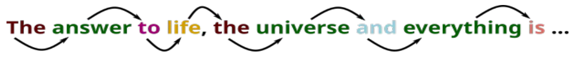
Introduction Pratique au Deep Learning

Graph Neural Networks (GNN)

INSTITUT DU DÉVELOPPEMENT ET DES RESSOURCES EN INFORMATIQUE SCIENTIFIQUE

- AI success was mainly due to computer vision, speech recognition, text completion ...
  - Highly structured data



What about other problems?
Chemistry, social science, physics, etc

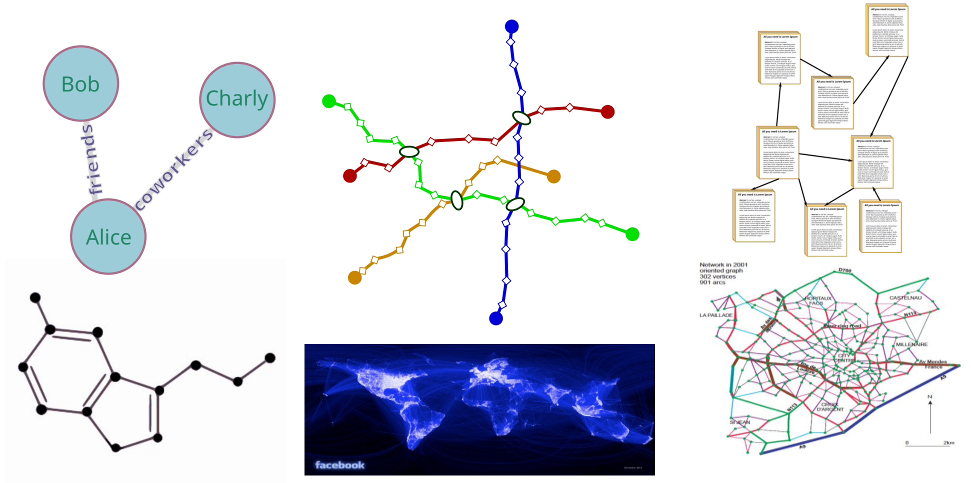
Damn you Euclid

The breakthrough of machine learning methods and their applications is mainly due to highly structured data. More precisely pictures (every pixel has a fixed number of neighbors) and text (either written or spoken).

Those data are said to be euclidean.

Is it possible to apply machine learning methods when the field do not generate euclidean data?

We will give you an insight with this introduction to Neural Networks applied to graphs.



Entities and relationships: nodes/vertices and edges

Entities (usually called nodes) sharing relationships is what defines a graph.

Relationships are modeled by a connection between 2 nodes (called edge). One node can share edges with several others.

Numerous fields naturally use data that can be represented by graphs. For example social networks or maps.

Other examples (maybe a bit less straightforward) include:

- Chemistry with atoms as nodes and bonds as edges
- Citations in research papers

What is a graph?

- Graphs can store information (features) on **nodes**, **edges** and **globally**

	Globally	Nodes	Edges
Social Network	Group of interest, ...	Name, Age, Job, ...	Is friend, follows, family, ...
Molecule	Is a drug, Energy, ...	Atomic number, ...	Bond order, ...
Citations	Field, ...	Article, ...	Was cited, ...
Particle physics	Experiment	Particle	Decayed to, ...
Motion capture	Character	Joints	Is connected to, ...
Natural language	Paragraph, ...	Group of words, ...	Refers to, ...

- It can be a number, a concept, ...

It is important to understand that graphs contains different kinds of information that are being stored on different parts:

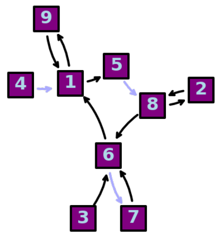
- Nodes
- Edges
- Globally

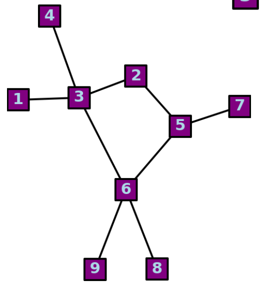
From now on we will call the pieces of information attributes.

Attributes can have different nature for example numbers or abstract concepts.

If we use abstract concepts, it is necessary to transform them in numbers. Otherwise it is not possible for computer to treat them.

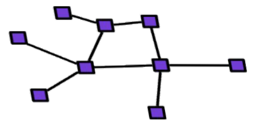
- Direction
  - Directed: Relationships are not symmetric
    - On twitter you can follow someone but not be followed by this person
    - A paper is cited in another paper
    - ...
  - Undirected: Relationships are symmetric
    - 2 atoms share the same kind of bond
    - ...



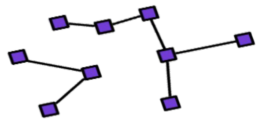


Graph properties

- Connectivity on undirected graphs
  - All nodes are connected via a path → **Connected Graph**



  - If some nodes are not connected to other via a path they are **disconnected**



Graph properties

Direction is an important features of graphs.

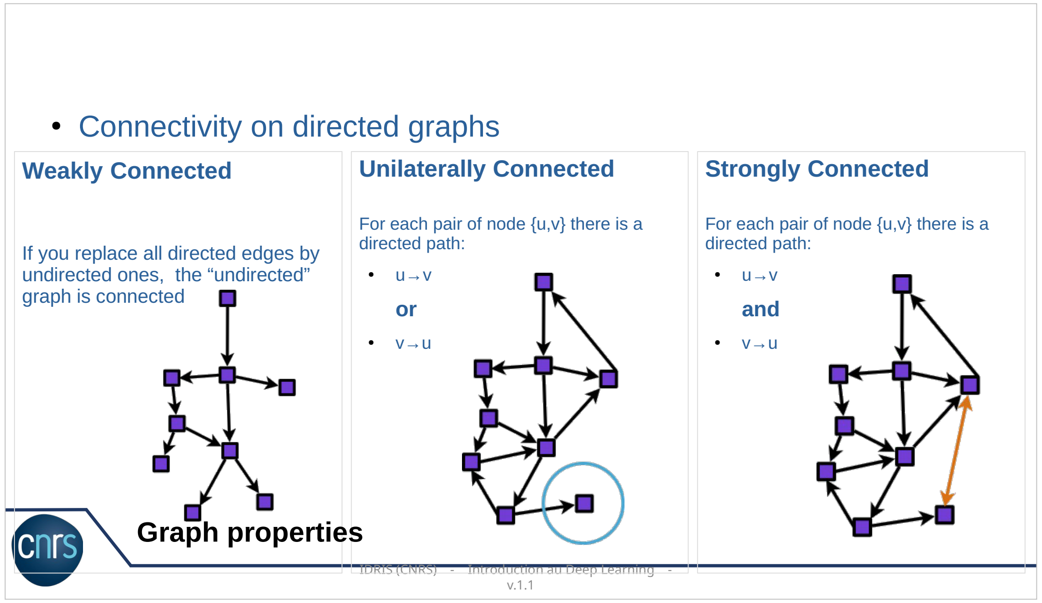
A graph is directed when the edges/relationships between nodes are not symmetrical.

As an example, on Twitter it is not necessary to be followed by a person to follow her.

When the edges are symmetrical, the graph is undirected. It is the case for molecules where the nature of the bond is identical between the atoms sharing it.

Connectivity is a feature of graphs telling if it is possible the follow a path from one node to any other in the graph.

In the case of undirected graphs, it is quite easy to tell if they are connected. If they are not you can spot several isolated graphs not sharing any edge.



- Connectivity on directed graphs

Weakly Connected

If you replace all directed edges by undirected ones, the “undirected” graph is connected

Unilaterally Connected

For each pair of node {u,v} there is a directed path:

- u→v

or

- v→u

Strongly Connected

For each pair of node {u,v} there is a directed path:

- u→v

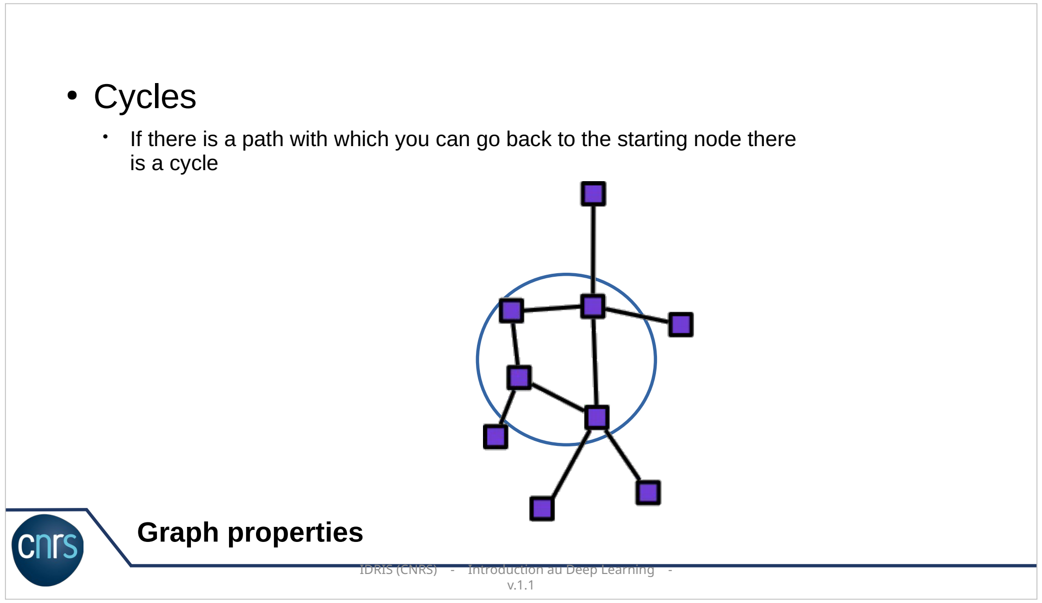
and

- v→u

Graph properties

For directed graphs we have 3 possibilities :

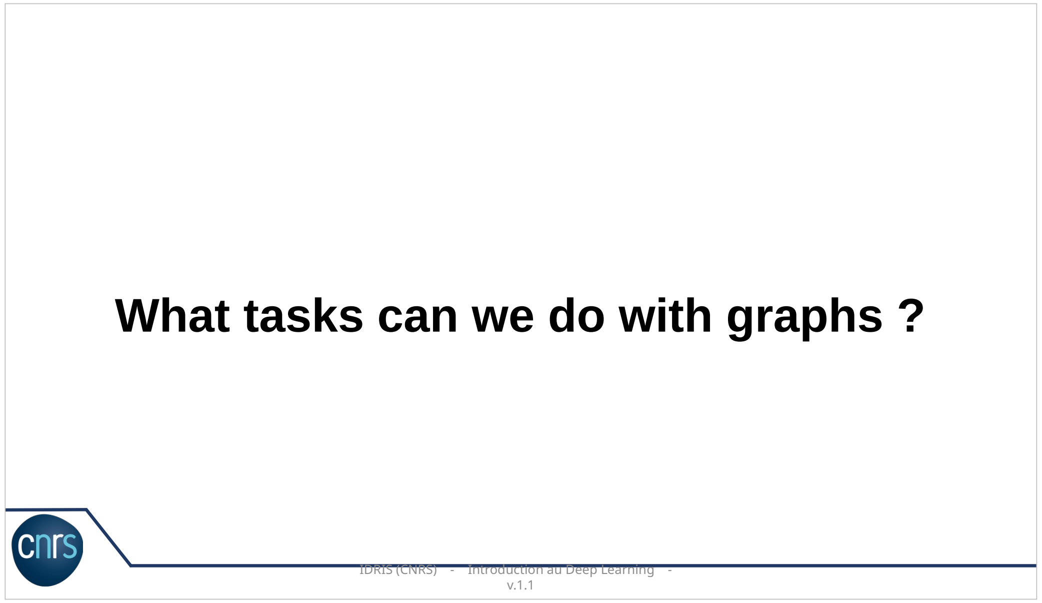
- A graph is said to be **weakly connected** if when you symmetrize the edges it is connected.
- A graph is said to be **unilaterally connected** if there is a path between 2 nodes (u and v):
  - u → v **OR** v→ u
- A graph is said to be strongly connected is there is a path:
  - u → v **AND** v→ u



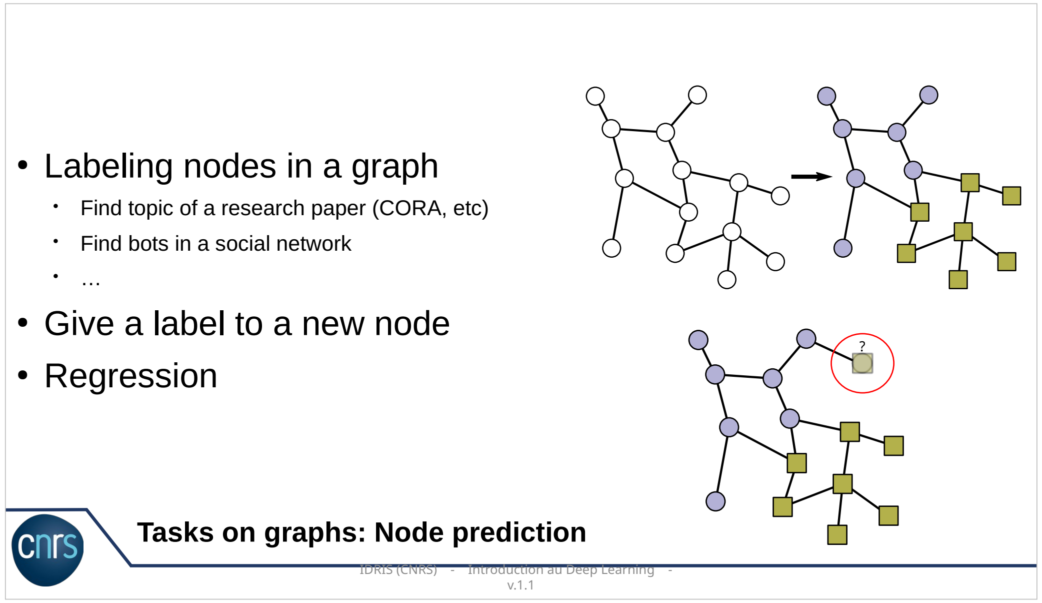
- Cycles
  - If there is a path with which you can go back to the starting node there is a cycle

Graph properties

If there is a path connecting a node to itself it is called a **cycle**.

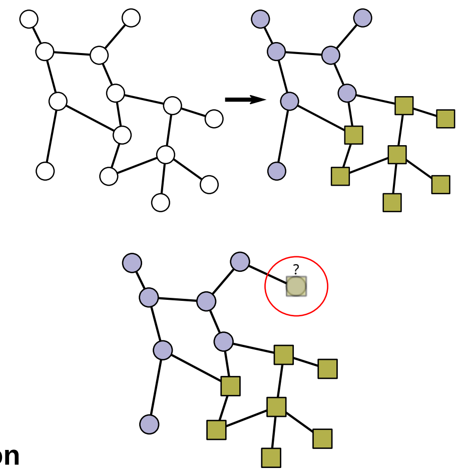


What tasks can we do with graphs ?



Tasks on graphs: Node prediction

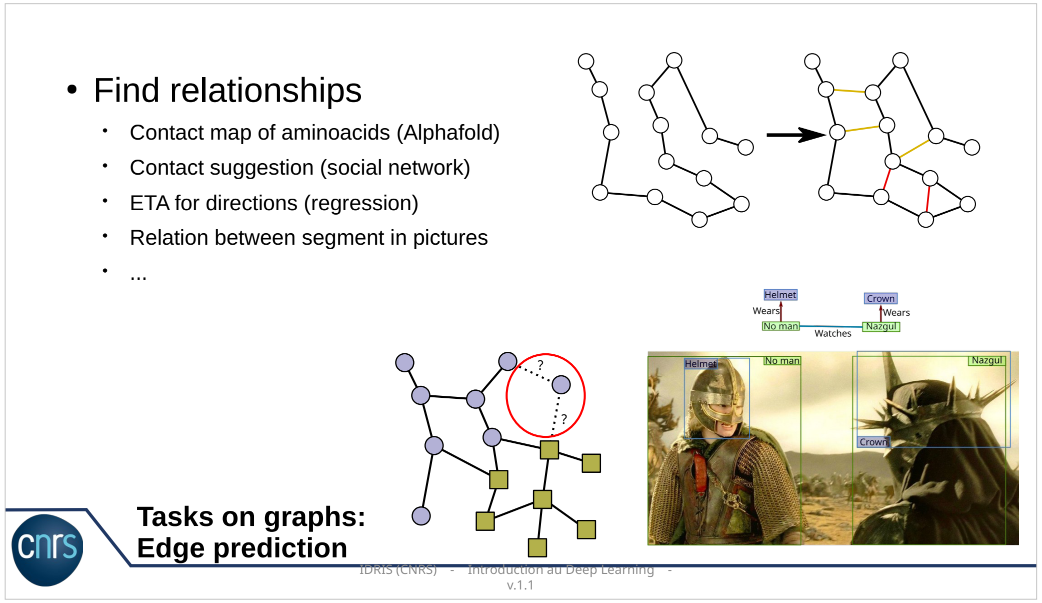
- Labeling nodes in a graph
  - Find topic of a research paper (CORA, etc)
  - Find bots in a social network
  - …
- Give a label to a new node
- Regression



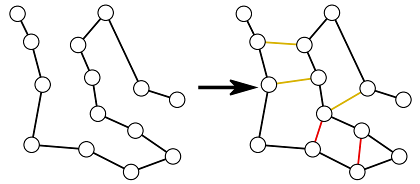
We can use Graph Neural Networks to make predictions on nodes.

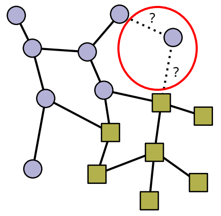
It can be either:

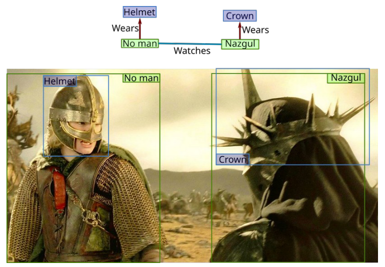
- Classification: we try to find a missing label for a node.
- Regression: we try to find a real value for a node attribute. For example, we can try to estimate the age of a social network user.



- Find relationships
  - Contact map of aminoacids (Alphafold)
  - Contact suggestion (social network)
  - ETA for directions (regression)
  - Relation between segment in pictures
  - ...





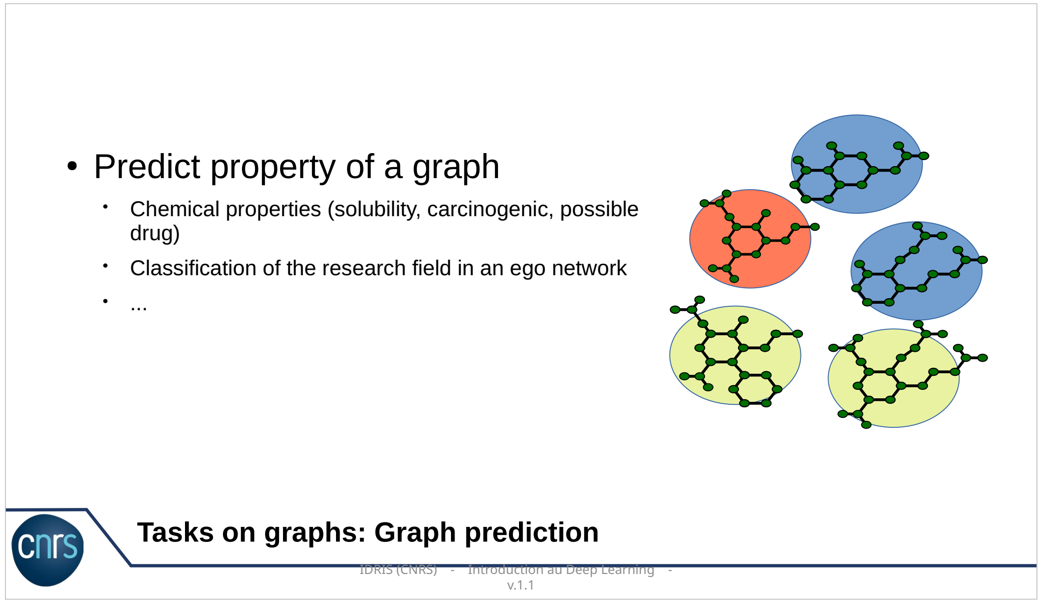


Tasks on graphs: Edge prediction

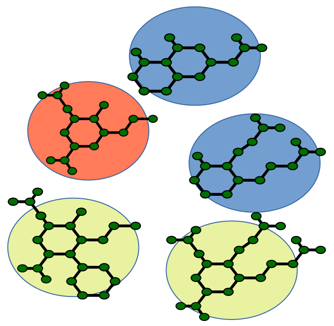
We can train neural networks to make prediction on edges. As for nodes we can perform:

- Classification
- Regression

An interesting application is to find relationship between parts of a pictures. The definition of the parts can be done by a semantic segmentation method done by another network beforehand.



- Predict property of a graph
  - Chemical properties (solubility, carcinogenic, possible drug)
  - Classification of the research field in an ego network
  - ...



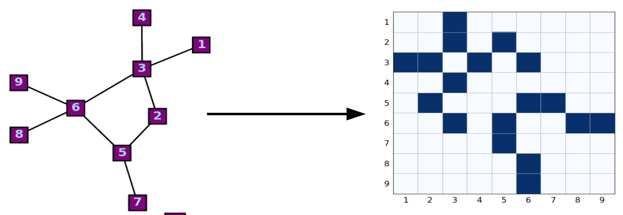
Tasks on graphs: Graph prediction

As for nodes and edges we can perform classification and regression on the global attributes of graphs.

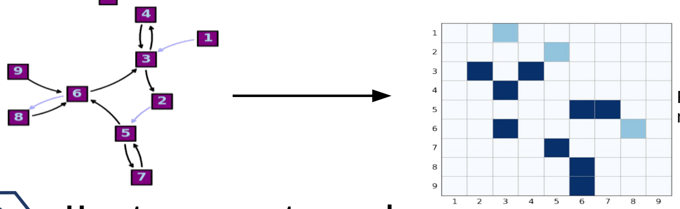
For example, we can find some chemical properties for molecules (carcinogenic, possible drug, …).

How to represent graphs?

- Adjacency matrix
  - NxN matrix with value ≠ 0 when an edge exists



For undirected graphs the adjacency matrix is symmetric



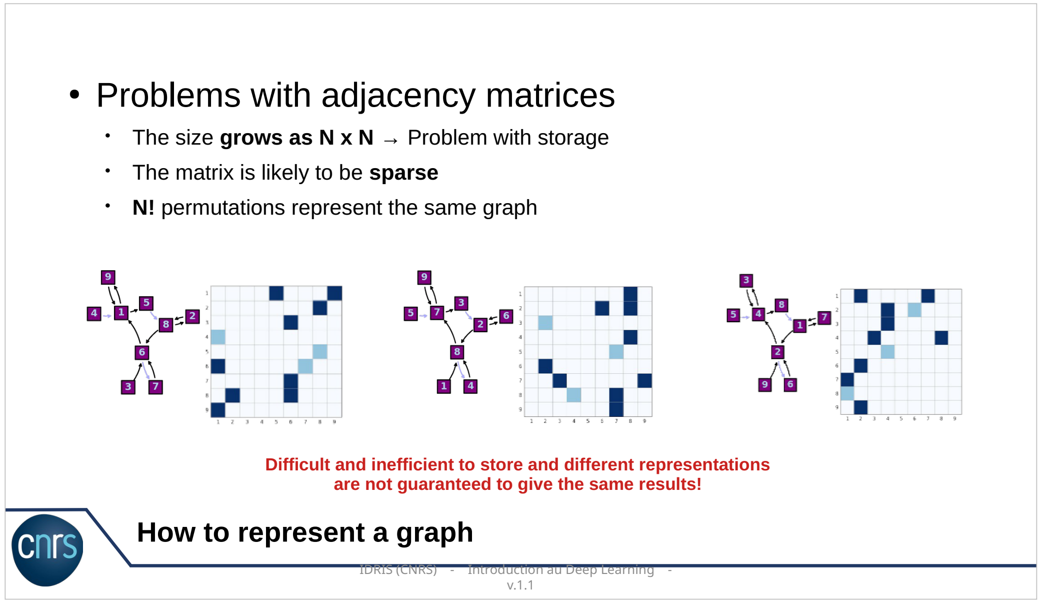
For directed graphs the adjacency matrix is NOT symmetric

How to represent a graph

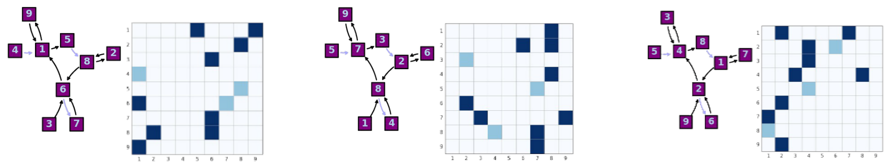
The most straightforward way to represent a graph on a computer is to use its **adjacency matrix.**

For a N nodes graph, we have a NxN adjacency matrix. It contains non-zero elements where an edge exists between 2 nodes.

For undirected graph, the adjacency matrix is symmetrical.

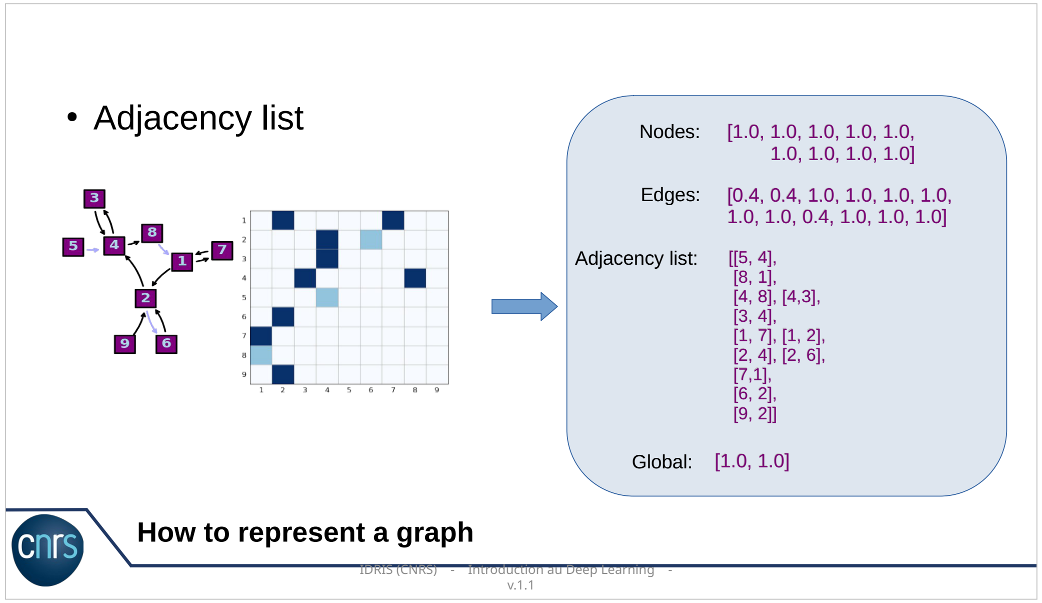


- Problems with adjacency matrices
  - The size **grows as N x N** → Problem with storage
  - The matrix is likely to be **sparse**
  - **N!** permutations represent the same graph

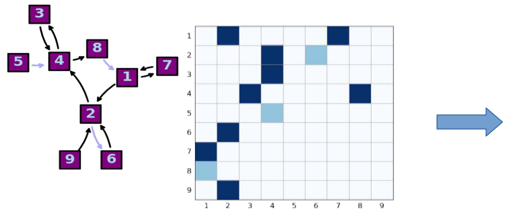


Difficult and inefficient to store and different representations are not guaranteed to give the same results!

How to represent a graph



- Adjacency list



Nodes:	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Edges:	[0.4, 0.4, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.4, 1.0, 1.0, 1.0]
Adjacency list:	[[5, 4], [8, 1], [4, 8], [4,3], [3, 4], [1, 7], [1, 2], [2, 4], [2, 6], [7,1], [6, 2], [9, 2]]
Global:	[1.0, 1.0]

How to represent a graph

Several problems make the adjacency matrix difficult to use in practice:

- The size increases quadratically with the number of nodes. The memory space needed raises quite fast.
- The sparsity of the matrix is likely to be high i.e. the number of non-zero values quite small.
- If we number arbitrarily the nodes there are N! permutations representing the same graph. It is a problem since there is no guaranty that the neural network will return the same results for all permutations.

Most GNN libraries choose to represent graphs with an **adjacency list**.

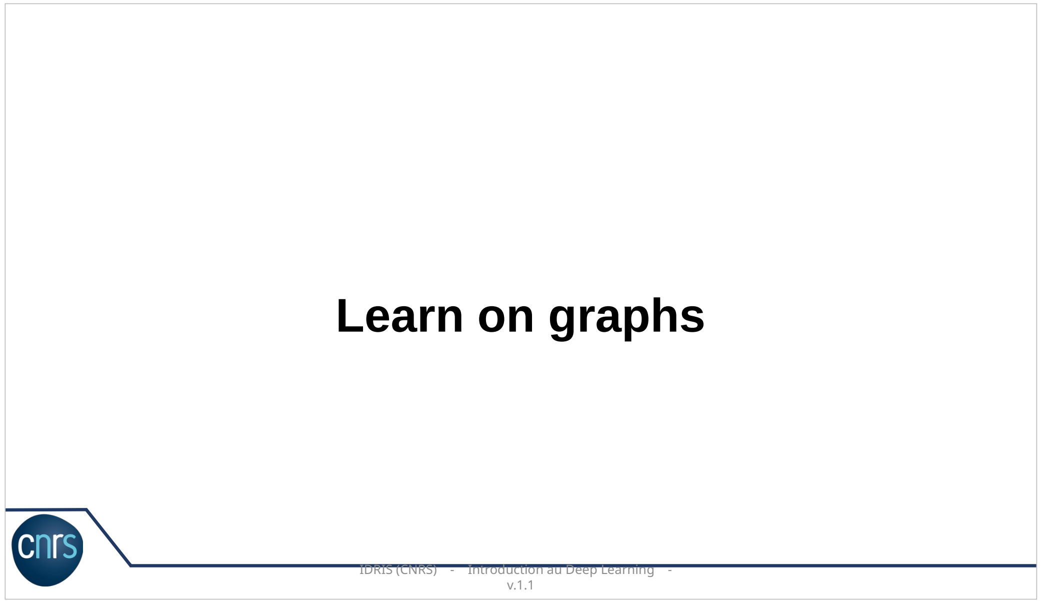
It is a list of pairs of nodes sharing an edge.

The size of the list is proportional to the number of edges in the graph.

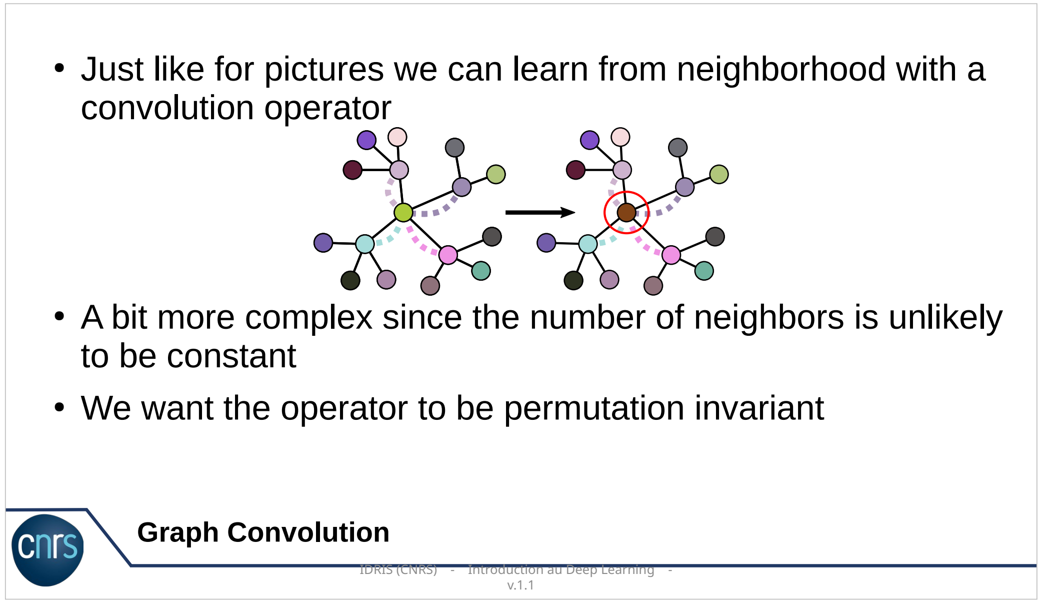
At the end we store several lists:

- Attributes on nodes
- Attributes on edges
- Global attributes
- The adjacency matrix

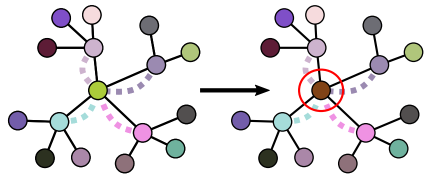
In the general case the attributes are vectors and not scalars as in the example.



Learn on graphs



- Just like for pictures we can learn from neighborhood with a convolution operator



- A bit more complex since the number of neighbors is unlikely to be constant
- We want the operator to be permutation invariant

Graph Convolution

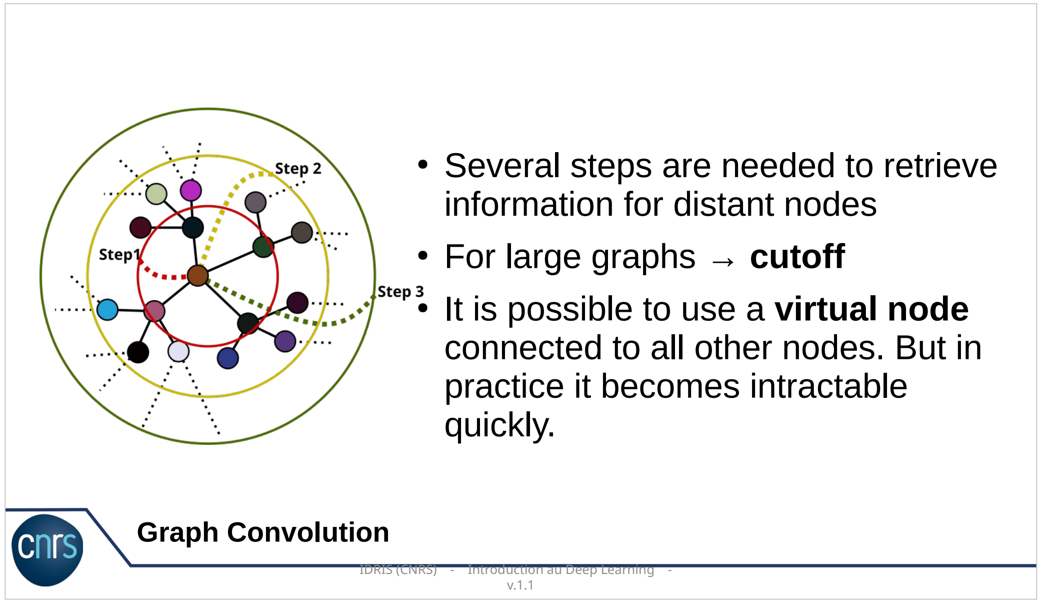
To learn on a graph we want to gather information from the edges and nodes environment.

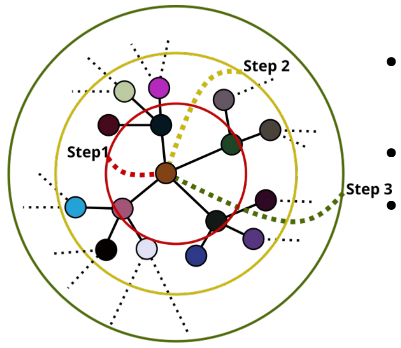
The neighbors attributes help to understand which role the node/edge has on the graph. So we want to retrieve information from neighbors.

It is quite similar to what you have already seen with CNNs. It is possible to consider GNN as a generalization of CNN on non-euclidean structures.

For pictures the number of neighbors for each pixel is known (8). On graph this number is variable.

A convolution step gets the information for directly connected neighbors. We want the convolution operator to be permutation invariant.





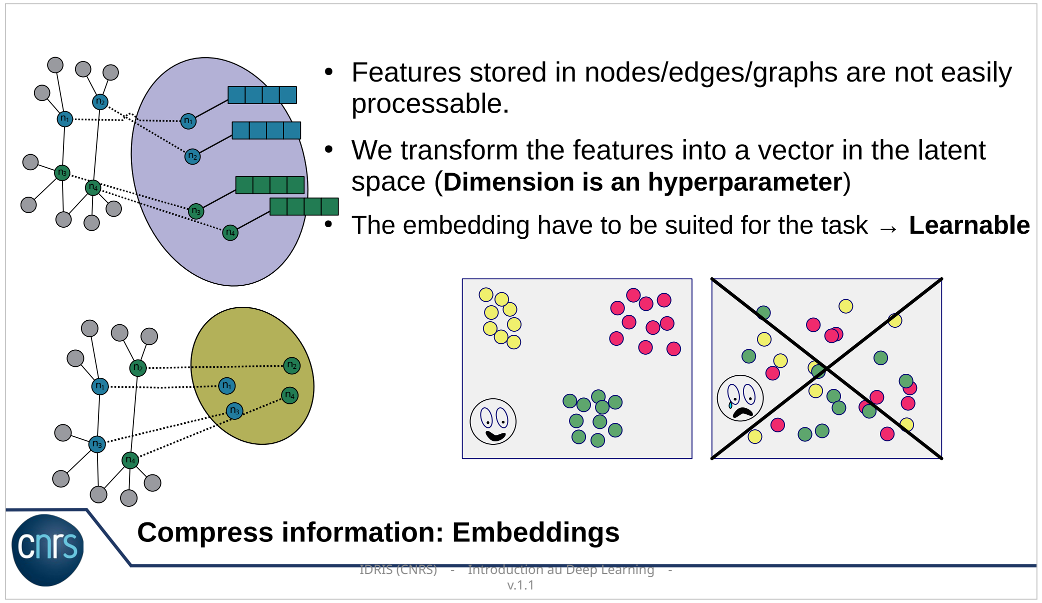
- Several steps are needed to retrieve information for distant nodes
- For large graphs → **cutoff**
- It is possible to use a **virtual node** connected to all other nodes. But in practice it becomes intractable quickly.

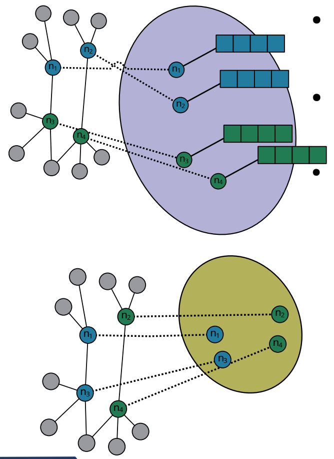
Graph Convolution

Now that we have the information from direct neighbors we may want to go further on the paths. To do so we can repeat several convolution steps. x steps allow to reach the xth neighbors.

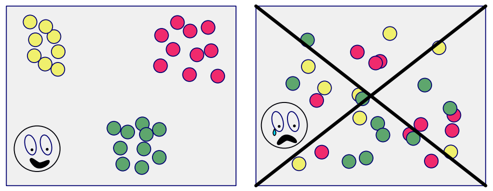
It is difficult to get information on all the graph when the number of nodes is high. Usually we can assume that only the closest neighbors have very useful information so we stop the convolution steps after a number of steps that is an hyperparameter to define.

Another solution is to have a virtual node connected to all other nodes which pass information more rapidly. Still the size of the graph can be a problem.





- Features stored in nodes/edges/graphs are not easily processable.
- We transform the features into a vector in the latent space (**Dimension is an hyperparameter**)
- The embedding have to be suited for the task → **Learnable**



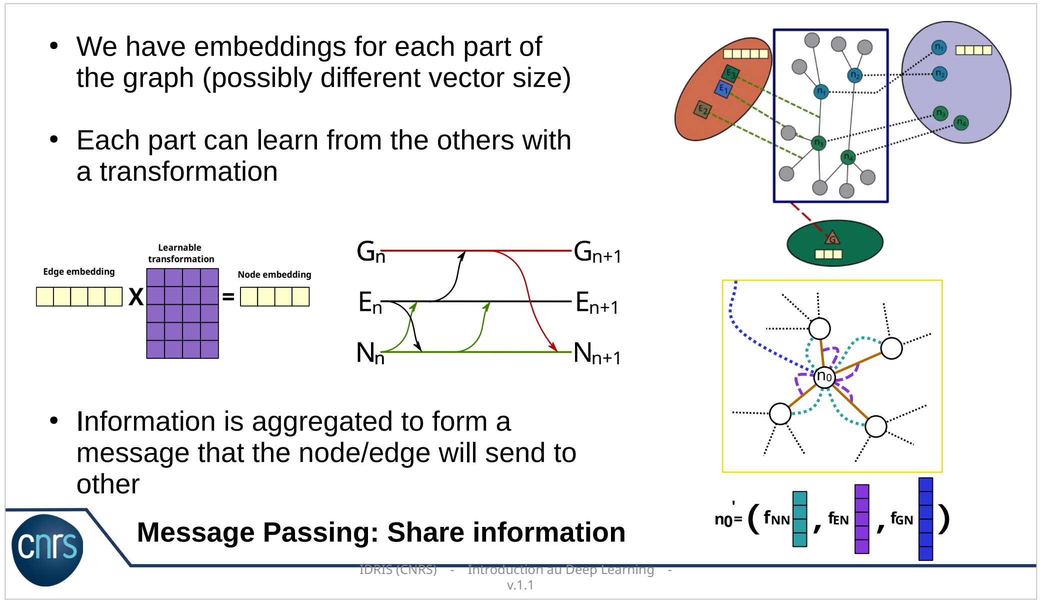
Compress information: Embeddings

Unlike pixels, graph attributes can be numerous. We want the most efficient representation of the attributes for the learning task we are performing.

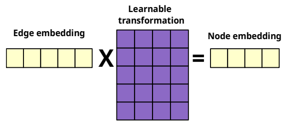
It is possible to define a vector space (latent space) whose dimension is a hyperparameter to set.

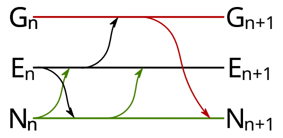
The dimension has to be correctly choosen so that the network can learn efficiently:

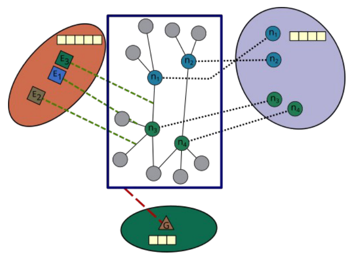
- If too small we face an underfitting problem
- If too large we can overfit

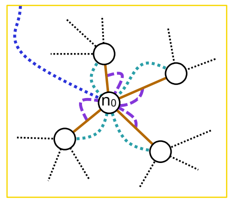


- We have embeddings for each part of the graph (possibly different vector size)
- Each part can learn from the others with a transformation





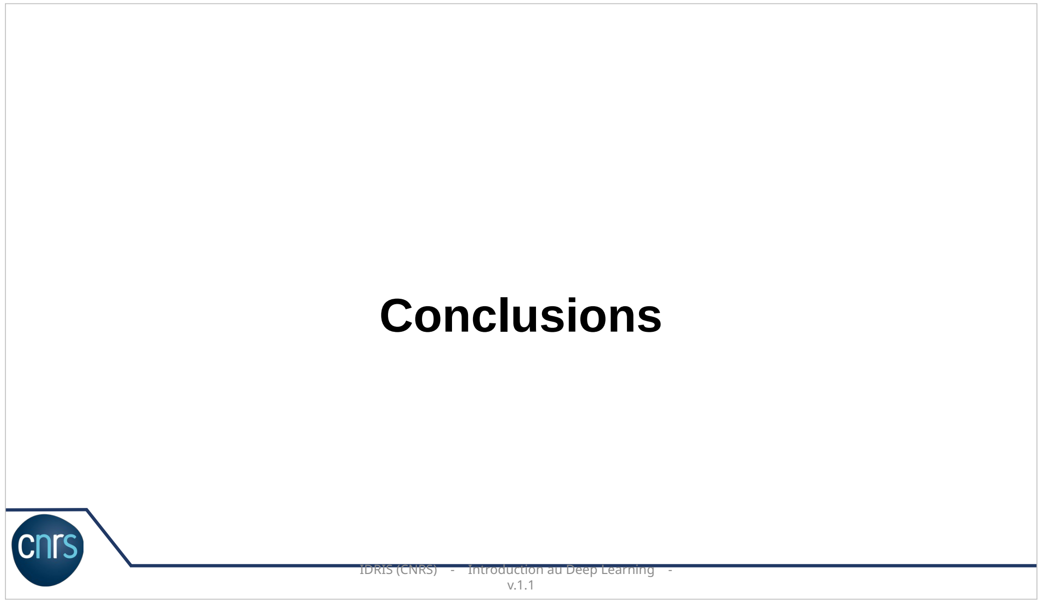




$n'_0 = (f_{NN} , f_{EN} , f_{GN})$

- Information is aggregated to form a message that the node/edge will send to other

Message Passing: Share information



Conclusions

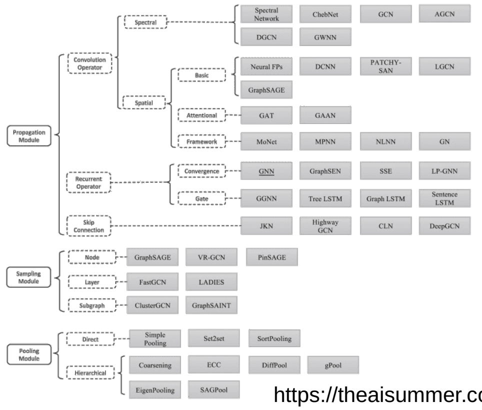
We introduce the concept of **Message Passing** for GNNs. Here the idea is to aggregate vectors to give a better representation.

A latent space can be defined for nodes, edges and globally. The dimension of those spaces do not have to be identical.

We can gather information from all parts of the graph to get a better picture of the environment. This can help with the learning task. However since the dimensions might not be the same, we need to define a (learnable) transformation from one part to another (eg. edge latent space → node latent space).

Once the messages are aggregated we can send a global message to the neighbors.

Several model architectures are available



https://theaisummer.com/gnn-architectures/

Models

- It is possible to use Deep Learning on non-Euclidean data structures. The field is called Geometric Deep Learning https://geometricdeeplearning.com/
- Graph structures appear easily on many scientific problems
- GNN can be seen as a generalization of convolution
- We can aggregate features to form a message to be passed
- There are several models already available
- A large part of the problem is to find a good way to transform the original data to fit NN architectures → Representation learning

Conclusion

Available libraries

- Pytorch Geometric
- Deep Graph Library
- Graph Nets
- Spektral
- ...

References

- Books
  - Deep Learning on Graphs (Jiliang Tang and Yao Ma)
  - Introduction to Graph Neural Networks (Introduction to Graph Neural Networks)
- Websites
  - https://distill.pub/2021/gnn-intro/
  - https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications
  - https://venturebeat.com/2021/10/13/what-are-graph-neural-networks-gnn/
  - https://theaisummer.com/graph-convolutional-networks/
  - https://towardsdatascience.com/node-embeddings-for-beginners-554ab1625d98
- Articles
  - **Zhou, Jie, et al. "Graph neural networks: A review of methods and applications." AI Open 1 (2020): 57-81.**
  - **Scarselli, Franco, et al. "The graph neural network model." IEEE transactions on neural networks 20.1 (2008): 61-80.**
  - **Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).**
  - **Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014.**
  - **Shlomi, Jonathan, Peter Battaglia, and Jean-Roch Vlimant. "Graph neural networks in particle physics." Machine Learning: Science and Technology 2.2 (2020): 021001.**
  - **Duong, Chi Thang, et al. "On node features for graph neural networks." arXiv preprint arXiv:1911.08795 (2019).**