



# 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

**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

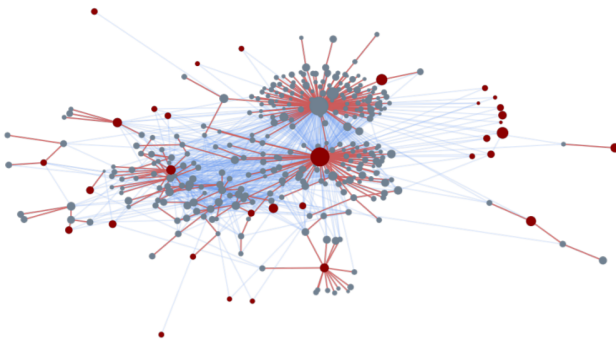
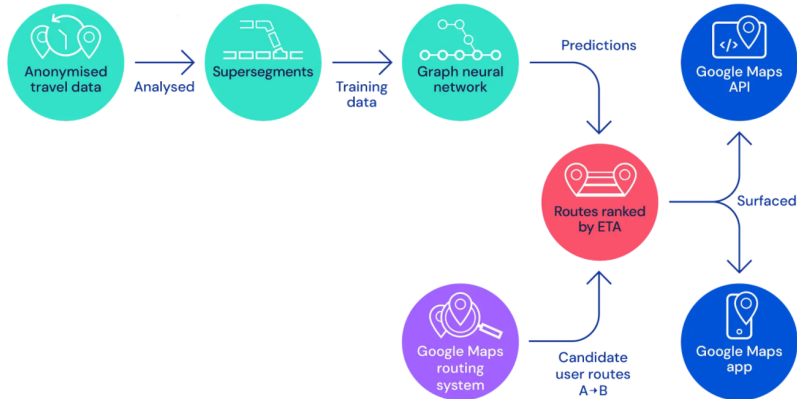


Figure 1: Example of a single news story spreading on a subset of the Twitter social network. Social connections between users are visualized as light-blue edges. A news URL is tweeted by multiple users (cascade roots denotes in red), each producing a cascade propagating over a subset of the social graph (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.

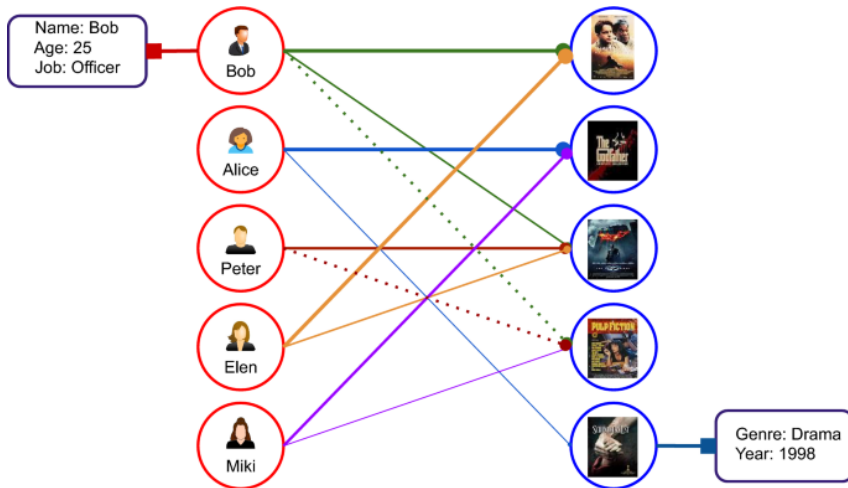
## Example #2: Traffic road prediction



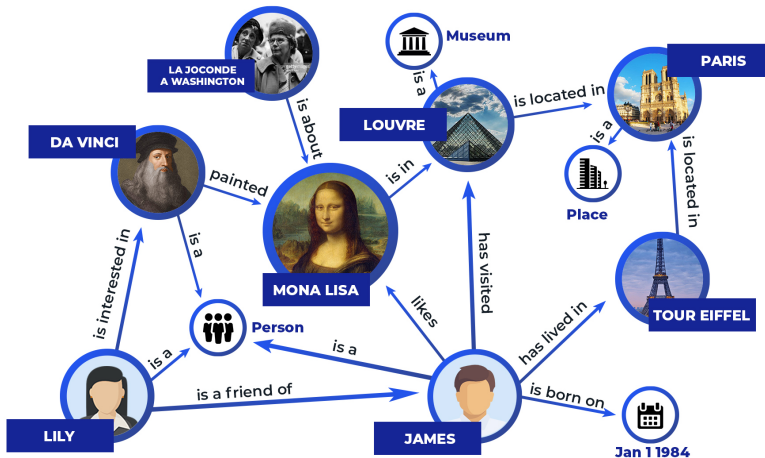
The model architecture for determining optimal routes and their travel time.

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

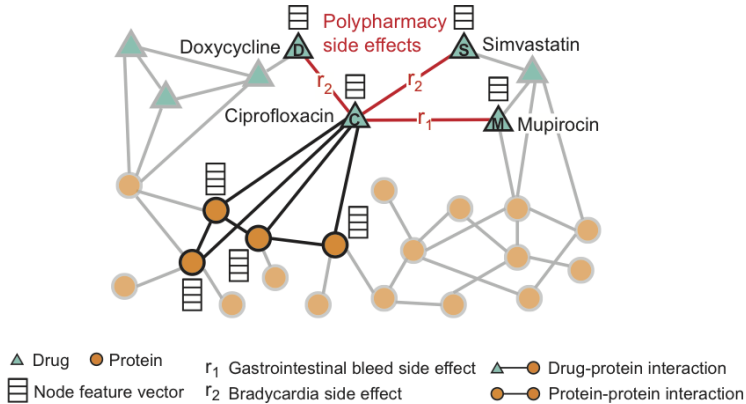
## Example #3: Recommender systems



## Example #4: Knowledge graphs



## Example #5: Drug 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

A **graph** is a pair  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V} = \{1, \dots, n\}$  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**  $\mathbf{A}_{(n,n)}$  where:

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

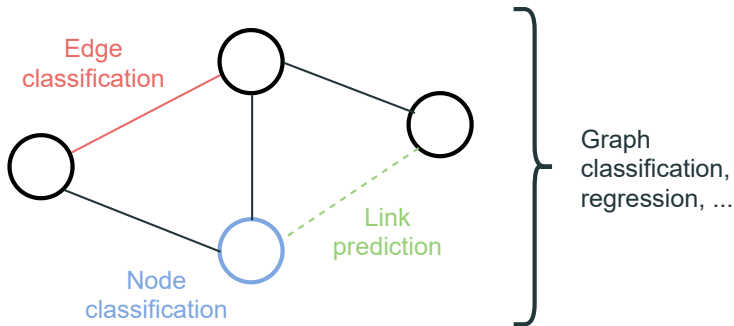
If  $\mathbf{A}^\top = \mathbf{A}$ , we say the graph is **undirected**. We consider undirected graphs for simplicity, but it is possible to extend everything to directed graphs.

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

$$\mathcal{N}_i = \{j \mid A_{ij} = 1\} .$$

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

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



**Figure 2:** Several learning problems can be defined on a graph, depending on the entity of interest.

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  $\mathbf{x}_i$  of features, from which we can build a matrix  $\mathbf{X}$ .

$(d)$

$(n, d)$

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

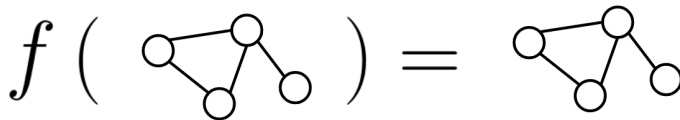


Figure 3: We want to build a neural network layer for graphs, i.e., a **differentiable**, **composable**, **trainable** operation for processing graphs.

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**).



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*.

Consider the  $3 \times 3$  matrix defined as:

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

It is easy to check that:

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

These are called **permutation matrices**.

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

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

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

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

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 **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:

$$\mathbf{H} = \phi (\mathbf{A}\mathbf{X}\mathbf{W}) .$$

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

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

1. A local **node-wise** operation  $\hat{\mathbf{x}}_i = \mathbf{W}^\top \mathbf{x}_i$ .
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.

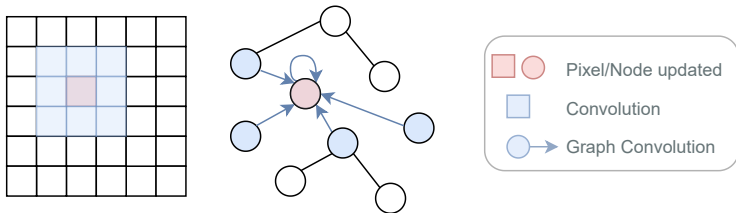
The GC layer remains valid if we replace the adjacency matrix with another matrix having the same sparsity pattern (i.e., 0 for pairs of nodes not connected in the graph).

- ▶ **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*.



**Figure 4:** Visualization of convolutions on images and graphs (taken from <https://spindro.github.io/post/gnn/>).



# Graph layers

---

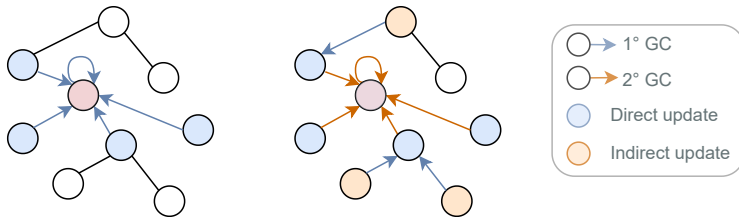
Stacking graph layers

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

$$f(\mathbf{X}, \mathbf{A}) = \phi(\mathbf{A} \phi(\mathbf{A} \mathbf{X} \mathbf{W}) \mathbf{Z}) .$$

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).



**Figure 5:** Visual example of a two-layered GCN (taken from <https://spindro.github.io/post/gnn/>).

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:

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

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.

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:

$$\hat{\mathbf{Y}}_{(n,c)} = \text{softmax}(\mathbf{A}\phi(\mathbf{A}\mathbf{X}\mathbf{W})\mathbf{Z}).$$

We optimize the weights of the network with gradient descent:

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

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:

$$\hat{y}_{ij} = g(\mathbf{H}_i, \mathbf{H}_j) , \quad (4)$$

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:

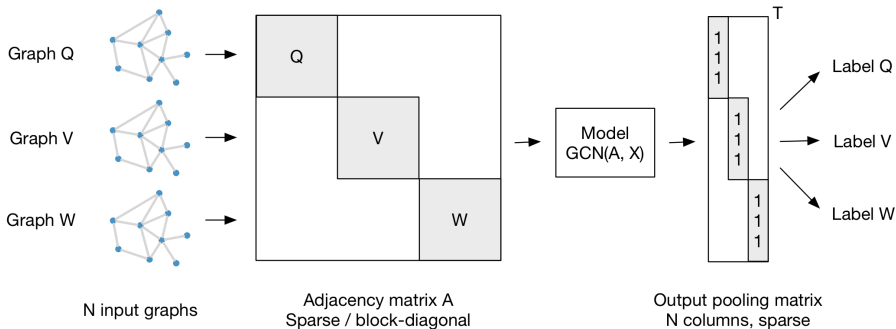
$$\mathbf{W}^*, \mathbf{Z}^* = \arg \min \frac{1}{|\mathcal{T}|} \sum_{(i,j) \in \mathcal{T}} \text{cross-entropy}(y_{ij}, \hat{y}_{ij}) .$$

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

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  $i$ th graph, we can obtain an embedding for the entire graph by performing a global pooling operation:

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

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



**Figure 6:** Batching multiple graphs in a single GCN call (taken from <https://github.com/tkipf/gcn>).



# Graph layers

---

Graph attention 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.

To conclude, we 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*.

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

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

where we recover the GC layer with  $\eta(\mathbf{x}_i, \mathbf{x}_j) = A_{ij}$  and  $g(\mathbf{x}) = \mathbf{W}^\top \mathbf{x}$ .

In a GAT layer, we first learn a set of *unnormalized* values, called the **attention scores** of the node:

$$a_{ij} = \mathbf{a}^\top \text{LeakyReLU}(\mathbf{W} [\mathbf{x}_i \parallel \mathbf{x}_j]) . \quad (6)$$

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

$$a_{ij} = \text{softmax}(a_{i0}, \dots, a_{iN}) .$$

---

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*.

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](https://github.com/pyg-team/pytorch_geometric).
- ▶ Spektral (TensorFlow, smaller than the other two):  
<https://graphneural.network/>.

- ▶ Graph Representation Learning Book:  
[https://www.cs.mcgill.ca/~wlh/grl\\_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/>.