Neural Networks for Data Science Applications

Master's Degree in Data Science

Lecture 6: Graph neural networks

Lecturer: S. Scardapane

SAPIENZA
UNIVERSITÀ DI ROMA

Introduction

Use cases for graph neural networks

Graph neural networks

Graphs are ubiquitous in the real-world, and over the last years we have seen a great increase in research and industrial interest into developing neural networks designed for them.

Graphs share a number of characteristics with images (e.g., each node has neighbours), leading to some extensions of convolutional layers, but they also have several peculiar characteristics requiring careful solutions.

Example #1: Fake news on social networks

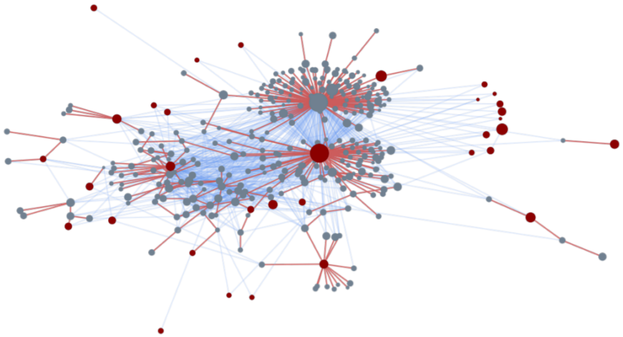


Figure 1: Example of a single news story spreading on a subset of the Twitter social network. Social connections between users are visualized as light-blue edges. A news URL is tweeted by multiple users (cascade roots denotes in red), each producing a cascade propagating over a subset of the social graph (red edges). Circle size represents the number of followers. Note that some cascades are small, containing only the root (the tweeting user) or just a few retweets.

Monti, F., Frasca, F., Eynard, D., Mannion, D. and Bronstein, M.M., 2019. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*.

Example #2: Traffic road prediction

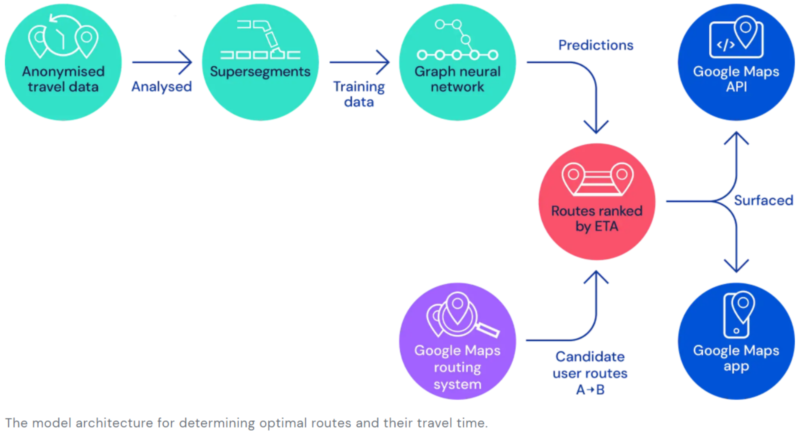
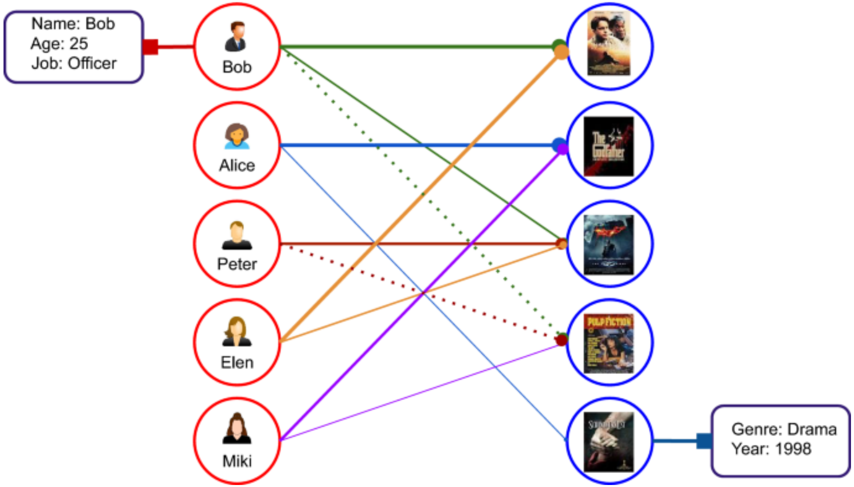


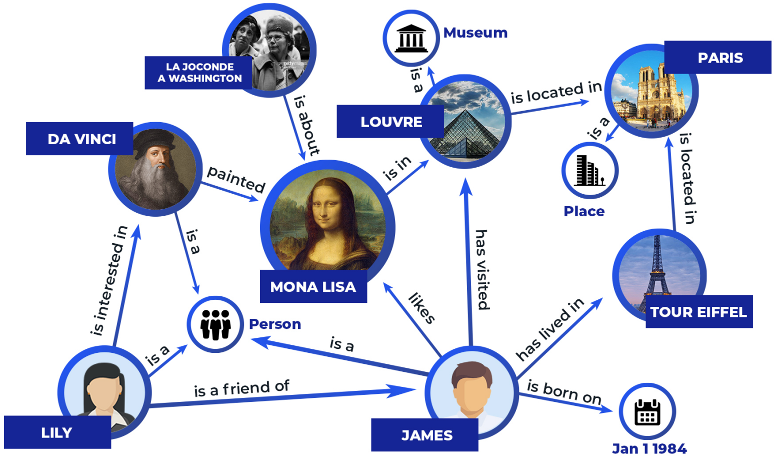
Figure 1: Traffic prediction with advanced Graph Neural Networks (DeepMind blog).

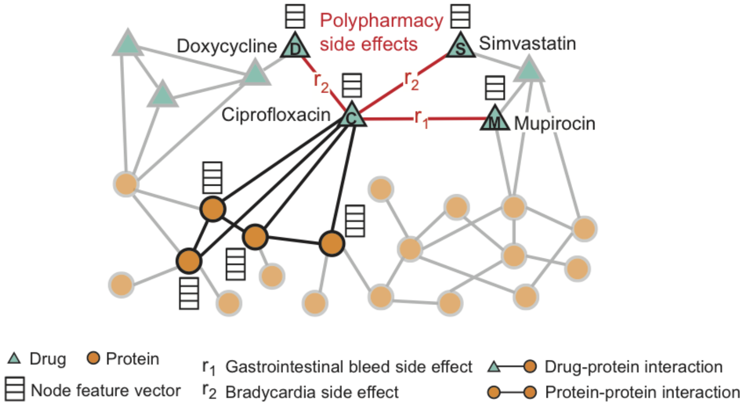
Example #3: Recommender systems



Tran, D.H., et al., 2021. **HeteGraph: graph learning in recommender systems via graph convolutional networks**. *Neural Computing and Applications*, pp.1-17.

Example #4: Knowledge graphs




Polypharmacy side effects
Doxycycline
Simvastatin
Ciprofloxacin
Mupirocin
D
S
C
M
r_2
r_2
r_1
Drug
Protein
Node feature vector
r_1 Gastrointestinal bleed side effect
r_2 Bradycardia side effect
Drug-protein interaction
Protein-protein interaction

Zitnik, M., Agrawal, M. and Leskovec, J., 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13), pp.i457-i466.

Introduction

Basic definitions

Definition of a graph

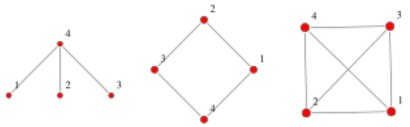
A **graph** is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, n\}$ is a set of n **vertices** (**nodes**), and $\mathcal{E} = \{(i,j) \mid i, j \in \mathcal{V}\}$ is a set of **edges** between them.

Alternatively, we can define an **adjacency matrix** $\underset{(n,n)}{\mathbf{A}}$ where:

$$A_{ij} = \begin{cases} W_{ij} & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} .$$

If $\mathbf{A}^\top = \mathbf{A}$, we say the graph is **undirected**. Note that we allow for a generic weight W_{ij} between two nodes, not necessarily 1.

Examples of (binary) adjacency matrix



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 2: Examples of adjacency matrices with 0/1 values (reproduced from Wolfram MathWorld).

Neighbours and degrees

The neighborhood $\mathcal{N}_i$ of a node i is defined as the set of nodes sharing an edge with the node:

$$\mathcal{N}_i = \{j \mid (i,j) \in \mathcal{E}\} \ .$$

The size $|\mathcal{N}_i|$ of the neighborhood is called the **degree** of the node. The **degree matrix** $\underset{(n,n)}{\mathbf{D}}$ is a diagonal matrix containing the degrees:

$$[\mathbf{D}]_{i,i} = \sum_j A_{ij} \ .$$

Learning problems on graphs

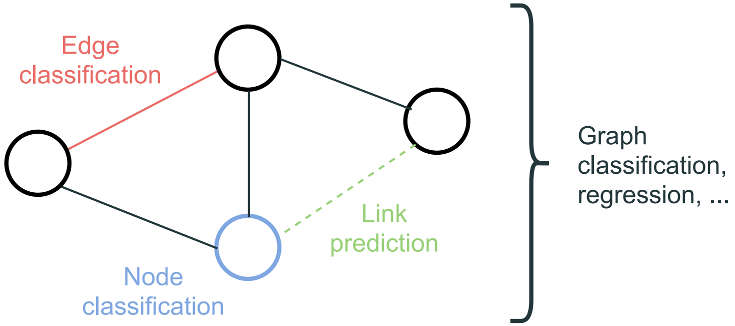


Figure 3: Several learning problems can be defined on a graph, depending on the entity of interest.

Features on a graph

Depending on the application, we can have **node features**, **edge features**, or **graph features**.

In the simplest case, we associate to each node i a vector $\underset{(d)}{\mathbf{x}_i}$ of features, from which we can build a matrix $\underset{(n,d)}{\mathbf{X}}$.

More in general, we might have edge features $\mathbf{e}_{ij}$ or graph features $\mathbf{g}$ (which we do not consider in this lecture).

Graph layers

Properties of a graph layer

Building a layer for graphs

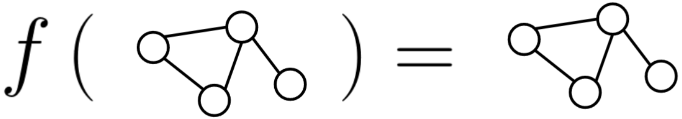


Figure 4: We want to build a neural network layer for graphs, i.e., a differentiable, composable, trainable operation for processing graphs.

Images and graphs

Like images, graphs have a notion of locality, embedded in the neighbour set $\mathcal{N}_i$.

Unlike images, however, two different nodes can have a different number of neighbours (a different degree).

In addition, we do not have any precise ordering over the neighbours, i.e., there is no ‘upper left’ node (a graph is not embedded in a metric space).

Locality in graph layers

We consider **graph layers** of the form $f(\mathbf{X}, \mathbf{A}) = (\mathbf{H}, \mathbf{A})$, acting on the node features, but keeping the connectivity the same. For simplicity, we can write $\mathbf{H} = f(\mathbf{X}, \mathbf{A})$, omitting the second output.

A graph layer is **local** if $[\mathbf{H}]_i$ only depends on $\mathcal{N}_i$.

Modifying the connectivity (e.g., **pooling**) is much harder than for images, and still an open problem in the research.

Grattarola, D., Zambon, D., Bianchi, F.M. and Alippi, C., 2021. **Understanding Pooling in Graph Neural Networks**. *arXiv preprint arXiv:2110.05292*.

Permutation matrices

Consider the 3×3 matrix defined as:

$$\mathsf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

It is easy to check that:

$$\mathsf{P} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_3 \\ x_2 \end{pmatrix}.$$

These are called **permutation matrices**.

Permutation equivariance

If we reorder the nodes in the graph by a permutation matrix $\mathbf{P}$, the effect on the different matrices is:

$$\mathbf{X}' = \mathbf{P}\mathbf{X}\,, \quad \mathbf{A}' = \mathbf{P}\mathbf{A}\mathbf{P}^\top\,. \tag{1}$$

A graph layer $f(\mathbf{X}, \mathbf{A})$ is **permutation equivariant** if:

$$f(\mathbf{X}', \mathbf{A}') = \mathbf{P} \cdot f(\mathbf{X}, \mathbf{A})\,. \tag{2}$$

If the property is not respected, then the predictions in the layer are dependent on the specific ordering we imposed on the nodes.

Graph layers

Graph convolutional layers

A convolution-like operation for graphs

A **graph convolutional (GC) layer** is defined as:

$$[\mathbf{H}]_i = \phi\left(\sum_{j \in \mathcal{N}_i} A_{ij}\mathbf{W}^\top \mathbf{x}_j\right) \,,$$

which can be written compactly as:

$$\mathbf{H} = \phi\left(\mathbf{AXW}\right) \,.$$

The graph is local (in the sense defined above) and permutation equivariant.

Expanding on the GC layer

The GC layer can be understood as a sequence of three operations:

1. A local **node-wise** operation $\widehat{\mathbf{x}}_i = \mathbf{W}^\top \mathbf{x}_i$.
2. An aggregation with respect to the neighborhood. This is called the **message-passing** phase.
3. A standard non-linearity.

In an image convolution, points (1)-(2) are combined in the filtering operation, because each pixel has a fixed neighborhood. Here, this is not possible because we cannot easily initialize trainable parameters to weight the neighborhood.

Normalized propagation matrices

The GC layer remains valid if we replace the adjacency matrix with another matrix having the same sparsity pattern (i.e., 0 for pairs of nodes not connected in the graph).

- **Adjacency matrix with self-loops**: $\mathbf{A} + \mathbf{I}$.
- **Normalized adjacency matrix** (with or without self-loops): $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

Some of these matrices can have better properties when training.

Kipf, T.N. and Welling, M., 2016. **Semi-supervised classification with graph convolutional networks**. *arXiv preprint arXiv:1609.02907*.

Visualization of the graph convolution

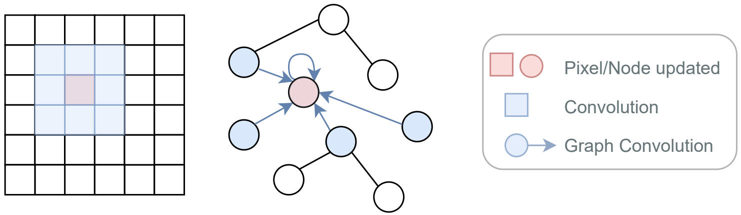


Figure 5: Visualization of convolutions on images and graphs (taken from `https://spindro.github.io/post/gnn/`).

Graph layers

Stacking graph layers

Building a graph neural network

The GC layer is easily composable into a multi-layered architecture, e.g., with two layers:

$$f(\mathsf{X},\mathsf{A}) = \phi\left(\mathsf{A}\phi\left(\mathsf{AXW}\right)\mathsf{Z}\right) \; .$$

Similarly to a CNN, the **receptive field** for a node increases with the number of layers. For example, the output for node *i* for the network above depends on its neighbours *and their neighbours*.

Stacking GC layers gives rise to a **graph convolutional network** (GCN).

Visualizing a two-layer GCN

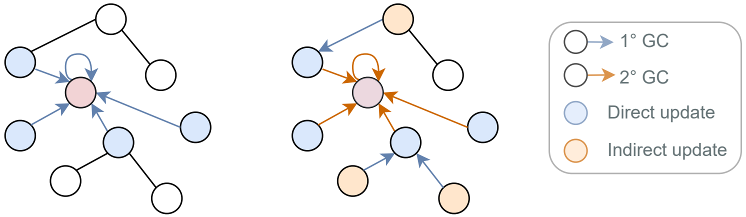


Figure 6: Visual example of a two-layered GCN (taken from `https://spindro.github.io/post/gnn/`).

Polynomial GCNs

We can also increase the receptive field of a single GC layer (similar to increasing the size of a kernel in a CNN).

Denoting by $\mathcal{N}_i^2$ the set of neighbours at distance 2 from node i, define a new order-2 layer as:

$$f(\mathbf{X}, \mathbf{A}) = \phi(\mathbf{X}\mathbf{W} + \overbrace{\mathbf{A}\mathbf{X}\mathbf{V}}^{\text{Order 1}} + \underbrace{\mathbf{A}^2\mathbf{X}\mathbf{Z}}_{\text{Order 2}}) \,, \tag{3}$$

where we have also separated the contribution of each node as $\mathbf{X}\mathbf{W}$.

Defferrard, M., Bresson, X. and Vandergheynst, P., 2016. **Convolutional neural networks on graphs with fast localized spectral filtering**. *Advances in Neural Information Processing Systems*, 29, pp.3844-3852.

Task #1: Node classification

Suppose a subset $\mathcal{T} \subset \mathcal{N}$ of nodes has a known class y (e.g., fake or certified users in a social network).

We can use a GCN to classify all the nodes of the graph simultaneously:

$$\underset{(n,c)}{\widehat{\mathbf{Y}}} = \text{softmax}\left(\mathbf{A}\phi\left(\mathbf{AXW}\right)\mathbf{Z}\right).$$

We optimize the weights of the network with gradient descent:

$$\mathbf{W}^*, \mathbf{Z}^* = \arg\min \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \text{cross-entropy}(y_i, [\widehat{\mathbf{Y}}]_i).$$

Kipf, T.N. and Welling, M., 2016. **Semi-supervised classification with graph convolutional networks**. *arXiv preprint arXiv:1609.02907.*

Task #2: Edge classification

In edge classification, we observe instead a subset of links $\mathcal{T} \subset \mathcal{E}$ with a given class y_{ij} (e.g., rated products in a recommender graph).

We can obtain a prediction for an edge by combining the features of the corresponding nodes:

$$\widehat{y}_{ij} = g\left(\mathbf{H}_i, \mathbf{H}_j\right) \,, \tag{4}$$

where g is an additional block (e.g., a fully-connected layer with a softmax applied on the concatenation of $\mathbf{H}_i$ and $\mathbf{H}_j$). Training proceeds similarly:

$$\mathbf{W}^*, \mathbf{Z}^* = \arg\min \frac{1}{|\mathcal{T}|} \sum_{(i,j)\in\mathcal{T}} \text{cross-entropy}(y_{ij}, \widehat{y}_{ij}) \,.$$

For **binary** classification, $g(\mathbf{h}_i, \mathbf{h}_j) = \mathbf{h}_i^\top \mathbf{h}_j$ is also common.

Task #3: Graph classification

Finally, assume we have multiple graphs $\mathcal{G}_i$, each with a target class y_i (e.g., a property of a certain molecule). Given the output $\mathbf{H}_i = \text{GCN}(\mathbf{X}_i, \mathbf{A}_i)$ for the ith graph, we can obtain an embedding for the entire graph by performing a global pooling operation:

$$\mathbf{h}_i = \frac{1}{n} \sum_j \left[\mathbf{H}_i\right]_j \,.$$

Mini-batching in this case is also easier, as shown next.

Batching multiple graphs

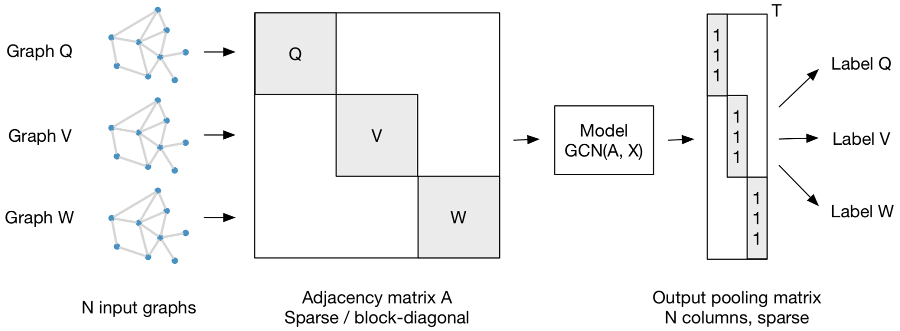


Figure 7: Batching multiple graphs in a single GCN call (taken from `https://github.com/tkipf/gcn`).

Graph layers

Graph attention networks

Anisotropic graph neural networks

The GCN treats the weights of the message-passing operation as *fixed*, depending only on the adjacency matrix or some scaled version of it.

We now show an important class of GNNs allowing for *learnable* message-passing coefficients, the **graph attention network** (GAT). This is an example of **anisotropic** GNN.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. and Bengio, Y., 2017. **Graph attention networks**. *arXiv preprint arXiv:1710.10903*.

Generalizing the message-passing operation

Let us consider a more general form of a message-passing neural network:

$$[\mathbf{H}]_i = \phi\left(\sum_{j \in \mathcal{N}_i} \eta(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)\right) \ , \tag{5}$$

where we recover the GC layer with $\eta(\mathbf{x}_i, \mathbf{x}_j) = A_{ij}$ and $g(\mathbf{x}) = \mathbf{W}^\top \mathbf{x}$. Basically, we are decomposing the layer as **building a message** for each neighbour, then aggregating the messages by **message weighting** and summation.

Generalizing the message-passing operation (2)

The previous formulation leads to several interesting extensions, e.g.:

1. If we have *edge features* we can add them to the message weighting function as $\eta(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ij})$, e.g., through concatenation.
2. If each node has a time-series instead of a vector (e.g., a grid of energy producers) we can replace the two functions with proper 1D-CNN models (or similar networks).

The graph attention layer

In a GAT layer, we first learn a set of *unnormalized* values a_{ij}, one for each edge, called the **attention scores**:

$$a_{ij} = \mathbf{a}^\top \mathrm{LeakyReLU}(\mathbf{V}\left[\mathbf{x}_i \parallel \mathbf{x}_j\right]) \,. \tag{6}$$

We then normalize the attention scores *for every node* to avoid instabilities:

$$\alpha_{ij} = \mathrm{softmax}(a_{i0}, \ldots, a_{iN}) \,.$$

$g(\mathbf{x}_j)$ remains the same as the standard GCN.

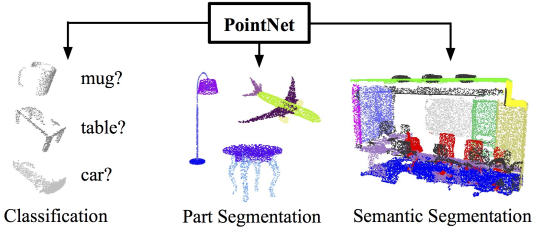
This is the so-called GATv2, see: Brody, S., Alon, U. and Yahav, E., 2021. **How Attentive are Graph Attention Networks?**. *arXiv preprint arXiv:2105.14491*.

Advanced concepts

Point cloud networks and equivariant message passing

Point cloud networks

An important subset of graphs is given by **point clouds**: a point cloud is a collection of points which, differently from standard graphs, have an associated **coordinate vector** $\mathbf{c}_i$ (e.g., the 3D position).



Qi, C.R., et al., 2017. **Pointnet: Deep learning on point sets for 3d classification and segmentation.** In *IEEE CVPR* (pp. 652-660).

Graphs with coordinates

In a more general setup, we have a coordinate matrix $\mathbf{C}$ and a feature matrix $\mathbf{X}$ (e.g., an energy grid where every station has a position and a time series).

We want the graph layer $f(\mathbf{X}, \mathbf{C}, \mathbf{A})$ to be equivariant to **permutations**, but at the same time it must be equivariant to **translations** and **rotations** on the coordinates:

$$f(\mathbf{X}, \mathbf{C}\mathbf{Q} + \mathbf{s}, \mathbf{A}) = \mathbf{Q}f(\mathbf{X}, \mathbf{C}, \mathbf{A}) + \mathbf{s}\,, \tag{7}$$

where $\mathbf{Q}$ is a generic rotation matrix and $\mathbf{s}$ a shift.

Rotation equivariance

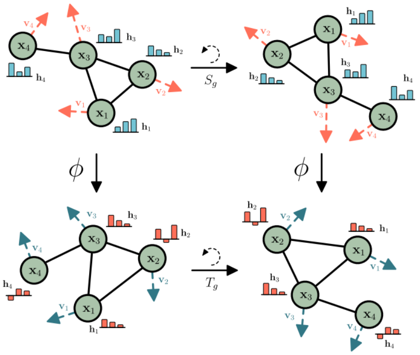


Figure 1. Example of rotation equivariance on a graph with a graph neural network ϕ

E(n) equivariant GNN

How can we incorporate all of this in the previous framework? The key insight is that rotations and translations do not modify the distance $\|\mathbf{C}_i - \mathbf{C}_j\|^2$. Thus, we modify our message passing as follows:

$$[\mathbf{X}]_i = \phi \left(\sum_{j \in \mathcal{N}_i} \underbrace{\eta(\mathbf{x}_i, \mathbf{x}_j, \|\mathbf{C}_i - \mathbf{C}_j\|^2) g(\mathbf{x}_j)}_{\mathbf{m}_{ij}} \right) , \tag{8}$$

We can also let the network adapt the coordinates layer by layer as:

$$\mathbf{C}_i = \mathbf{C}_i + \sum_{j \in \mathcal{N}_i} (\mathbf{C}_i - \mathbf{C}_j) \phi_c(\mathbf{m}_{ij}) \, . \tag{9}$$

Satorras, V.G., Hoogeboom, E. and Welling, M., 2021. **E(n) equivariant graph neural networks**. In *ICML* (pp. 9323-9332). PMLR.

Advanced concepts

Latent graph imputation

Differentiable graph module

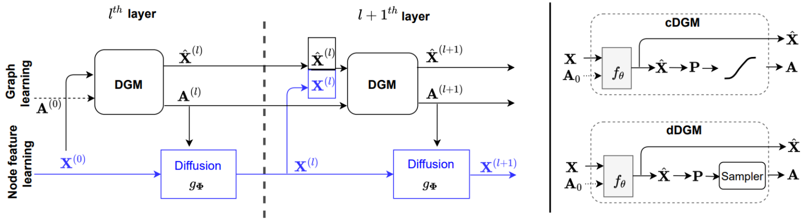


Fig. 1: *Left:* Two-layered architecture including Differentiable Graph Module (DGM) that learns the graph, and Diffusion Module that uses the graph convolutional filters. *Right:* Details of DGM in its two variants, cDGM and dDGM.

Kazi, A., Cosmo, L., Ahmadi, S.A., Navab, N. and Bronstein, M., 2022. **Differentiable graph module (dgm) for graph convolutional networks**. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Software libraries

Multiple libraries build on standard deep learning frameworks to provide GNN functionalities:

- Deep Graph Library (framework agnostic, very scalable): `https://www.dgl.ai/`.
- PyTorch Geometric (great for reproducing results): `https://github.com/pyg-team/pytorch_geometric`.
- Spektral (TensorFlow, smaller than the other two): `https://graphneural.network/`.
- TensorFlow Graph Neural Networks: `https://github.com/tensorflow/gnn` (early release, poor documentation).

Further reading

- **Graph Representation Learning Book**: `https://www.cs.mcgill.ca/~wlh/grl_book/`.
- A paper that describes a more general GNN with node, edge, and graph features: Relational inductive biases, deep learning, and graph networks.
- Introductory blog post on Distill: `https://distill.pub/2021/gnn-intro/`.