Graph Neural Networks - Introduction

week 17

Manon

Euclidean object vs. non euclidean objects

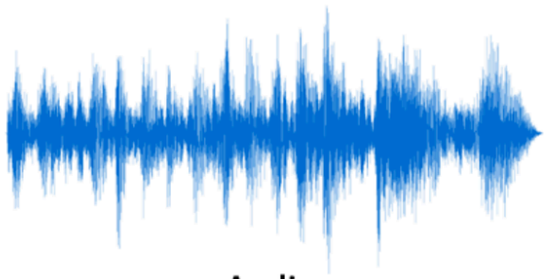
7

Numbers

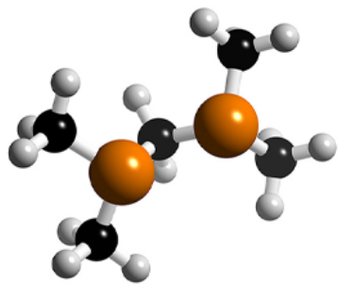
Images

The quick brown fox jumps over the lazy dog

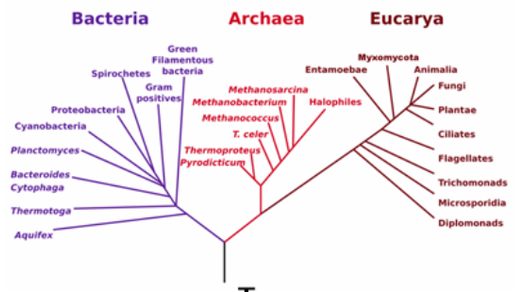
Text



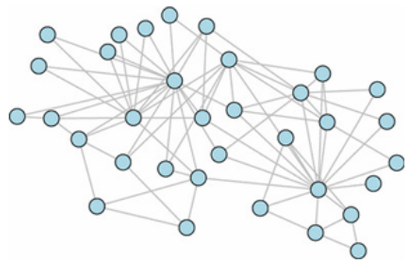
Audio



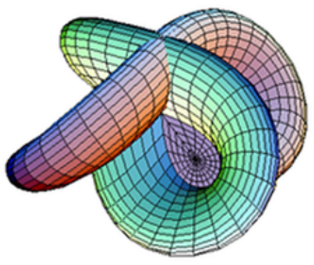
Molecules



Trees



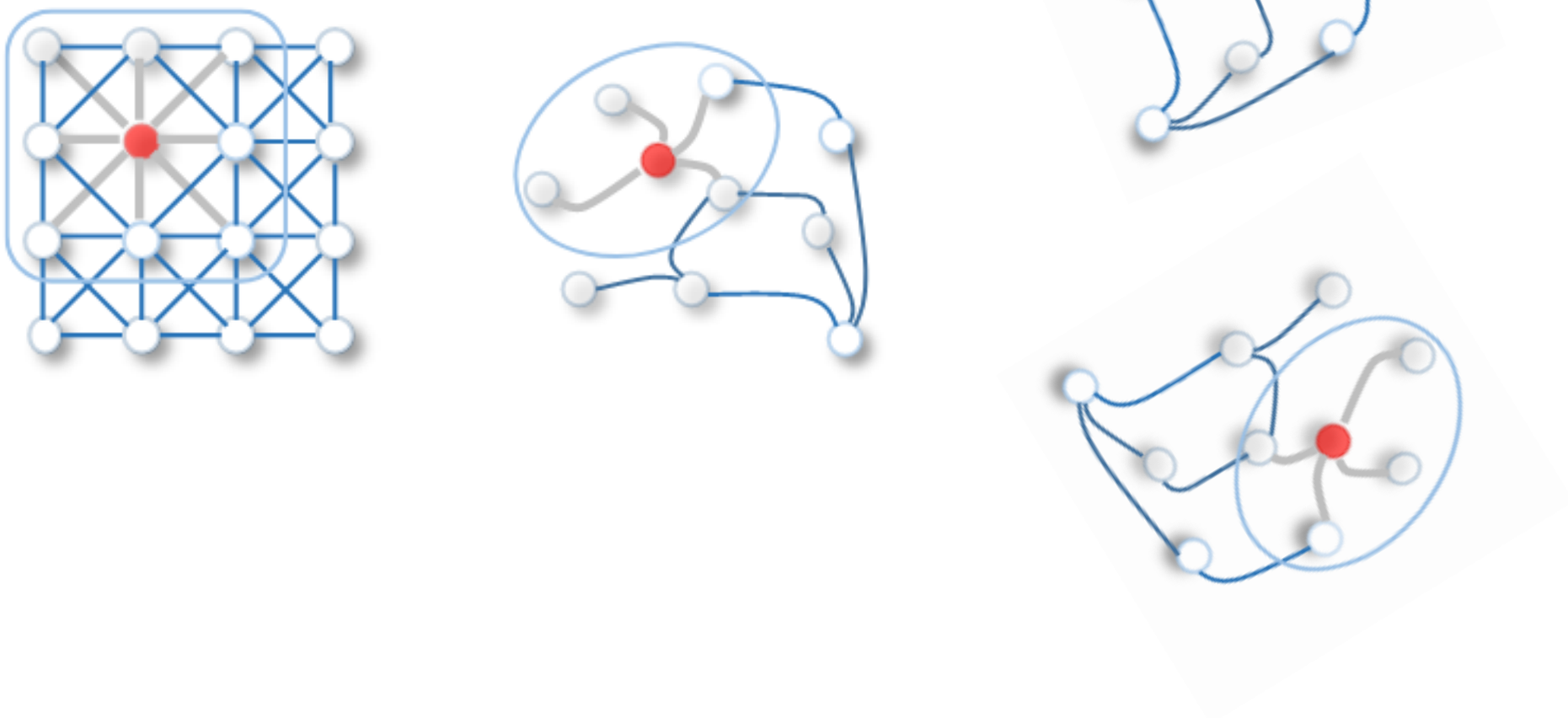
Networks



Manifolds

https://medium.com/@flawnsontong1/what-is-geometric-deep-learning-b2adb662d91d

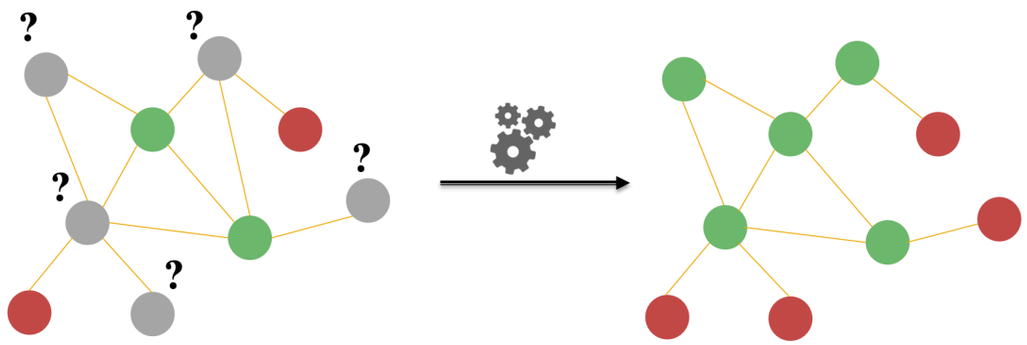
Regular convolution vs. spatial graph convolution



https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d
https://towardsdatascience.com/an-introduction-to-graph-neural-networks-e23dc7bdfba5

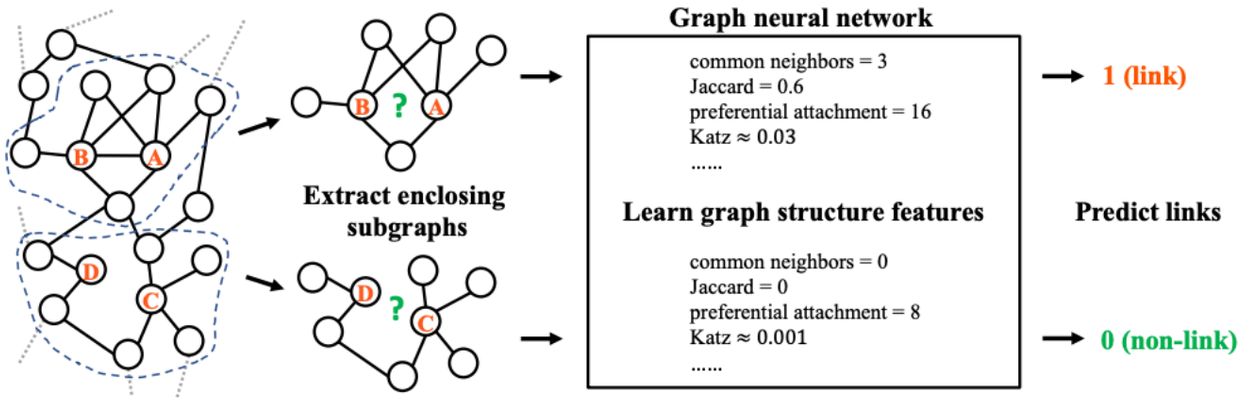
Different types of Graph Neural Networks

Node classification



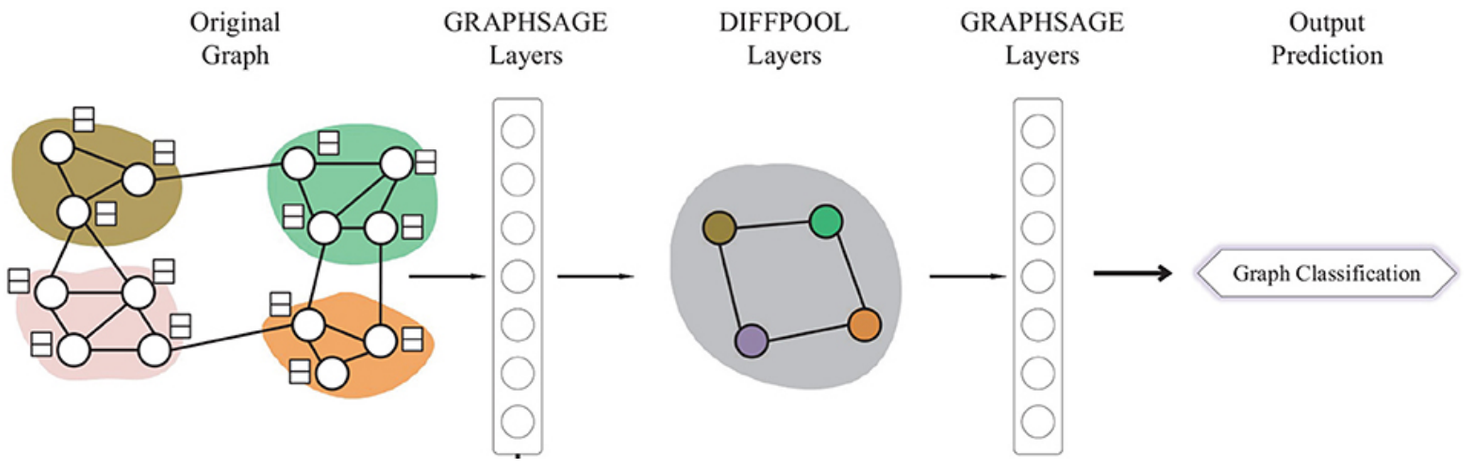
Edge classification

Zhang et al. NIPS 2018



Graph classification

Xie et al, Front. Neurosci., 2020



https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/message-passing-and-node-classification

First challenge : Graph representation

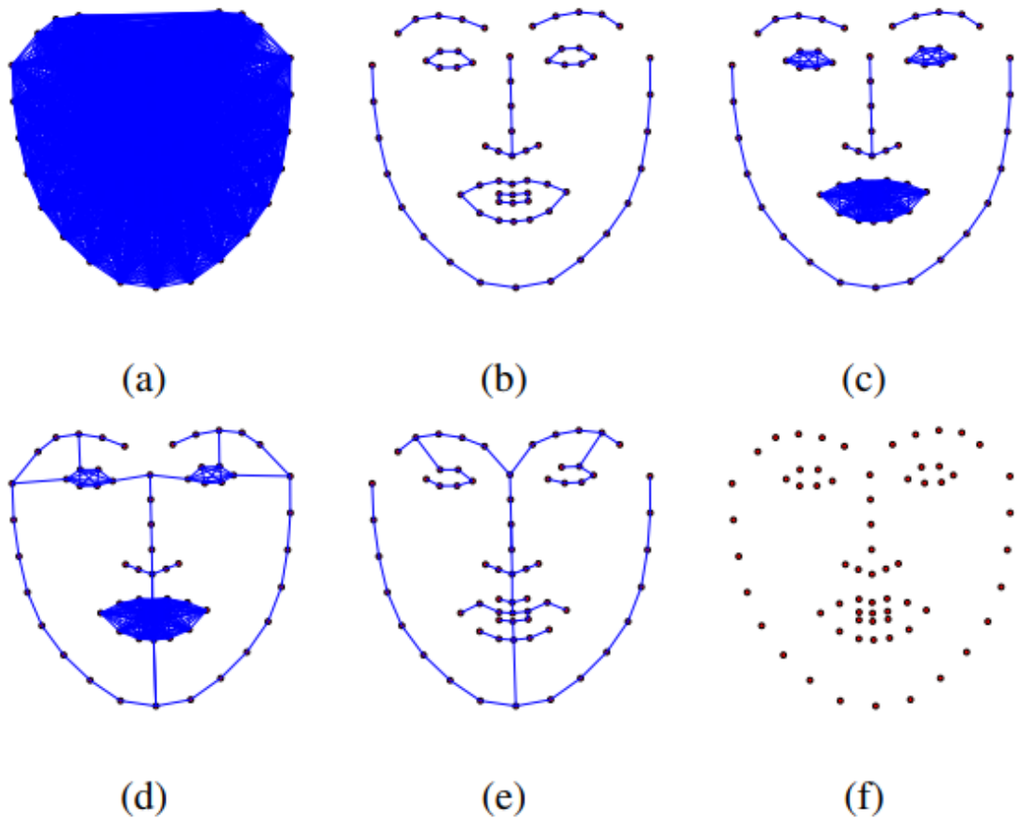
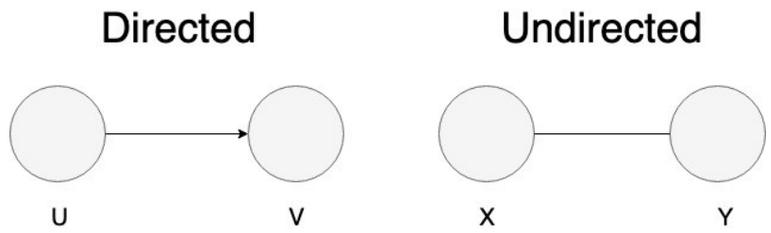
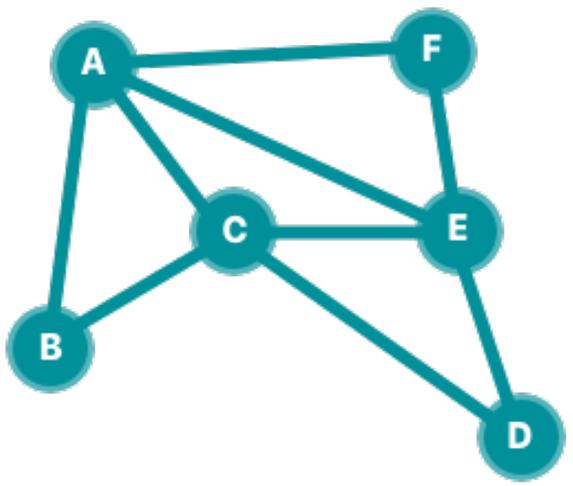


Figure 2: Employed graph structures. *(a)* Complete graph. *(b)* Chain per area. *(c)* Chain and complete per area. *(d)* Chain and complete per area with connections between them. *(e)* Minimum spanning tree. *(f)* Empty graph.



Matrix representation of graph connectivity

Undirected graph



V: {A,B,C,D,E,F}

E: {AB,AC,AF,BC,CD,CE,DE,EF}

=

Adjacency matrix

	A	B	C	D	E	F
A	0	1	1	0	1	1
B	1	0	1	0	0	0
C	1	1	0	1	1	0
D	0	0	1	0	1	0
E	1	0	1	1	0	0
F	1	0	0	0	1	0

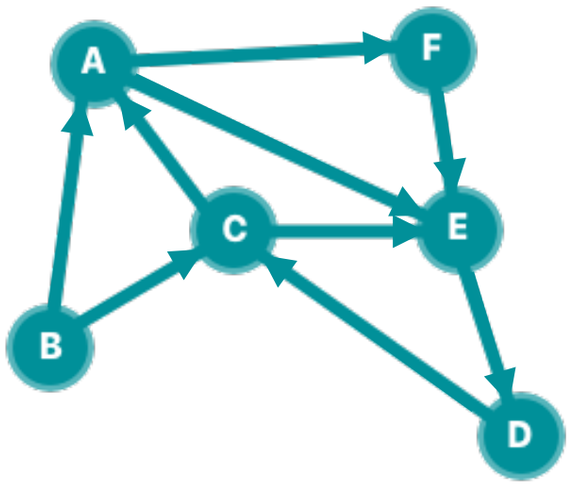
→

Degree Matrix

4
2
4
2
3
2

Matrix representation of graph connectivity

Directed graph

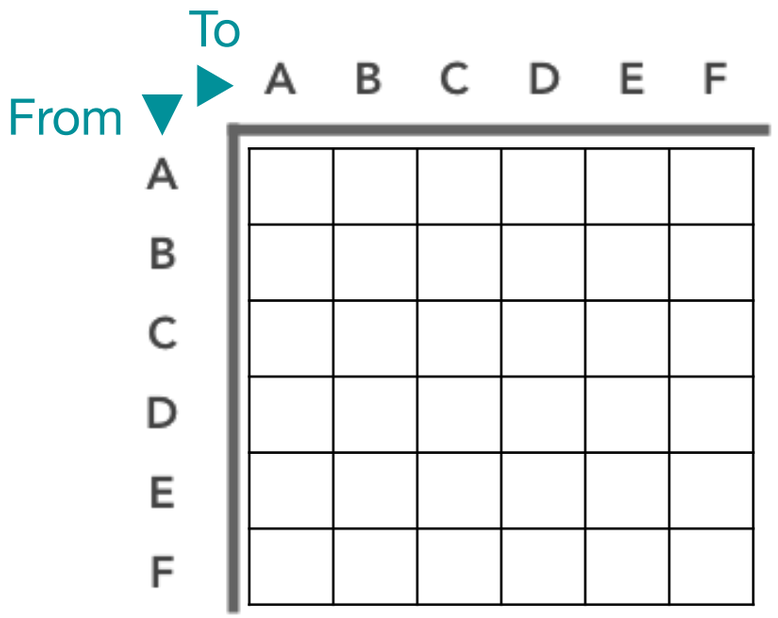


V: {A,B,C,D,E,F}

E: {AB,AC,AF,BC,CD,CE,DE,EF}

=

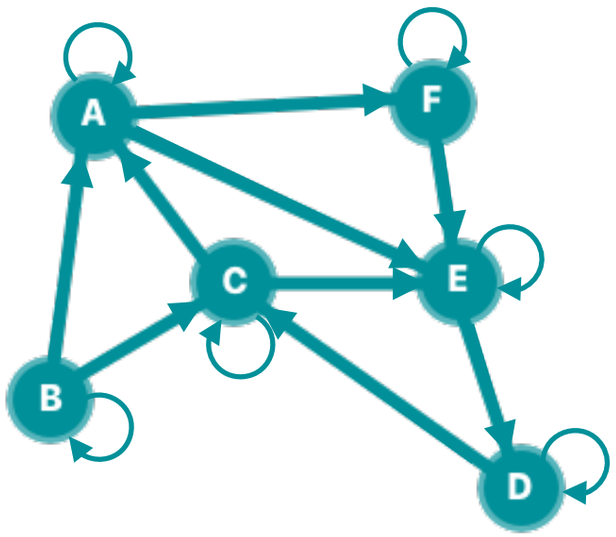
Adjacency matrix



Degree Matrix

Matrix representation of graph connectivity

Directed graph



V: {A,B,C,D,E,F}

E: {AB,AC,AF,BC,CD,CE,DE,EF}

=

Adjacency matrix

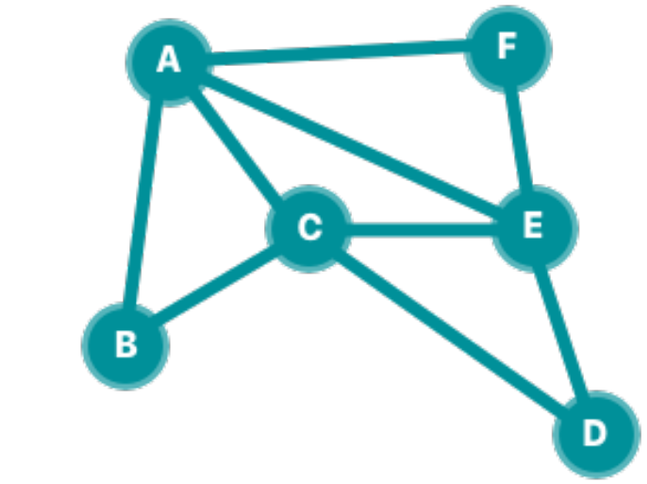
From \ To	A	B	C	D	E	F
A	1	0	0	0	1	1
B	1	1	1	0	0	0
C	1	0	1	0	1	0
D	0	0	1	1	0	0
E	0	0	0	1	1	0
F	0	0	0	0	1	1

→

Degree Matrix

3
3
3
2
2
2

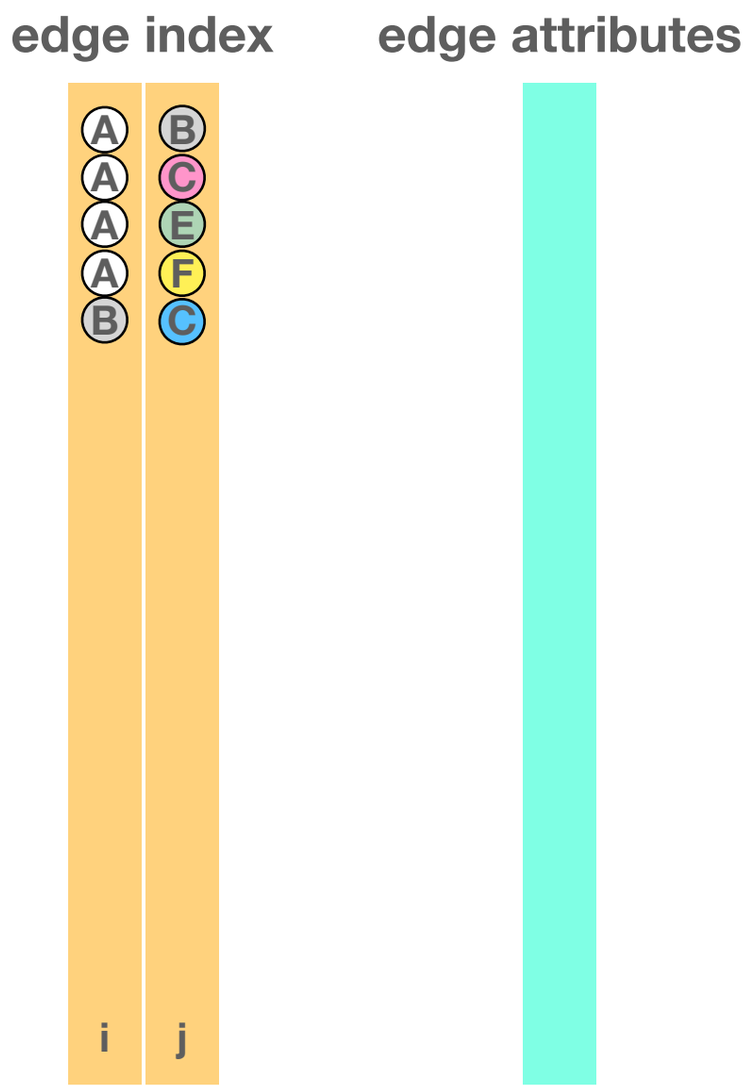
Other representation of graph connectivity



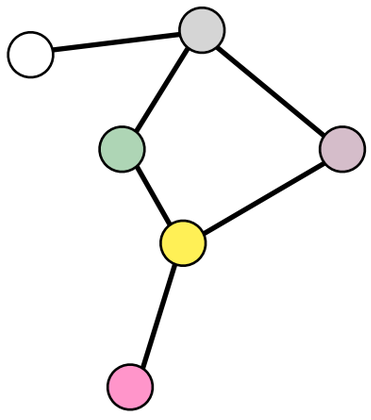
V: {A,B,C,D,E,F}

E: {AB,AC,AF,BC,CD,CE,DE,EF}

=



Classical representation of a graph



=

number of features

Number of nodes

node features

edge index

1 2 2 2 3

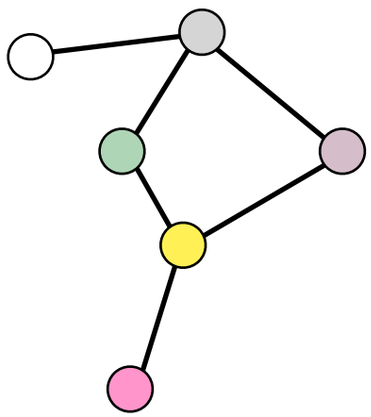
2 1 3 6 2

i j

edge attributes

distance
between i and j

Classical representation of a graph



=

number of features

Number of nodes

node features

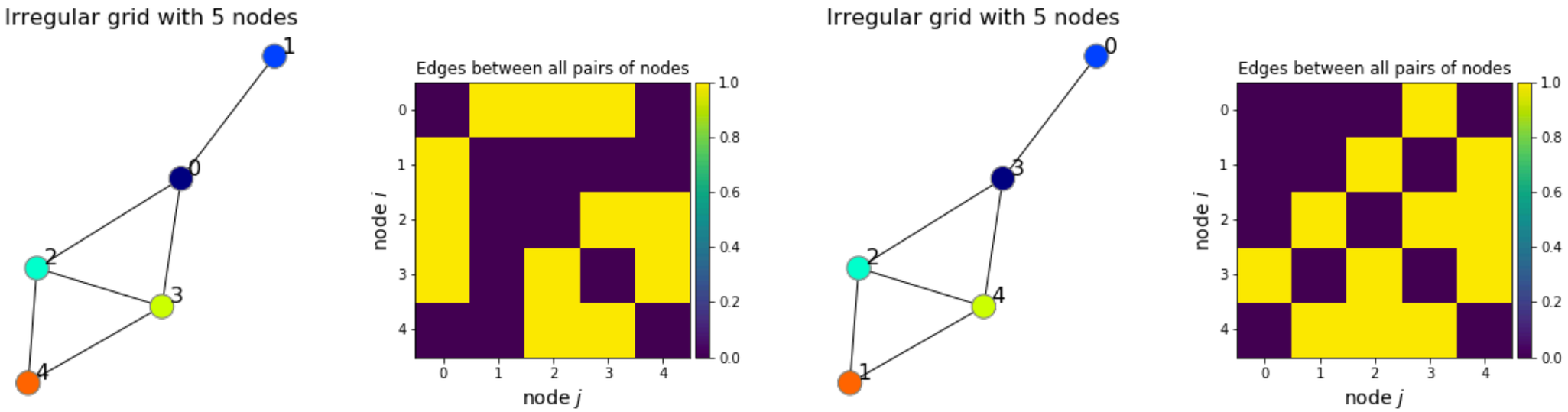
Adjacency matrix

○	1	1	0	0	0	0
○	1	1	1	1	0	0
○	0	1	1	0	1	0
○	0	1	0	1	1	0
○	0	0	1	1	1	1
○	0	0	0	0	1	1

(includes self loops)

How to make convolution on a graph ?

One key rule : the convolution HAS TO BE permutation invariant

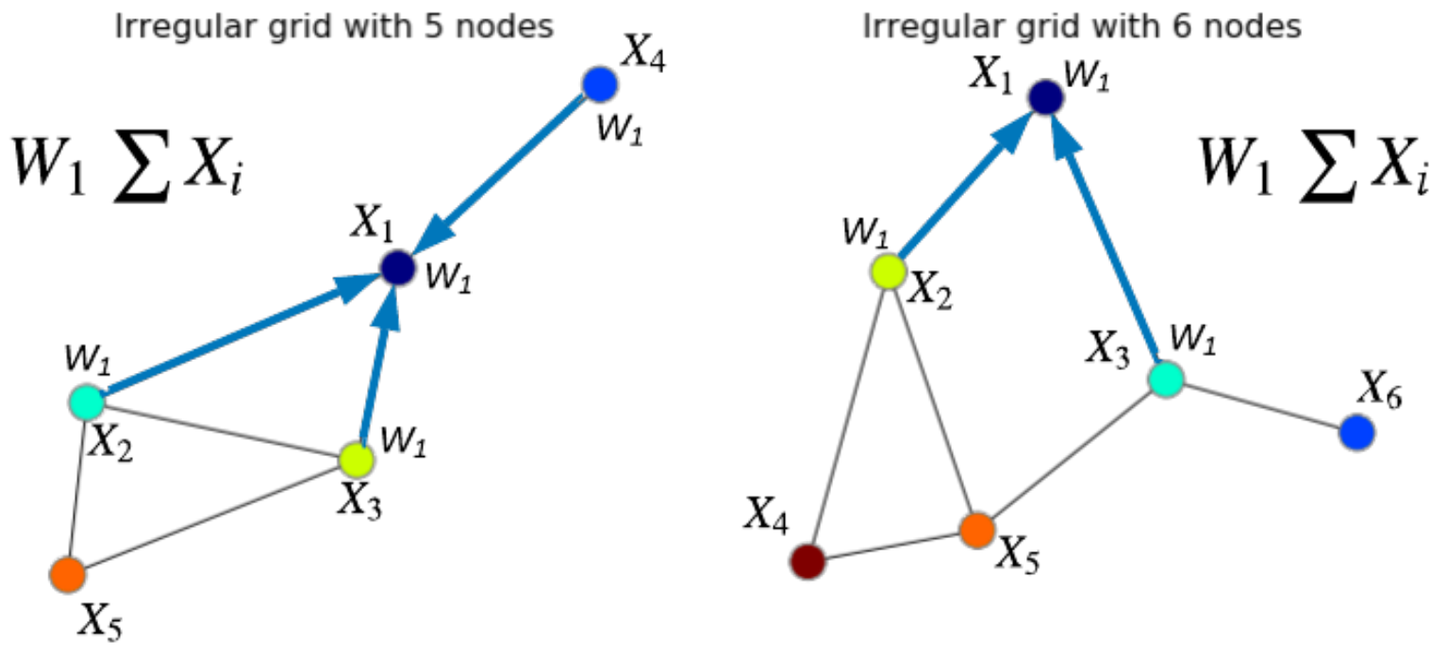


i.e. any nodes order must lead to the same result

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

How to make convolution on a graph ?

Graph convolutions make convolutions by aggregating a node features and it’s neighbors features into a new node

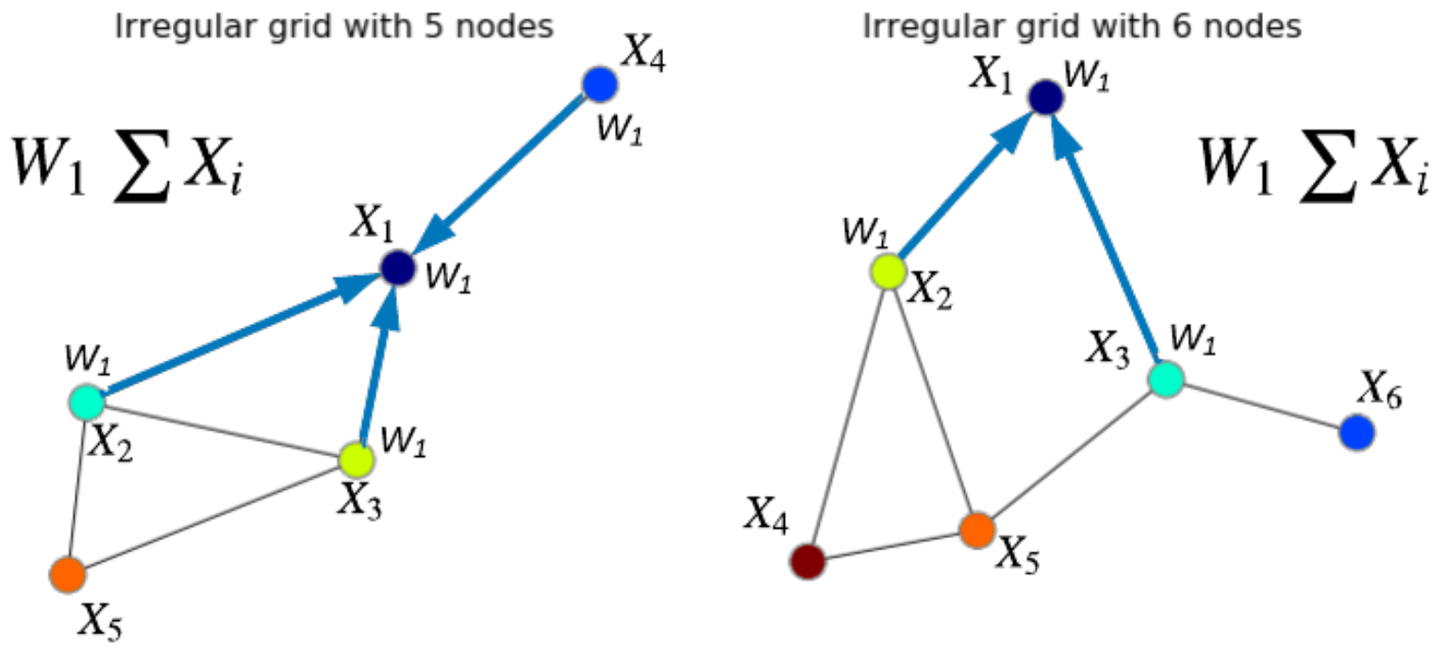


https://www.youtube.com/watch?v=cWleTMklzNg

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

How to make convolution on a graph ?

GNN :
neural **message passing** in which vector messages are exchanged between nodes and **updated** using neural networks [Gilmer et al., 2017]

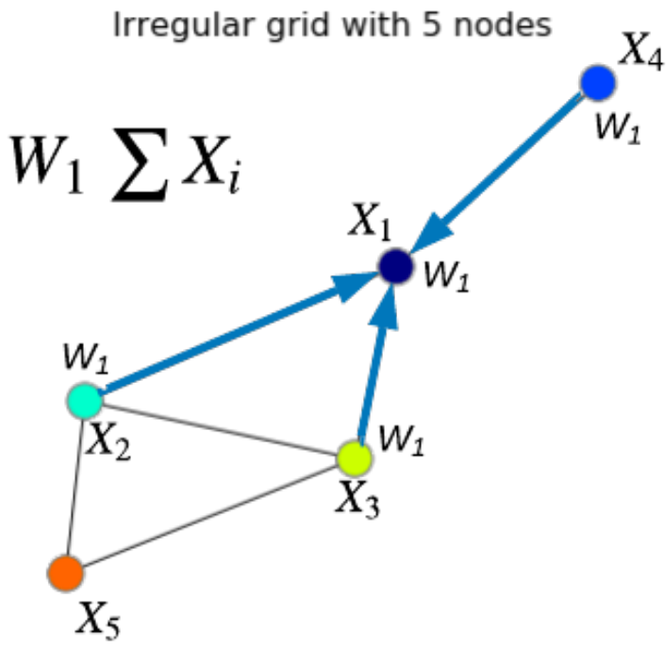


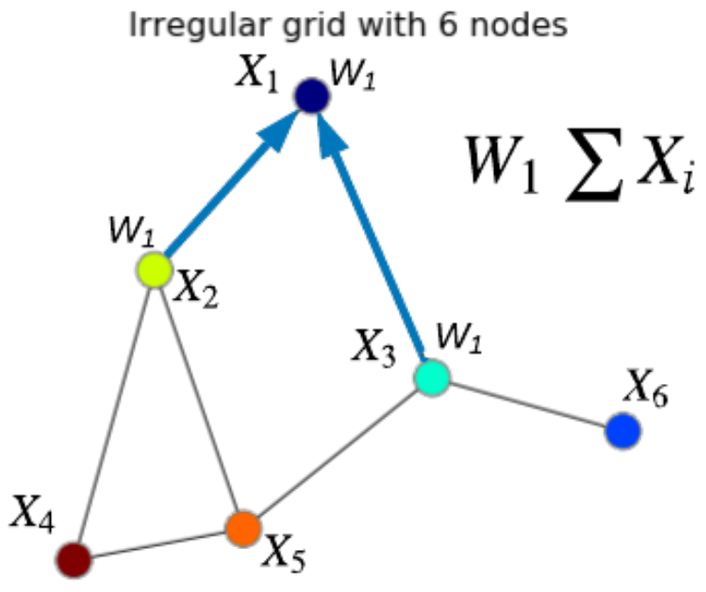
Same features share the **same weight** (this way, the learnable parameters do not depend on the graph size nor on the number of neighbors)

Different features have **different weights** (so that graph NN can try many different combinations of features)

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

How to make convolution on a graph ?





option 1. Aggregate neighbors info, then update the current node

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right)$$
$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right),$$

option 2. Aggregate current node + neighbor info at once

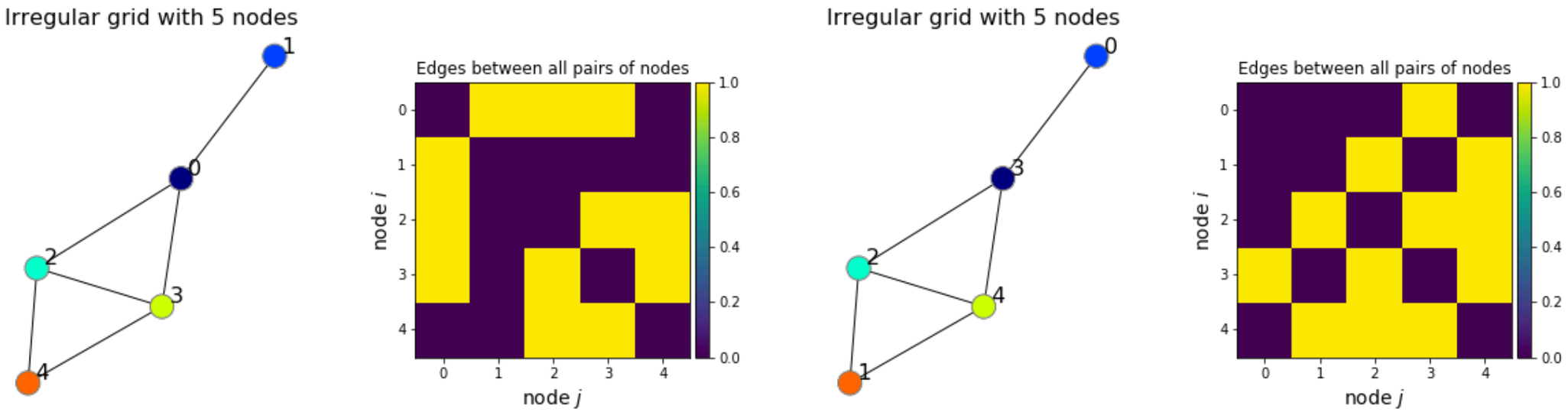
$$\mathbf{h}_u^{(k)} = \text{AGGREGATE}(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u) \cup \{u\}\}),$$

Message passing

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

How to make convolution on a graph ?

One key rule : the convolution HAS TO BE permutation invariant



i.e. any nodes order must lead to the same result

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

Use invariant aggregators

The most basic aggregator : **Sum**

/!\ The sum is very sensitive to node degree

One solution :
normalize the aggregation operation based upon the degrees of the nodes involved

Average

$$\mathbf{m}_{\mathcal{N}(u)} = \frac{\sum_{v \in \mathcal{N}(u)} \mathbf{h}_v}{|\mathcal{N}(u)|},$$

Symmetric normalization

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}.$$

Use invariant aggregators

Any remarks on those aggregators ?

Use invariant aggregators

Any remarks on those aggregators ?

1. They assign the same weight to the current node and its neighboring nodes

Can use linear interpolation

$$\text{UPDATE}_{\text{interpolate}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \boldsymbol{\alpha}_1 \circ \text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) + \boldsymbol{\alpha}_2 \odot \mathbf{h}_u,$$

with : $\boldsymbol{\alpha}_2 = \mathbf{1} - \boldsymbol{\alpha}_1$

Update functions can also be **recurrent unit, concatenation, Jumping Knowledge Connections** etc.

Use invariant aggregators

Any remarks on those aggregators ?

1. They assign the same weight to the current node and its neighboring nodes

Can use linear interpolation

$$\text{UPDATE}_{\text{interpolate}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \boldsymbol{\alpha}_1 \circ \text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) + \boldsymbol{\alpha}_2 \odot \mathbf{h}_u,$$

with : $\boldsymbol{\alpha}_2 = \mathbf{1} - \boldsymbol{\alpha}_1$

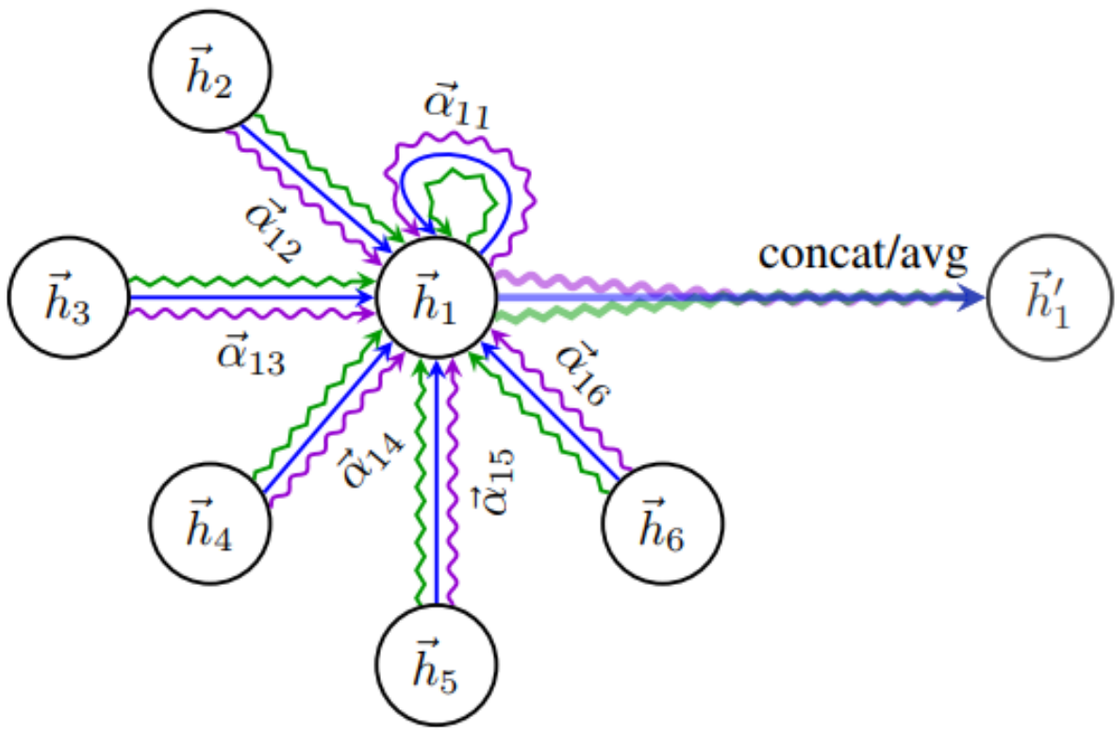
2. They assign the same weight to all neighbors

Use **attention heads** to weight the importance of the neighbor information

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v,$$

Attention heads - aggregators

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v,$$

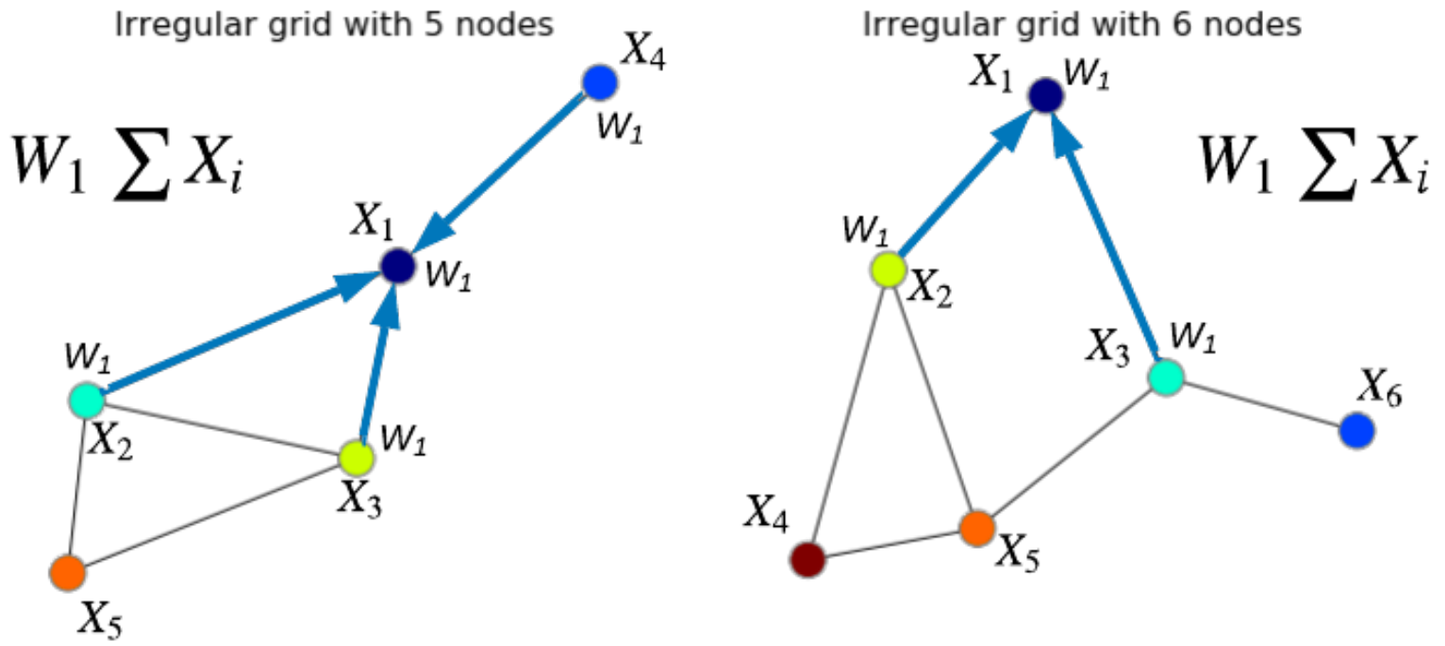


Attention heads can be :

- learnt
- deduced from neighbors information
- given as an input

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y. (2018) **Graph Attention Networks**.

Now we have updated node features, but still the graph arrangement



Useful for node classification

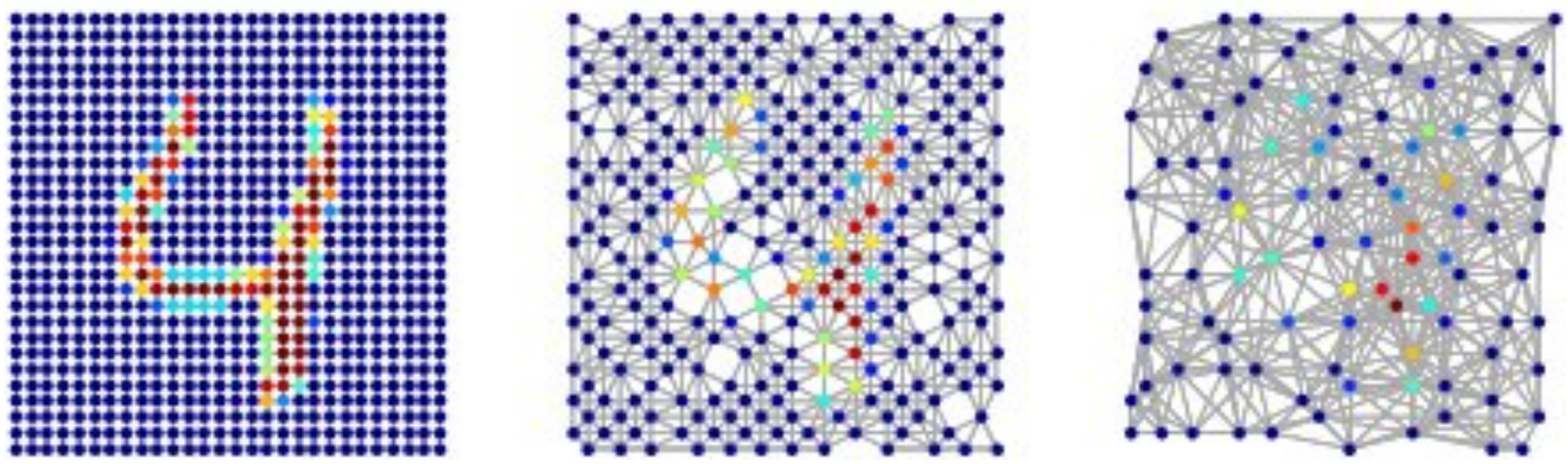
Useful for edge prediction

What about graph classification ?

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

Opt. reduce the data = Clustering

- **graph coarsening/clustering approach :**
  - Either based on clustering algorithm
  - Or based on pre-computed cluster (ex residue-level clustering)



It’s common to consider only the max value for each feature considering all cluster members
Many other possibilities

https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

Flatten the data to add fully connected layer(s)

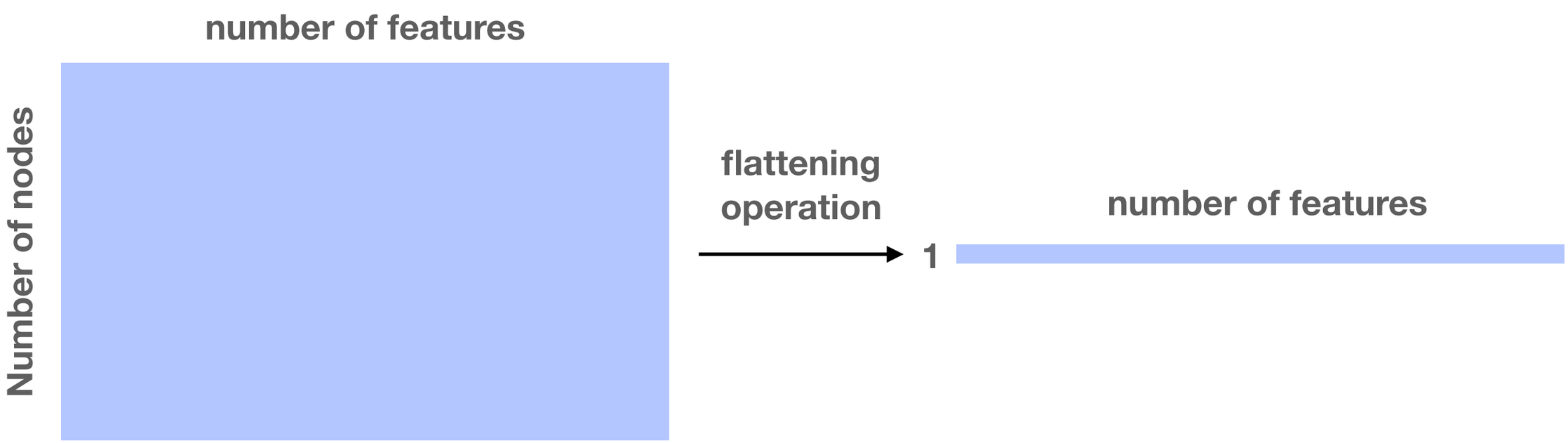
Any Idea ?

Flatten the data to add fully connected layer(s)

Any Idea ?

Sum, mean, max, min, median etc. on the feature dimension

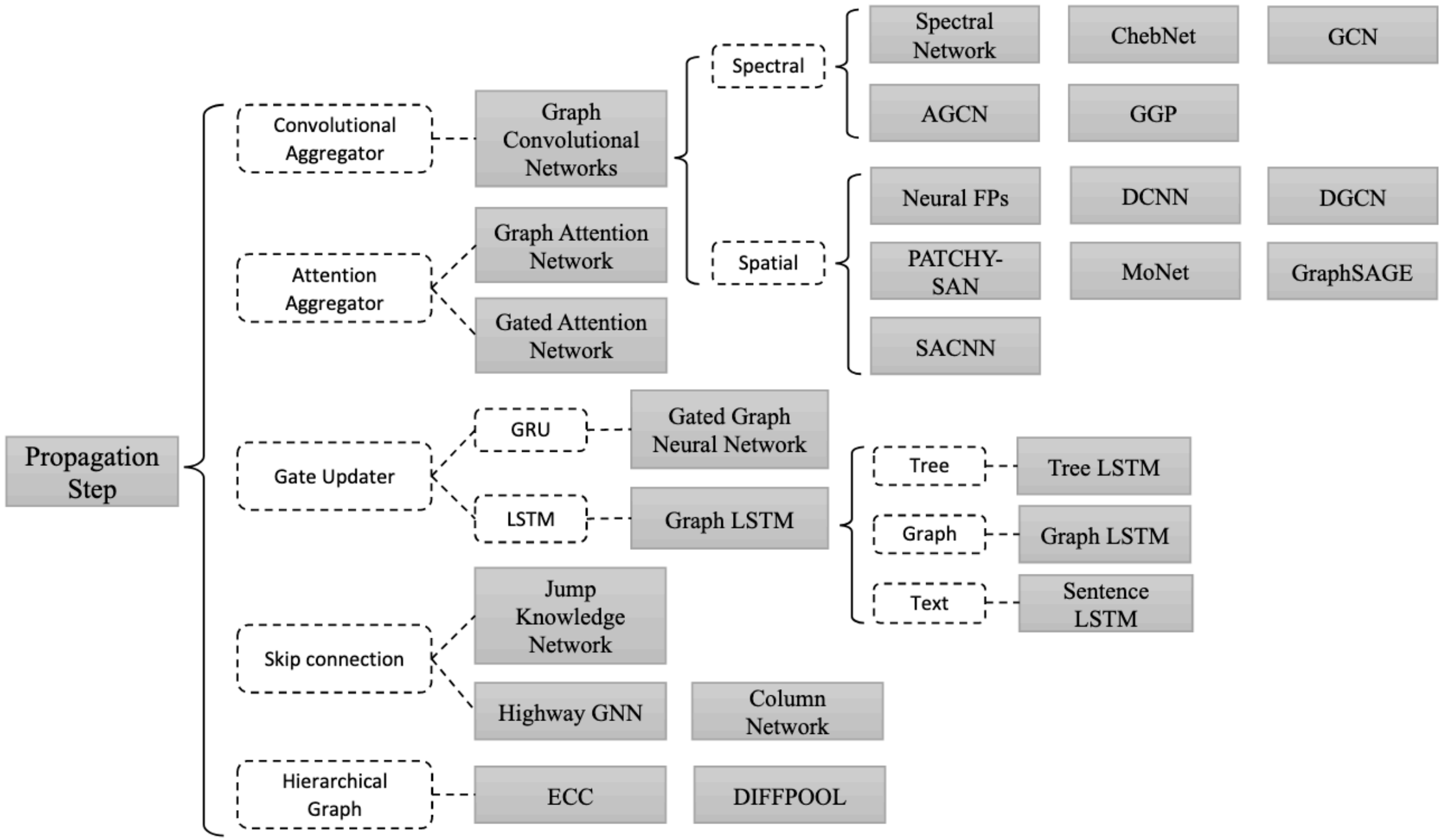
(Permutation invariant metric)



**Now you've extracted information,
you can learn from them and build a classifier with fully connected layers**

Summary

- A Message Passing function: M_t: $m^{l+1} = M_t(H^l, A)$
- A Node Update function: U_t: $H^{l+1} = U_t(H^l, m^{l+1})$
- A Readout function: R_t (for graph classification): $y = R(H^l)$



(c) Propagation Steps

Next week :

- Examples of entire Graph Neural Networks
- Focus on Graph Attention Network

In 2 weeks :

- Limits and unsolved GNN challenges