e}\mathfrak{g} = \mathfrak{g}\mathfrak{e} = \mathfrak{g}$ for all $\mathfrak{g} \in \mathfrak{G}$.

Inverse: For each $\mathfrak{g} \in \mathfrak{G}$ there is a unique inverse $\mathfrak{g}^{-1} \in \mathfrak{G}$ such that $\mathfrak{g}\mathfrak{g}^{-1} = \mathfrak{g}^{-1}\mathfrak{g} = \mathfrak{e}$.

Closure: The group is closed under composition, i.e., for every $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$, we have $\mathfrak{g}\mathfrak{h} \in \mathfrak{G}$.



R : rotation ; F : flip

Group actions

- We are interested in how these groups affect data
  - Group action $(\mathfrak{g}, u) \mapsto \mathfrak{g}.u$ for a group element g, and a domain element u
  - E.g. translating or rotating an image, or permuting a set
- We will be interested in ***linear*** group actions: $\mathfrak{g}.(\alpha x + \beta x') = \alpha(\mathfrak{g}.x) + \beta(\mathfrak{g}.x')$
  N.F.
- This also allows us to represent group actions using **linear algebra**; $\rho : \mathfrak{G} \to \mathbb{R}^{n \times n}$

> A n-dimensional real *representation* of a group $\mathfrak{G}$ is a map $\rho : \mathfrak{G} \to \mathbb{R}^{n \times n}$, assigning to each $\mathfrak{g} \in \mathfrak{G}$ an *invertible* matrix $\rho(\mathfrak{g})$, and satisfying the condition $\rho(\mathfrak{g}\mathfrak{h}) = \rho(\mathfrak{g})\rho(\mathfrak{h})$ for all $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$. A representation is called *unitary* or *orthogonal* if the matrix $\rho(\mathfrak{g})$ is unitary or orthogonal for all $\mathfrak{g} \in \mathfrak{G}$.

Invariance and equivariance

- We can largely simplify **high-dimensional learning** by *exploiting* the symmetries in $\mathfrak{G}$!

> A function $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ is $\mathfrak{G}$-*invariant* if $f(\rho(\mathfrak{g})x) = f(x)$ for all $\mathfrak{g} \in \mathfrak{G}$, i.e., its output is unaffected by the group action on the input.

e.g. **image classification**: output class (*likely?*) won't depend on image **shifts**

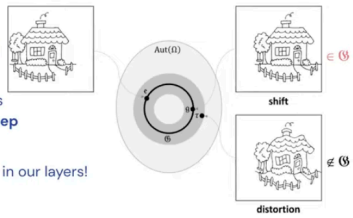
> A function $f : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega)$ is $\mathfrak{G}$-*equivariant* if $f(\rho(\mathfrak{g})x) = \rho(\mathfrak{g})f(x)$ for all $\mathfrak{g} \in \mathfrak{G}$, i.e., group action on the input affects the output in the same way.

e.g. **image segmentation**: segmentation mask must **follow** any shifts in the input

x(omega): mapping of domain ; Y : prediction on entire domain ; capital G : group; small g : element of G

Principle 2: Scale separation

- Want signal to be **stable** under slight *deformations* of the domain
- We derive: highly beneficial to compose **local** operations to model larger-scale ones
  - local ops won't globally propagate errors
  - e.g. CNNs with 3 x 3 kernels, but **very deep**
- Accordingly, we would like to support ***locality*** in our layers!
- cf. Fourier Transform vs. Wavelets



The key "building blocks" of Geometric Deep Learning

Linear $\mathfrak{G}$-*equivariant layer* $B : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega', \mathcal{C}')$, satisfying $B(\mathfrak{g}.x) = \mathfrak{g}.B(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$.

Nonlinearity $\sigma : \mathcal{C} \to \mathcal{C}'$ applied element-wise as $(\boldsymbol{\sigma}(x))(u) = \sigma(x(u))$.

Coarsening layer

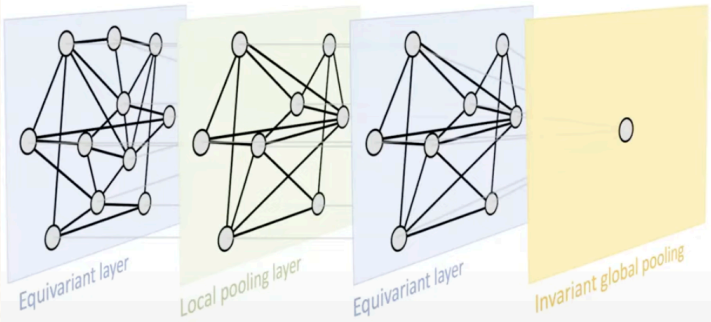
Local pooling (coarsening) $P : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega', \mathcal{C})$, such that $\Omega' \subseteq \Omega$.

(Not covered here in detail, but follows from scale separation!)

$\mathfrak{G}$-*invariant layer (global pooling)* $A : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{Y}$, satisfying $A(\mathfrak{g}.x) = A(x)$for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$.

If we want hierarchical data processing

All you need to build the architectures that are all you need :)



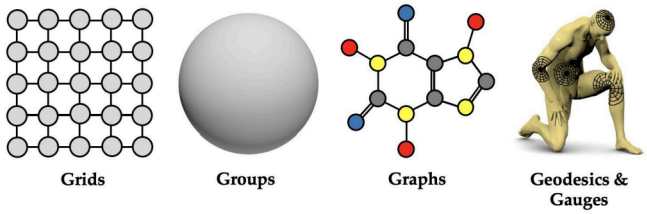
All the fan-favourites are easily derivable

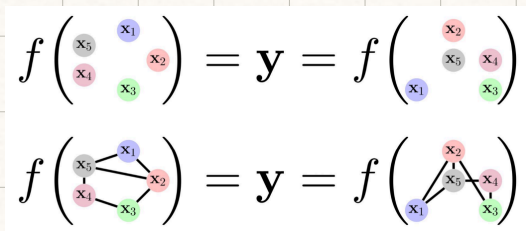
...with many (potentially) unexpected and useful **extras**! :)

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

The "5G" of Geometric Deep Learning

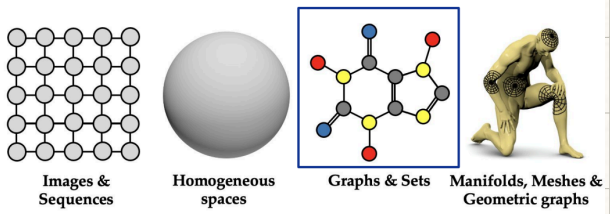
We will use the remainder of this lecture to study a few interesting **instances** of this blueprint





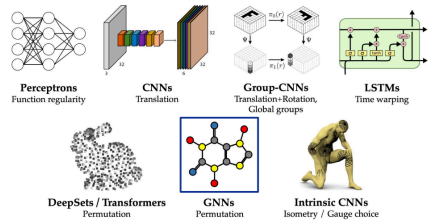
The "5G" of Geometric Deep Learning

We will use the remainder of this lecture to study a few interesting **instances** of this blueprint



Throughout CS224W, the focus was primarily on the domain of **graphs**.

Architectures of interest



References :

collaborative filtering :

- CF model for books RS : https://youtu.be/1YoD0fg3_EM (code : https://github.com/campusx-official/book-recommender-system/blob/master/book-recommender-system.ipynb)
- video : https://youtu.be/RVJV8VGa1ZY
- Collaborative and content based filtering video : https://youtu.be/v90un9ALRzw
- Neural Collaborative Filtering recommendation model : https://youtu.be/_uhU7s9kOEM

- GNN RS video : https://youtu.be/NyNqzDKcKG4
- article series : https://medium.com/data-from-the-trenches/graphical-neural-networks-link-prediction-part-ii-c60f6d97fd97
- code from articles : https://github.com/linafaik08/graph_neural_networks/tree/main
- GCMC: https://arxiv.org/pdf/1706.02263.pdf
- PinSAGE: https://arxiv.org/pdf/1806.01973.pdf
- GraphSAGE : https://arxiv.org/pdf/1706.02216.pdf
- video on above 2 papers : https://youtu.be/vinQCnizqDA , https://youtu.be/ed0NJdqwEyg
- Link prediction based on GNNs : https://arxiv.org/pdf/1802.09691.pdf
- Diffnet++: https://arxiv.org/pdf/2002.00844.pdf
- GraphRec: https://arxiv.org/abs/1902.07243
- Dynamic GNN for Seq. Rec: https://arxiv.org/pdf/2104.07368.pdf

- https://github.com/tsinghua-fib-lab/GNN-Recommender-Systems

papers :

GNN models :

- GAT : https://arxiv.org/pdf/1710.10903.pdf
- Heterogeneous Graph Attention Network Xiao Wang et al. : https://arxiv.org/pdf/1903.07293v1.pdf
- GCMC: https://arxiv.org/pdf/1706.02263.pdf
- GraphSAGE : https://arxiv.org/pdf/1706.02216.pdf
- PinSAGE: https://arxiv.org/pdf/1806.01973.pdf
- Multi-Component Graph Convolutional Collaborative Filtering : https://ojs.aaai.org/index.php/AAAI/article/view/6094
- Diffnet++: https://arxiv.org/pdf/2002.00844.pdf
- GraphRec: https://arxiv.org/abs/1902.07243
- Dynamic GNN for Seq. Rec: https://arxiv.org/pdf/2104.07368.pdf

- Link prediction based on GNNs : https://arxiv.org/pdf/1802.09691.pdf

survey papers :

- https://www.sciencedirect.com/science/article/pii/S2666651021000012?via%3Dihub
- Graph Neural Networks in Recommender Systems: A Survey(https://arxiv.org/pdf/2011.02260.pdf)
- A Comprehensive Survey on Graph Neural Networks(https://ieeexplore.ieee.org/abstract/document/9046288)

code :

- pytorch geometric : https://pytorch-geometric.readthedocs.io/en/latest/Cx2i8an--Tkt7O8Z , https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html
- GCN : https://github.com/pyg-team/pytorch_geometric/blob/master/examples/gcn.py
- GAT : https://github.com/gordicaleksa/pytorch-GAT/blob/main/models/definitions/GAT.py#L53 , https://github.com/PetarV-/GAT/blob/master/models/gat.py
-

articles :

- https://towardsdatascience.com/graph-neural-network-gnn-architectures-for-recommendation-systems-7b9dd0de0856
- https://medium.com/data-from-the-trenches/graphical-neural-networks-link-prediction-part-ii-c60f6d97fd97
- Transformers are Graph Neural Networks : https://thegradient.pub/transformers-are-graph-neural-networks/

video/playlist/book for learning :

- book : https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf?trk=public_post_comment-text
- GNN playlist : https://youtube.com/playlist?list=PLV8yxwGOxvvoNkzPf
- AMMI Geometric Deep Learning Course - First Edition (2021) : https://youtube.com/playlist?list=PLn2-dEmQeTfQ8YVuHBOvAhUlnIPYxkeu3
- ICLR keynote 2021(GNN) : https://www.youtube.com/watch?v=w6Pw4MOzMuo&t=334s
- Theoretical Foundations of Graph Neural Networks by Petar Veličković : https://youtu.be/uF53xsT7mjc
- graphsage in uber eats recommendations : https://www.uber.com/en-IN/blog/uber-eats-graph-learning/ , https://www.youtube.com/live/9O9osybNvyY?feature=share
- https://youtu.be/Qip-Cgv1cMU
- https://youtube.com/playlist?list=PLBoQnSflObckArGNhOcNg7IQG_f0ZlHF5
- ML with graphs playlist : https://youtube.com/playlist?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn

already complied resources :

- https://geometricdeeplearning.com/
- GNN recommender systems repo : https://github.com/tsinghua-fib-lab/GNN-Recommender-Systems
- https://github.com/thunlp/GNNPapers
- https://github.com/safe-graph/graph-fraud-detection-papers
- https://github.com/tsinghua-fib-lab/GNN-Recommender-Systems#Recommendation-Stages
- https://twitter.com/PetarV_93/status/1306689702020382720articles%20on%20GNNs%20:%20
- https://towardsdatascience.com/graph-deep-learning/home
- https://petar-v.com/GAT/