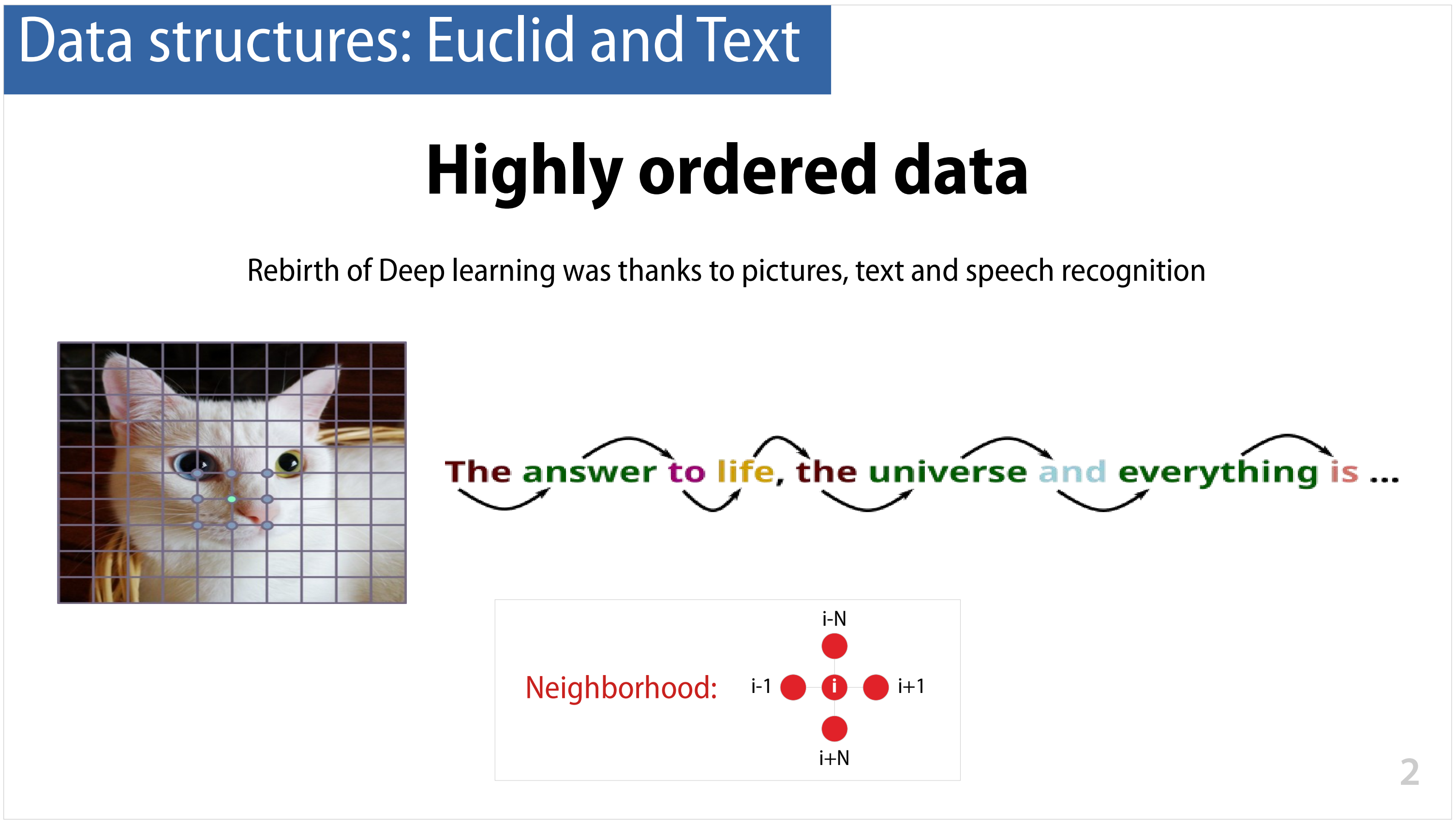
Hands-on Introduction to Deep Learning

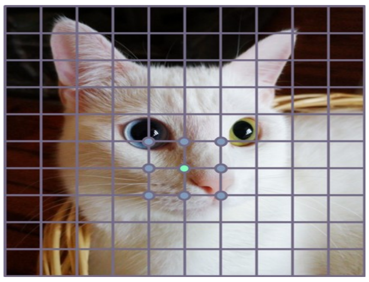
Graphs are everywhere

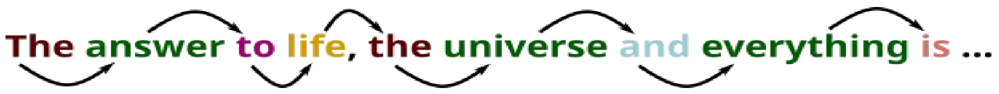


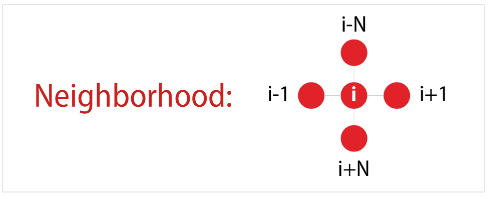
Data structures: Euclid and Text

Highly ordered data

Rebirth of Deep learning was thanks to pictures, text and speech recognition







So far, we have worked with very regular data: images and language processing.

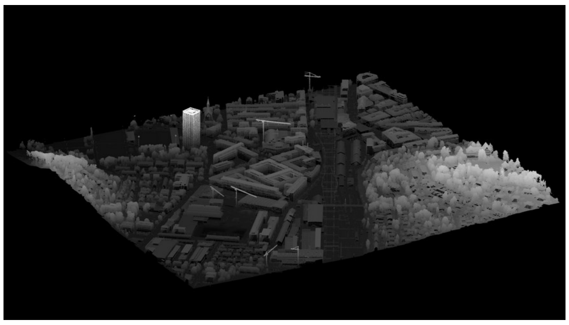
An image is a set of pixels spread out on a regular grid, and each pixel has 4 neighbors (north, south, east, west).

Similarly, a text is a set of words that follow one another.

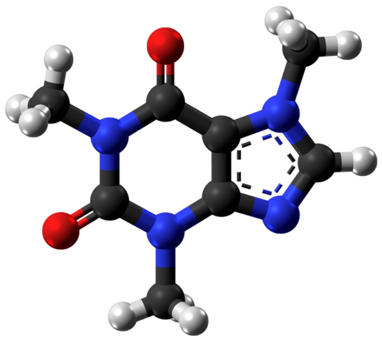
The first Deep Learning models were created to train neural networks on this type of data.

Data structures: Data is not always euclidean

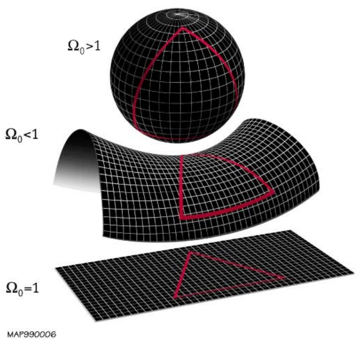
LIDAR



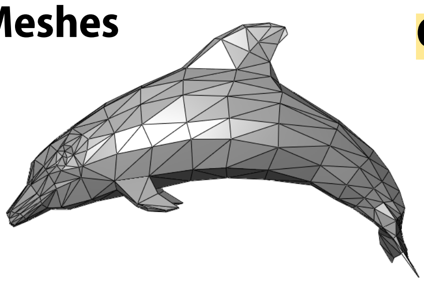
Molecules



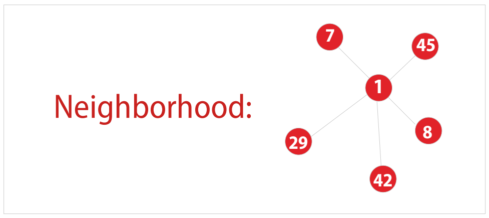
Complex geometries



Meshes



Geometric deep learning



3

Michael M. Bronstein, Joan Bruna, Taco Cohen, Petar Veličković, Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges https://arxiv.org/abs/2104.13478

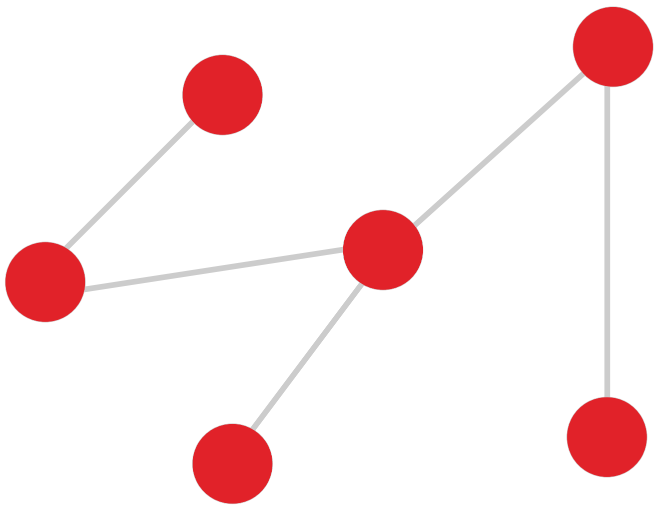
The methods developed to process regular data (images or text) are not necessarily applicable to all types of data.
Everywhere, we find data with a much more irregular structure, with in particular a concept of non-generalizable neighborhood.
Examples:

- Object detection in a LIDAR scan (point cloud)
  - The point cloud is divided into overlapping voxels
  - 1 entity = 1 voxel, information is averaged over the voxel
  - 2 entities are linked if they have points in common (~neighborhood)
- Searching for the function of a molecule from its structure
  - 1 entity = 1 atom
  - relation = 1 bond (which can be single, double, etc.)
  - Unstructured (free) meshes, locally refined
- We use the finite element/volume method, for example, discretizing the surface into small elements (triangles here)
  - 1 entity = intersection point between the vertices of the triangles
  - Relation = neighborhood
- Complex geometries

It is difficult to apply a CNN filter, for example, on this type of data.

The branch of Geometric Deep Learning attempts to apply methods initially created for regular data (images, text) to more complex geometries.
Graph neural networks are part of this research branch.

Graphs are everywhere

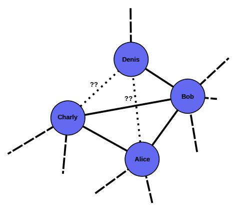


Data as a set of interconnected entities

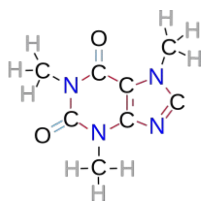
In an informal way, graphs define relationships between different entities.
We are going to give more details after.

Graphs are everywhere

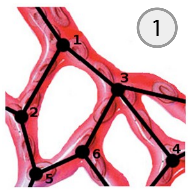
Social networks



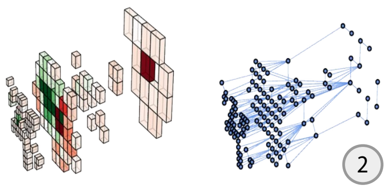
Molecules



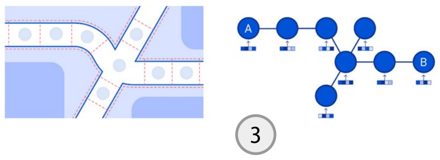
Capillary networks



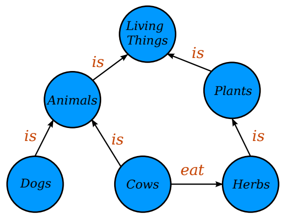
Particle physics



Directions recommendation



Knowledge graphs



Many other fields

- Biology
- Recommendation systems
- Computer vision
- Medical diagnosis
- Robotics
- ...

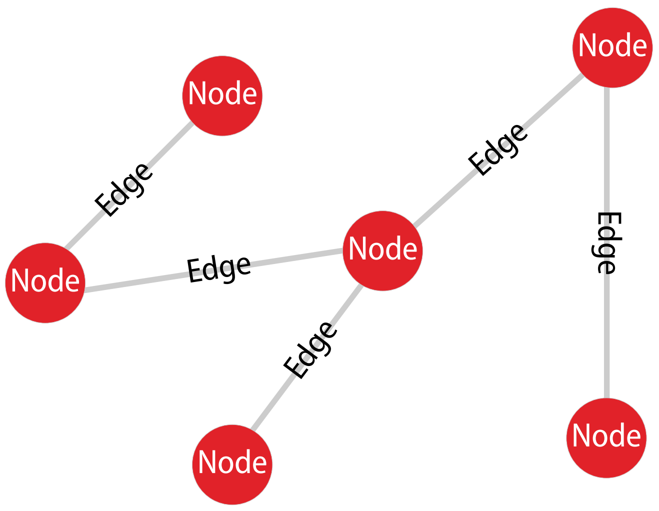
[1] Erbertseder, K., Reichold, J., Flemisch, B., Jenny, P., & Helmig, R. (2012). A coupled discrete/continuum model for describing cancer-therapeutic transport in the lung. PLoS One, 7(3), e31966.
[2] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," Mach. Learn.: Sci. Technol., vol. 2, no. 2, p. 021001, Jan. 2021, doi: 10.1088/2632-2153/abbf9a.
[3] A. Derrow-Pinion et al., "ETA Prediction with Graph Neural Networks in Google Maps," in Proceedings of the 30th ACM International Conference on Information & Knowledge Management New York, NY, USA, Oct. 2021, pp. 3767–3776. doi: 10.1145/3459637.3481916.

We find graphs in various application domains:

- Social networks (canonical example): profiles + friend/colleague/family relationships
- Molecules: atoms + bonds
- Capillary networks: intersections + vessels
- Particle physics (particle decay): final state linked to the original state
- In particle physics, particle decay is the spontaneous process of one unstable subatomic particle transforming into multiple other particles.
- Route recommendations: road sections + travel time
- Knowledge bases (used by Google's search engine, for example): concepts + semantic links.

Vocabulary

Graph

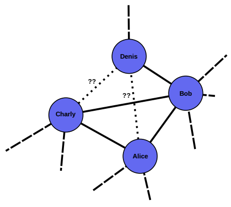


On met en place du vocabulaire et des notions de base.

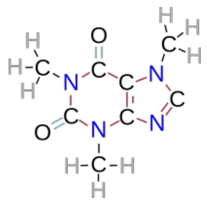
Node/vertex

Some example of nodes

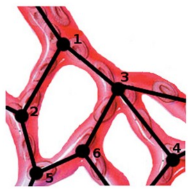
Persons



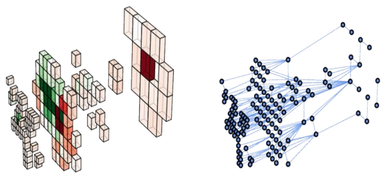
Atoms



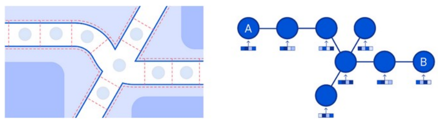
Intersections



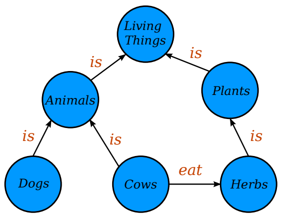
Particles



Road sections



Concepts

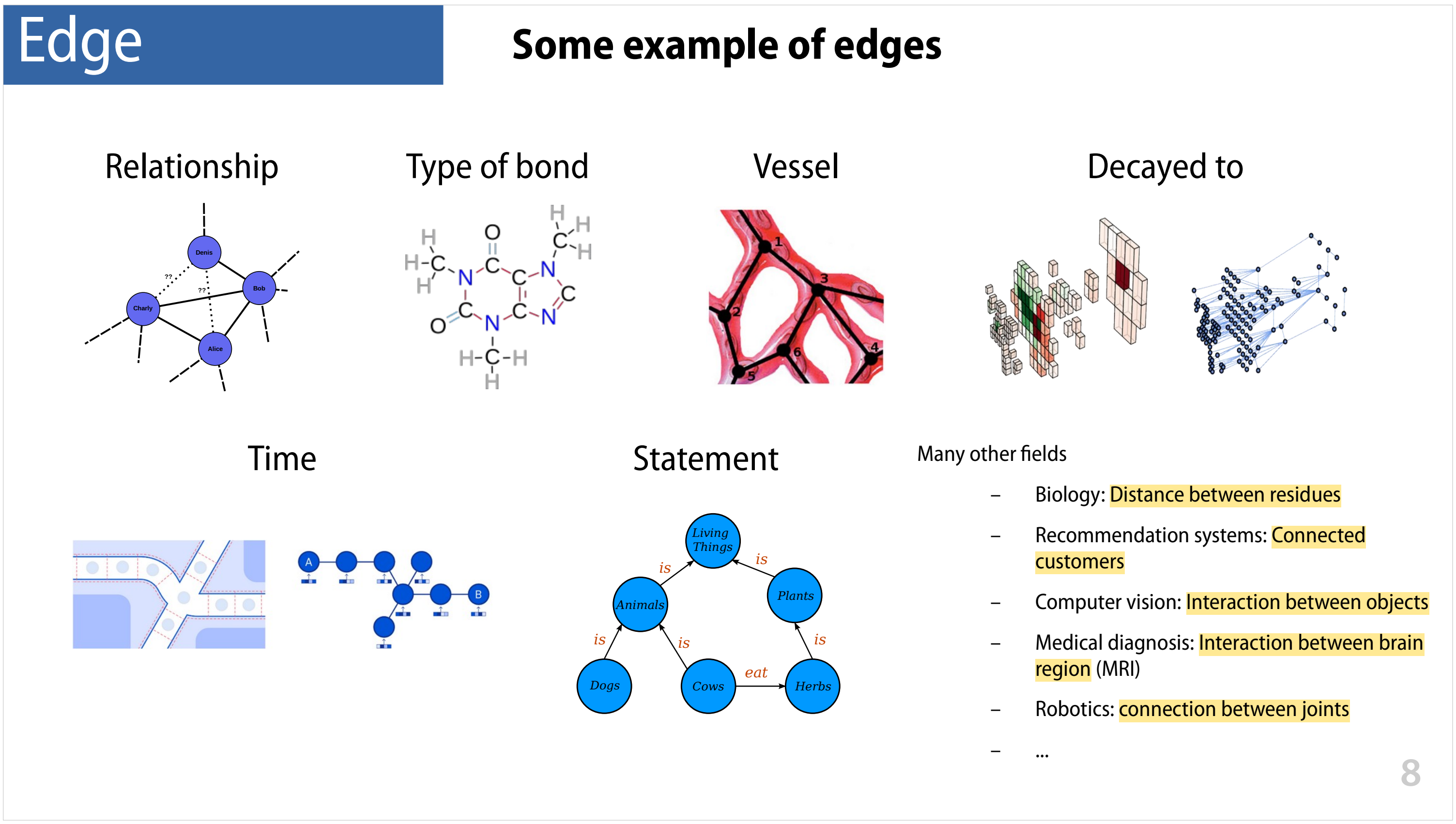


Many other fields

- Biology: An aminoacid in a protein
- Recommendation systems: A customer
- Computer vision: An object in a picture
- Medical diagnosis: Brain region (MRI)
- Robotics: Joints
- …

Exemples de noeuds dans différents domaines d’application :

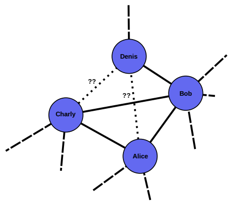
- Les réseaux sociaux : profils
- Les molécules : atomes
- Réseaux capillaires : intersections
- La physique des particules : état de la particule
- Les recommandations d’itinéraire : section de route
- Les bases de connaissance : concepts



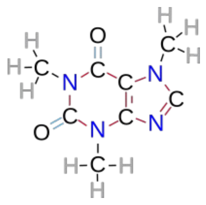
Edge

Some example of edges

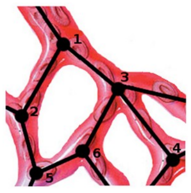
Relationship



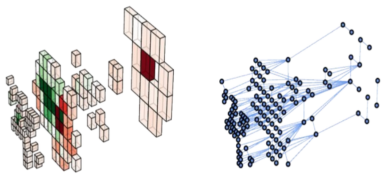
Type of bond



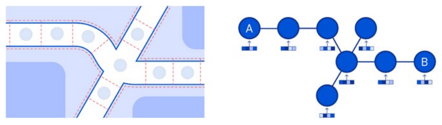
Vessel



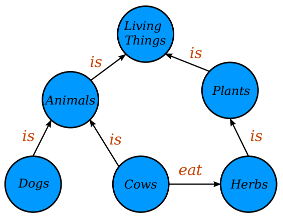
Decayed to



Time



Statement



Many other fields

- Biology: Distance between residues
- Recommendation systems: Connected customers
- Computer vision: Interaction between objects
- Medical diagnosis: Interaction between brain region (MRI)
- Robotics: connection between joints
- ...

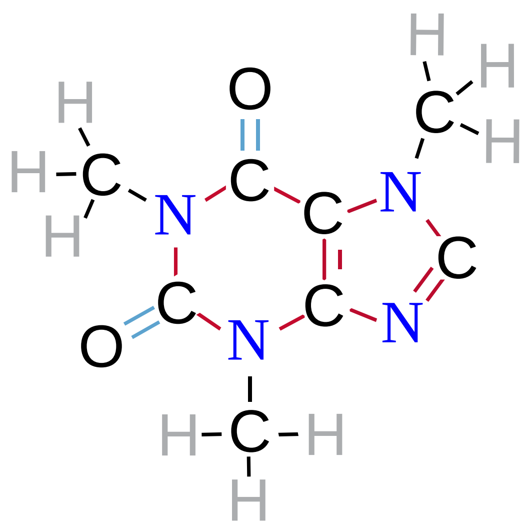
Examples of edges in various application domains:

- Social networks: relationships: friend/colleague, family
- Molecules: bonds (single, double, etc.)
- Capillary networks: vessels
- Particle physics: "decays into"
- Route recommendations: travel time
- Knowledge bases: semantic link

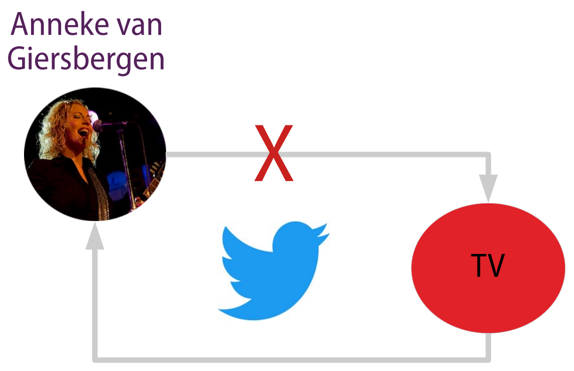
Edge: orientation

A relationship can be symmetrical or not between nodes

Undirected graphs



Directed graphs



In a graph, edges can be oriented or non-oriented. This means that the relationship that connects two nodes can be reciprocal (bidirectional) or not (unidirectional).

If the concept of orientation exists on the edges, we refer to a directed graph.

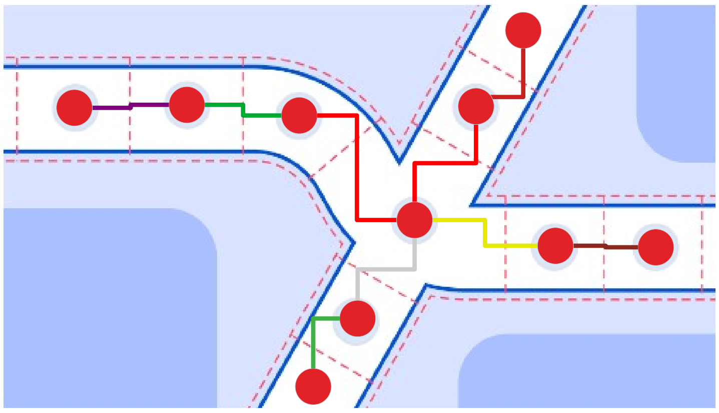
Example of an undirected graph: a molecule. If the carbon atom is bonded to a hydrogen atom, the reverse is also true.

Example of a directed graph: social networks like Twitter. TV follows the Dutch singer Anneke van Giersbergen on Twitter, but the reverse is not true :(

We will see later that the concept of orientation affects how we numerically represent a graph.

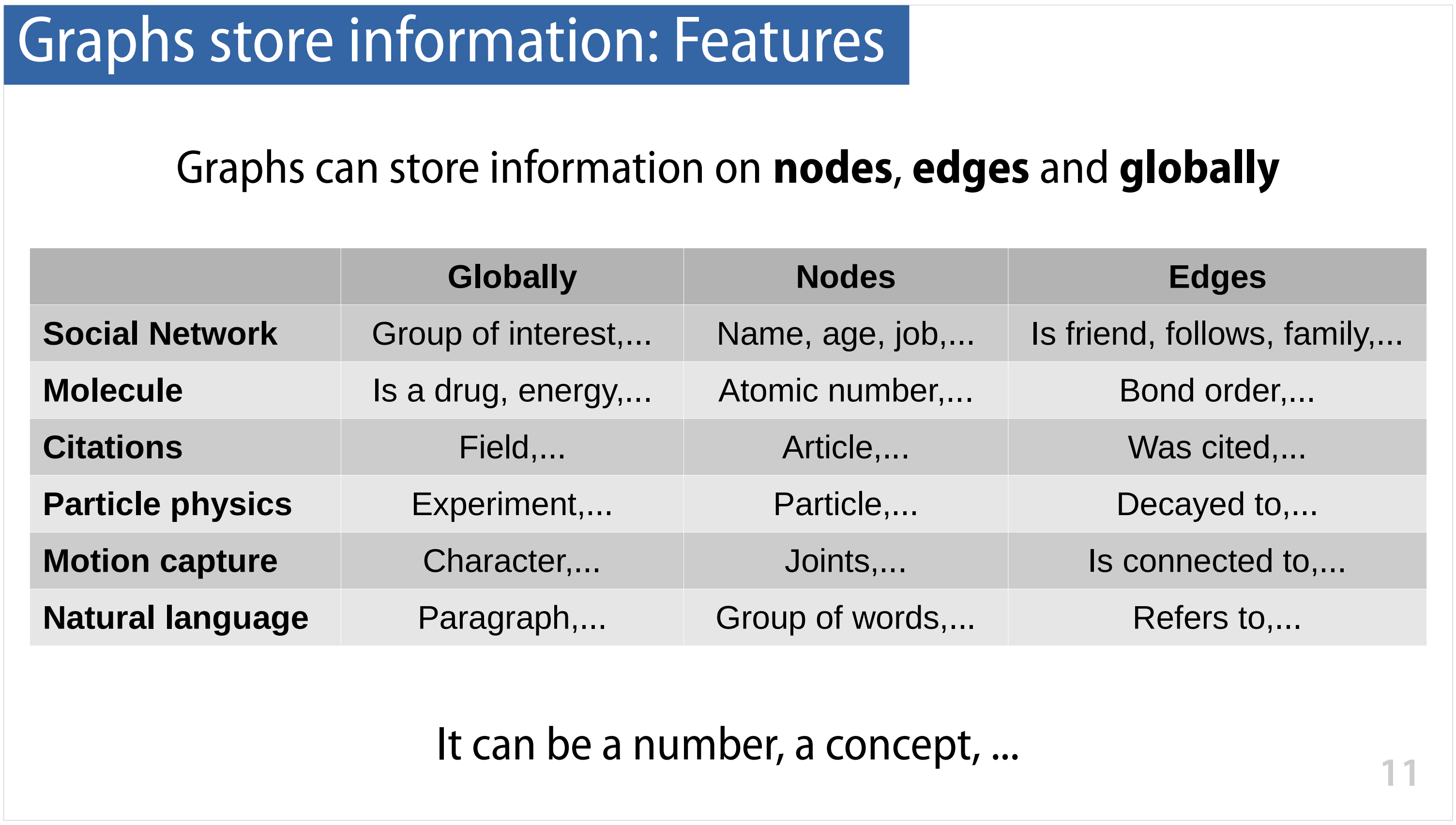
Edge: weight

Edges can carry information → **edge weight**



Edges can carry quantitative information. In this case, we refer to the weight of the edge.

For example: travel time between two road segments.



Graphs store information: Features

Graphs can store information 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, ...

Information (quantitative or qualitative) can be stored at several levels in a graph:

- At the level of the entire graph
- At the level of a node
- At the level of an edge

We refer to these as features.

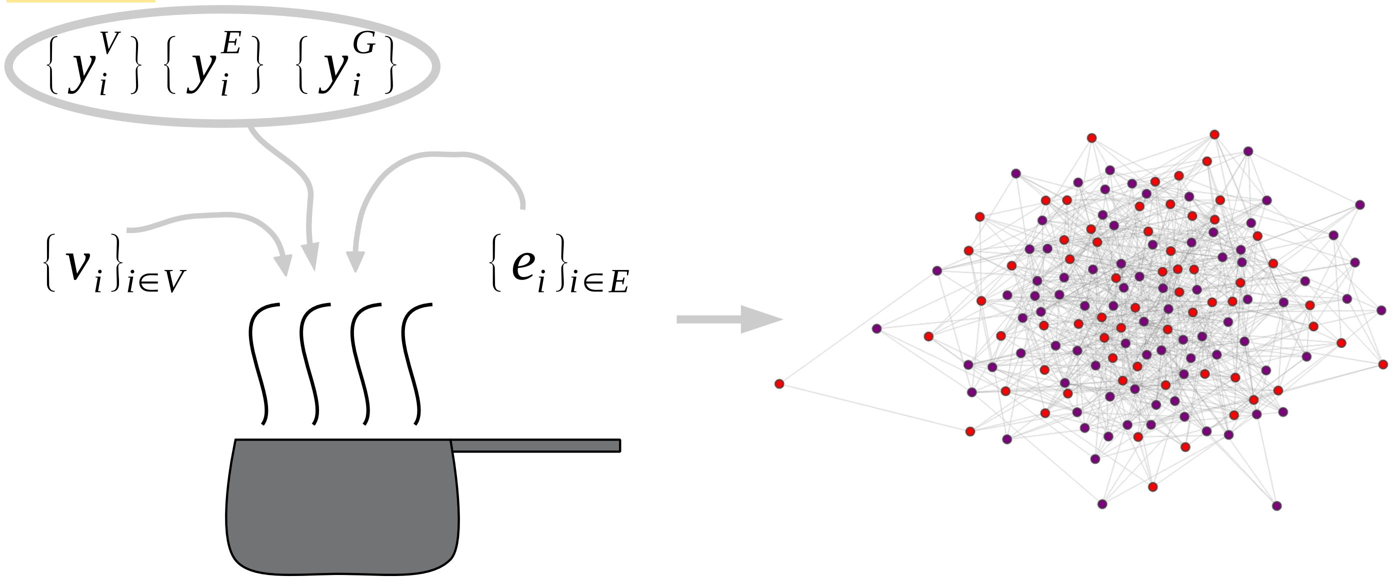
It is on this type of information that our model will be able to learn.

Features can be carried by the input data of the model and/or serve as labels during evaluation.

Formal definition

G = (V, E): a set of nodes and edges

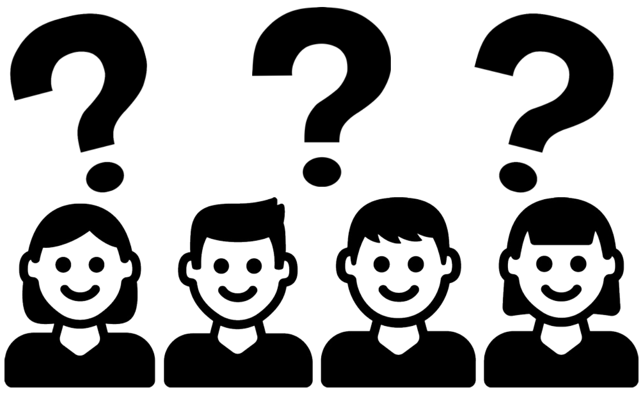
Features



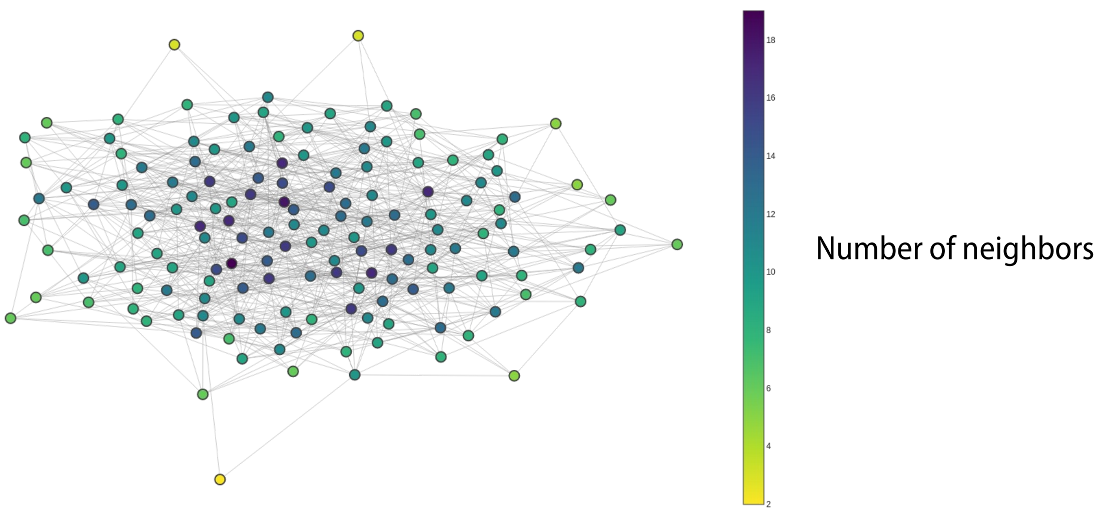
Now we have all the elements to build a graph.

We're cooking up the basics: a graph is nodes, edges, and features (on nodes, edges, and/or the graph).

Question break



Graph: Complexity



- The inner structure of a graph can vary a lot
- The number of edges/nodes might vary a lot from one graph to another
- One single graph can contain several thousand of nodes/edges
- ...

Graphs are complex structures.

The nodes of a single graph can have an extremely variable number of neighbors (see figure, number of neighbors ranging from 2 to 18).

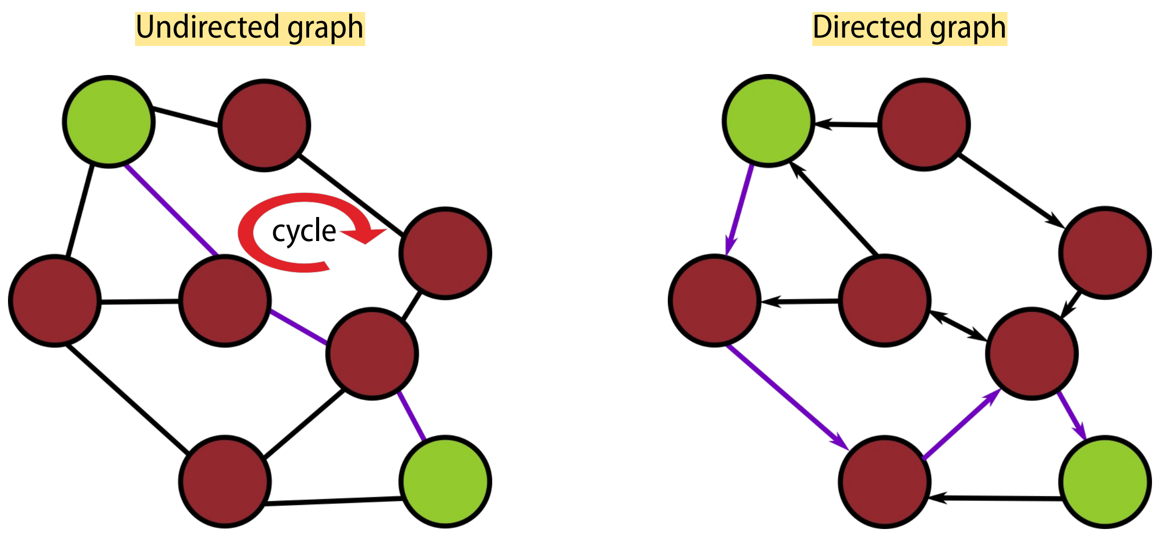
In the same database, two graphs can have very different structures (for example, a database of molecules will contain simple molecules with 2 atoms, up to complex molecules with several dozen atoms).

Some graphs can contain millions of nodes and edges (social networks).

Working with graphs can be a challenge both from a numerical point of view (enormous tensors to manipulate), but also for finding patterns during learning (variability of structures)..

Graph: Paths

A **path** is a sequence of edges connecting 2 nodes



There are concepts that help measure certain structural characteristics of a graph, locally or globally. We'll see a few of them, but there are many others (graph theory).

Characterizing the structure of a graph will be important later to verify that a model learns correctly, without distorting the structure of the graph on which it learns, for example.

A path is a set of edges that connect two nodes together. There can be multiple paths between two nodes.

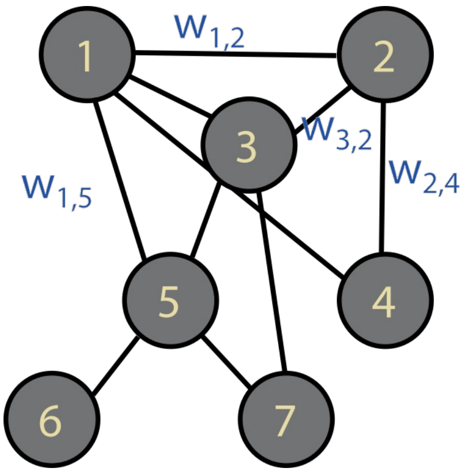
On a directed graph, the path between two nodes tends to lengthen. It may not even exist.

We speak of a cycle when there is a path connecting a node to itself without passing twice through the same edge.

The notion of paths and cycles can be used to characterize the structure of a graph locally or at the scale of a graph.

This can, for example, be used to define a "redundancy rate" of the network, useful in application cases such as the study of blood circulation in capillary networks (predicting the impact of an aneurysm rupture).

Graph: Node proximity and centrality



Node centrality

Measure how many paths goes through the node

Node proximity

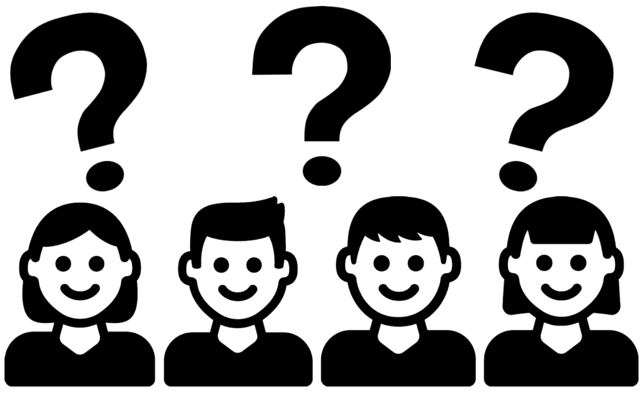
- 1st order: $\mathbf{w}_{i,j}$ between node i and j
- 2nd order: similarity of neighborhood structure
- Higher orders possible

The centrality of a node measures the number of paths that pass through this node. We are then able to identify the nodes that play a major role in the structure of the graph.

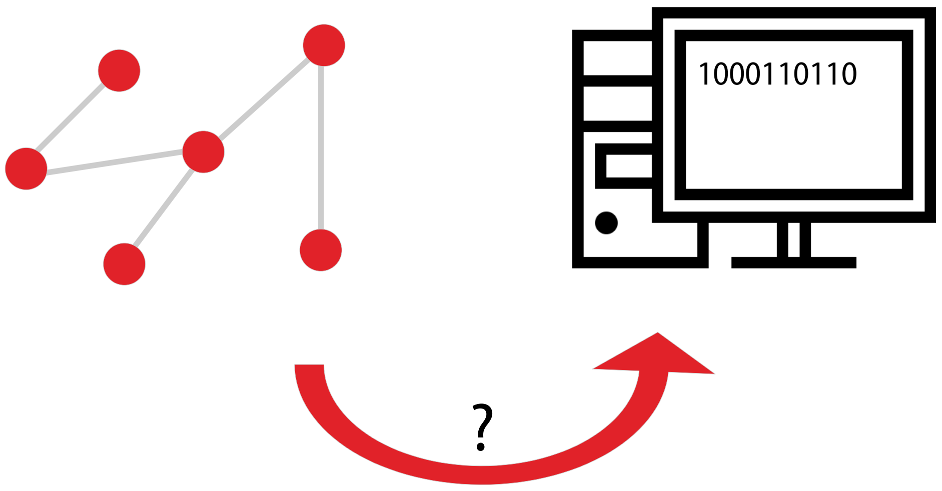
We can also measure the influence of one node on another through the concept of proximity:

- The first-order proximity between two nodes is the weight of the edge that connects them.
- The second-order proximity between two nodes measures the similarity of their neighborhoods (the more neighbors they share, the higher this measure is).
- There are higher-order proximities that can take into account the entire set of possible paths between two nodes.
- The higher the order of proximity, the more global the measure becomes (at the scale of the graph) and the calculation becomes more complex..

Question break

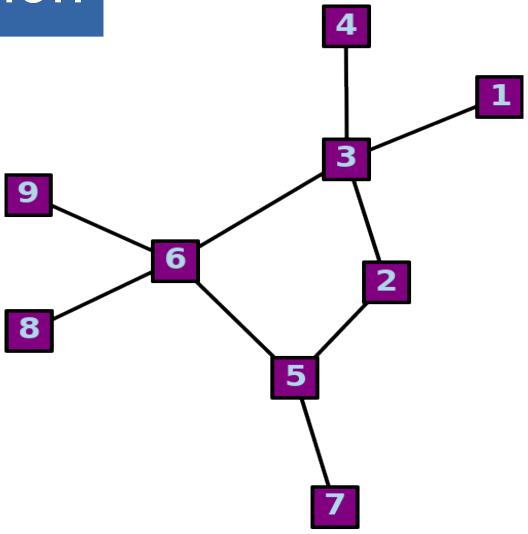


Graph representation



How to represent a graph numerically?

Graph representation



Random numbering of nodes

The nodes of the graph are numbered arbitrarily. Even if there may be a natural way to number them, which can be preferred.

We note that the representation of a graph is not unique.

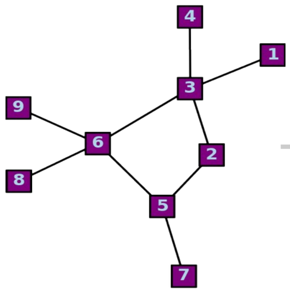
A graph is an object that is invariant under permutation of node numbers.

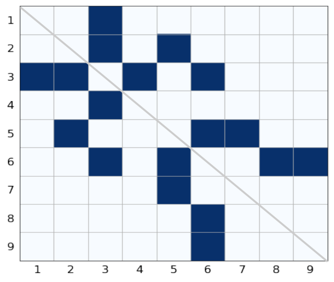
The operations used when manipulating a graph must be invariant under permutation (i.e. must not depend on the order of node numbering).

Graph representation

Adjacency matrix $W_{(i,j)} = \begin{cases} w_{i,j} \, \textit{if there is an edge} \\ 0 \, \textit{if not} \end{cases}$

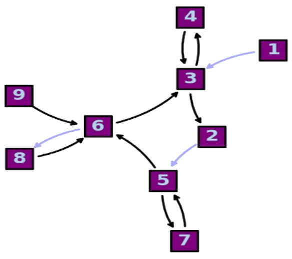
Undirected

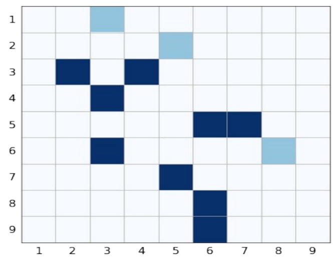




Symmetric

Directed



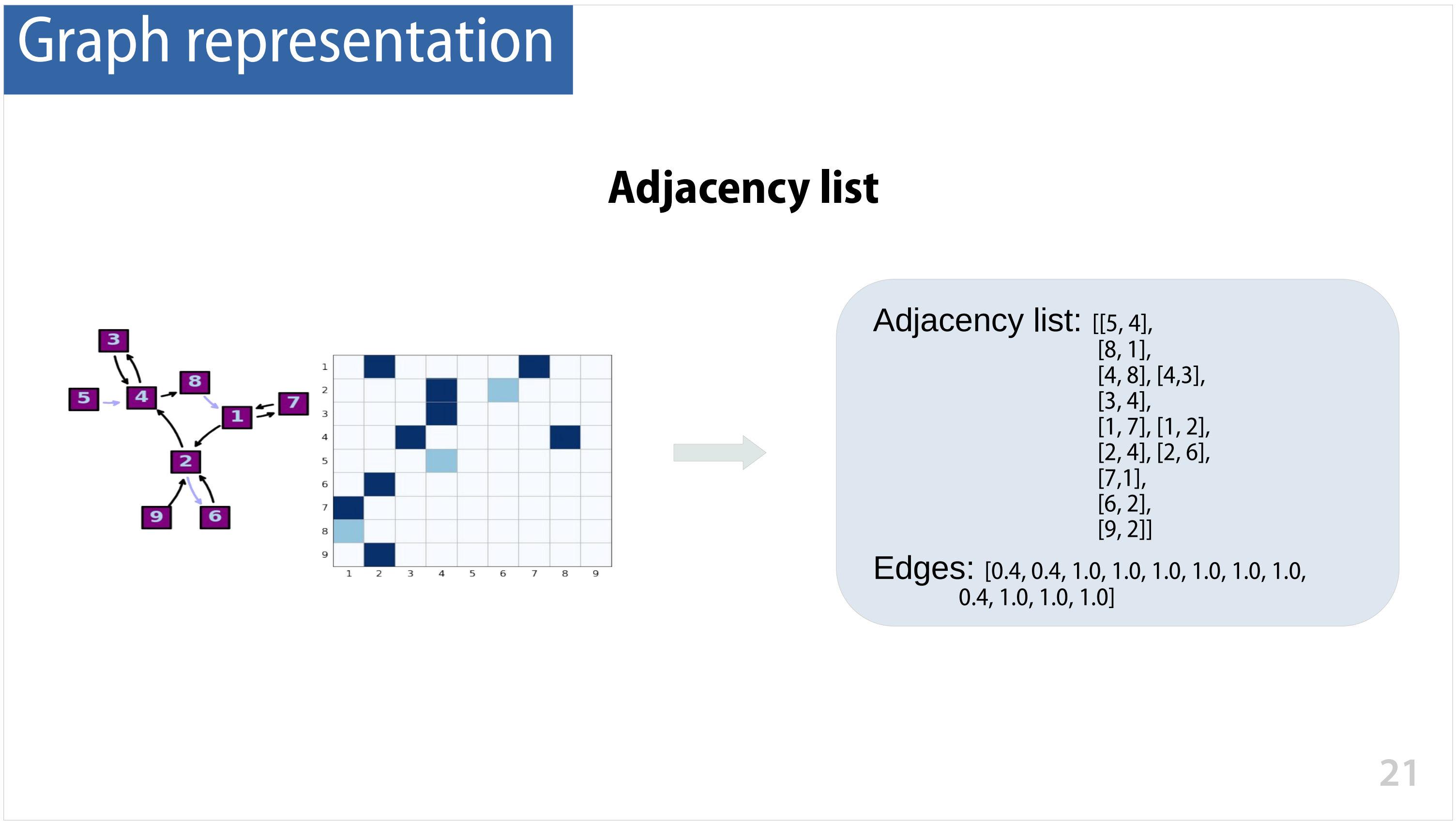


To describe the connectivity of the graph, we can use an adjacency matrix.

We construct a matrix W of size VxV, with w(i,j) non-zero if nodes i and j are connected. Zero otherwise.

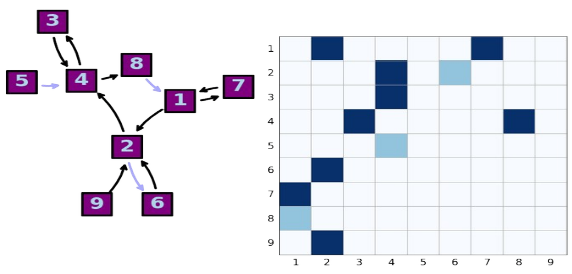
If i and j are connected, the value of w(i,j) is equal to the weight of the edge (equal to 1 if the edge has no weight).

For an undirected graph, W is symmetric. This is not the case for a directed graph.



Graph representation

Adjacency list



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]]

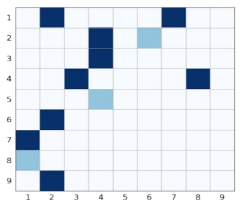
Edges: [0.4, 0.4, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
0.4, 1.0, 1.0, 1.0]

Another way to describe the connectivity of a graph is to define an adjacency list.

We list the edges in a list of 2x1 tensors containing the node numbers of the endpoints.

The edge weights are stored in a separate tensor.

Graph representation



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]]

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]

- Scale V^2 → lot of space
- Might be sparse
- Easy to find an edge

- Scale E → less space
- Might be difficult to find an edge

V = number of nodes/vertices
E = number of edges

https://www.geeksforgeeks.org/comparison-between-adjacency-list-and-adjacency-matrix-representation-of-graph/

Avantages et inconvénients des deux types de représentation.

La matrice d’adjacence prend beaucoup de place en mémoire (VxV). Si elle est creuse, c’est du gâchis.

La liste d’adjacence prend moins de mémoire, à moins que le graphe soit très connecté : liste + features (V x 2 x nb_neighbors + V). Dans ce cas, elle peut prendre plus de mémoire que la matrice.

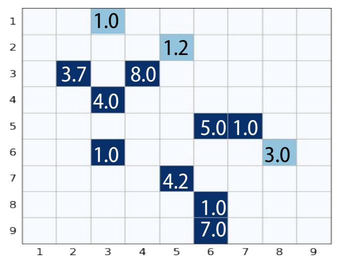
Il est plus facile de retrouver les voisins d’un noeud avec la matrice plutôt qu’avec la liste. Dans un cas, on connait directement l’adresse mémoire de la ligne de la matrice correspondant au noeud qui nous intéresse, dans l’autre il faut parcourir toute la liste.

Le choix de travailler avec la matrice ou la liste n’est pas toujours de notre ressort. C’est en général les librairies qui décident.

Graph representation

- Edge weights are stored either directly in the adjacency matrix, or in an independent tensor.

Adjacency matrix



	1	2	3	4	5	6	7	8	9
1			1.0						
2					1.2				
3		3.7		8.0					
4			4.0						
5						5.0	1.0		
6			1.0					3.0	
7					4.2				
8						1.0			
9						7.0			

```
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]]

Edges: [0.4, 1.4, 2.4, 9.0, 1.0, 5.0, 1.7, 3.0,
        0.4, 1.3, 7.0, 6.2]
```

- Information (features) on nodes and graphs will also be stored in independent tensors.

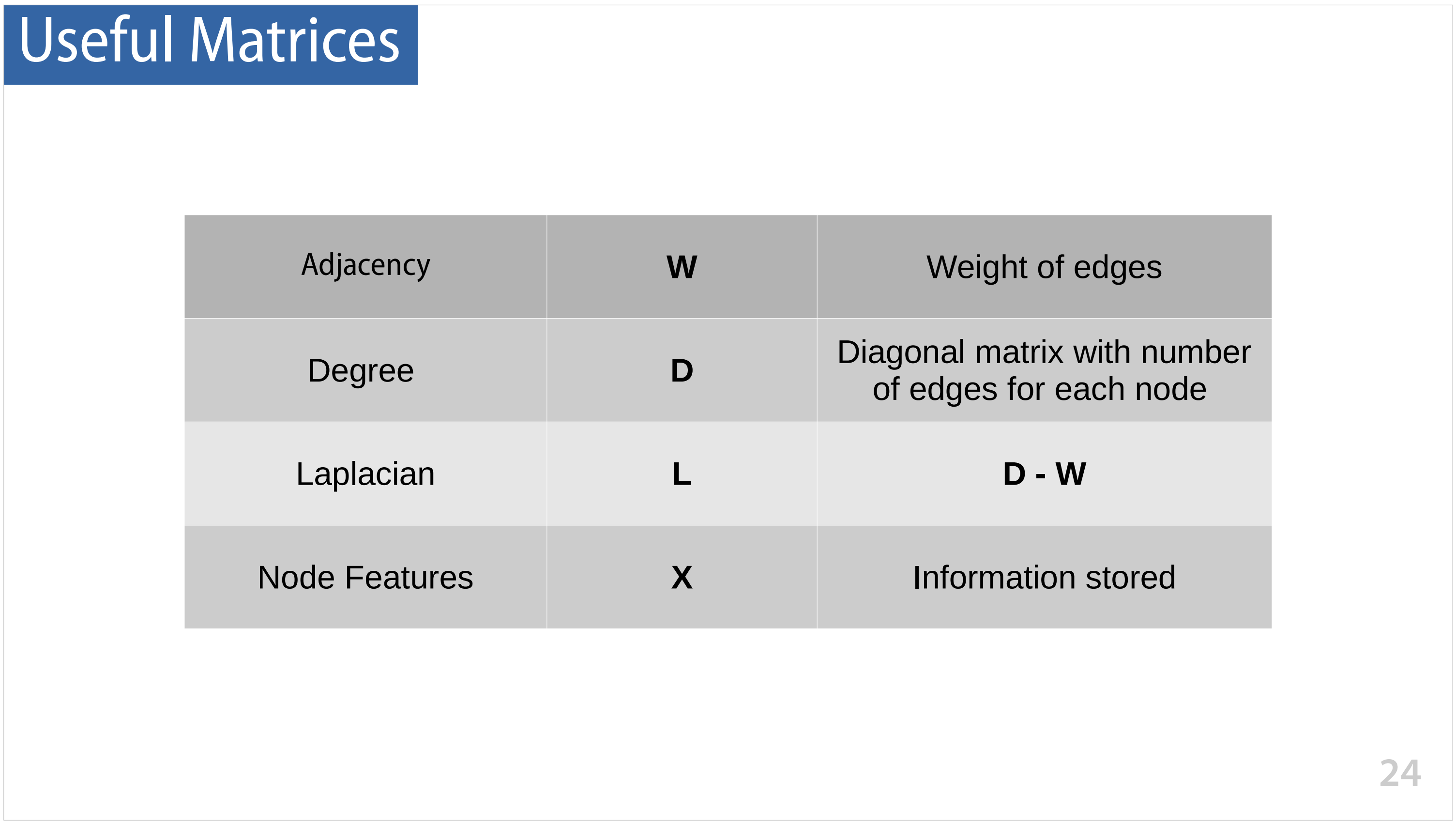
```
Nodes: [4.1, 4.2, 6.4, 1.0, 1.0, 5.0, 1.7,
        3.0, 5.0]

Graph: [8.0]
```

In addition to the edge weights, we need to store the information carried by the nodes and graphs.

This information must be numerical. No problem if it's quantitative, we'll need to build a mapping table <int, string> (or embedding table) if it's qualitative.

Note: An entity (node, edge or graph) can carry one or several features. We can then use multidimensional tensors (tensors of tensors), of size N_entities x N_features.



Useful Matrices

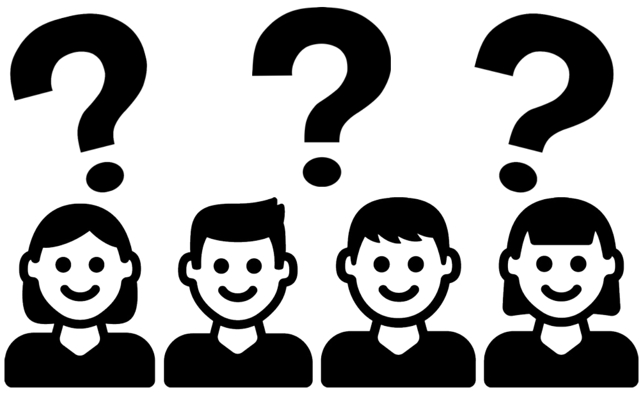
Adjacency	**W**	Weight of edges
Degree	**D**	Diagonal matrix with number of edges for each node
Laplacian	**L**	**D - W**
Node Features	**X**	Information stored

The adjacency matrix is constructed from the edge weights.

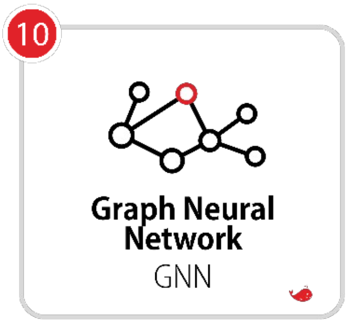
We can build more complex matrices by incorporating node characteristics.

These matrices are used in different information propagation models in the graph. For example, the Laplacian matrix will be used to describe a type of information propagation similar to "heat diffusion".

Question break



Roadmap



9.1 **Graphs are everywhere**

- Complex data structures
- Basics of graph theory

9.2 **Learning on Graphs**

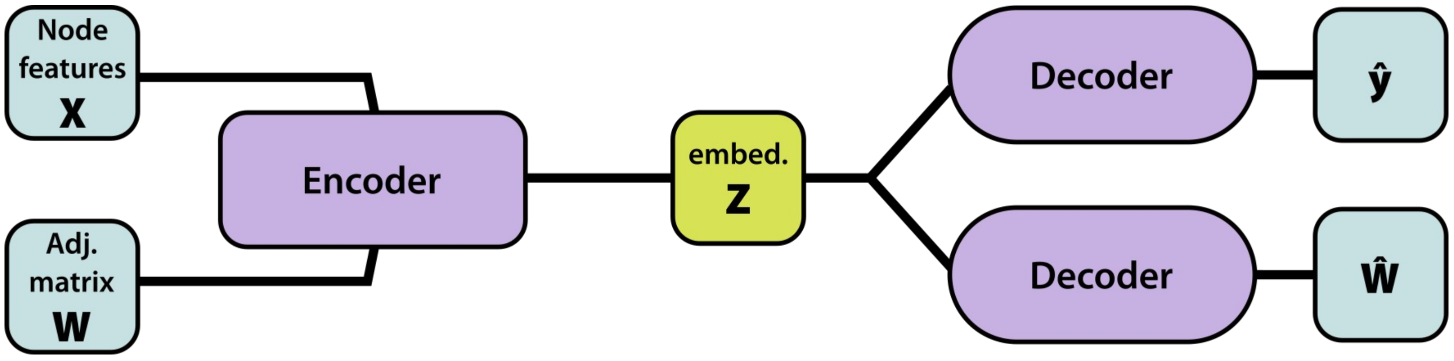
- Graph embedding
- Transductive and inductive learning
- Tasks on graph learning

9.3 **A few examples**

- Taxonomy of methods
- Graph convolution
- Message passing
- Graph Transformer

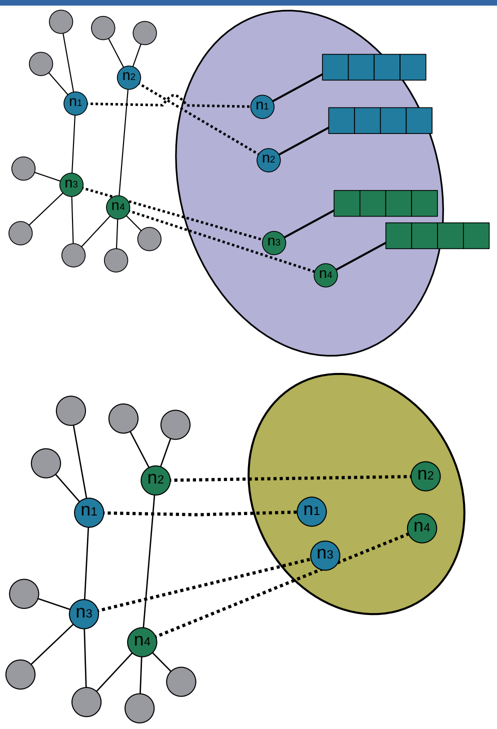
Graph embedding

- We need to find a representation of the graph that is processable

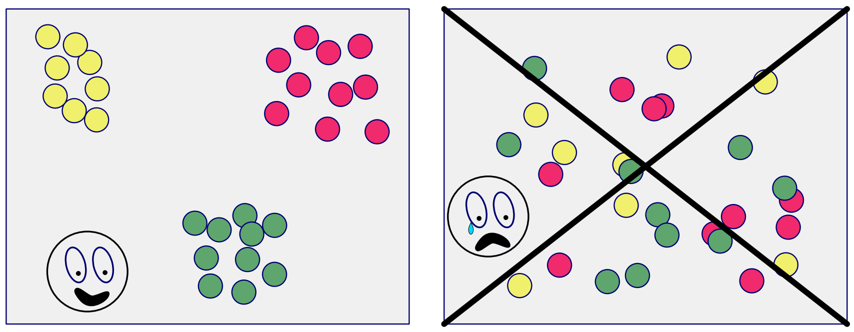


On this slide we present the general shape of the architecture of a model that will be able to learn on graphs.
The first part is en encoder that takes as input the node features (we can also give edges, and graph features).
The encoder creates a dense representation of the information given. This is called an embedding.
Then we use this embedding as an input for a decoder that will either be trained to give some properties or a new adjacency matrix.

Graph embedding

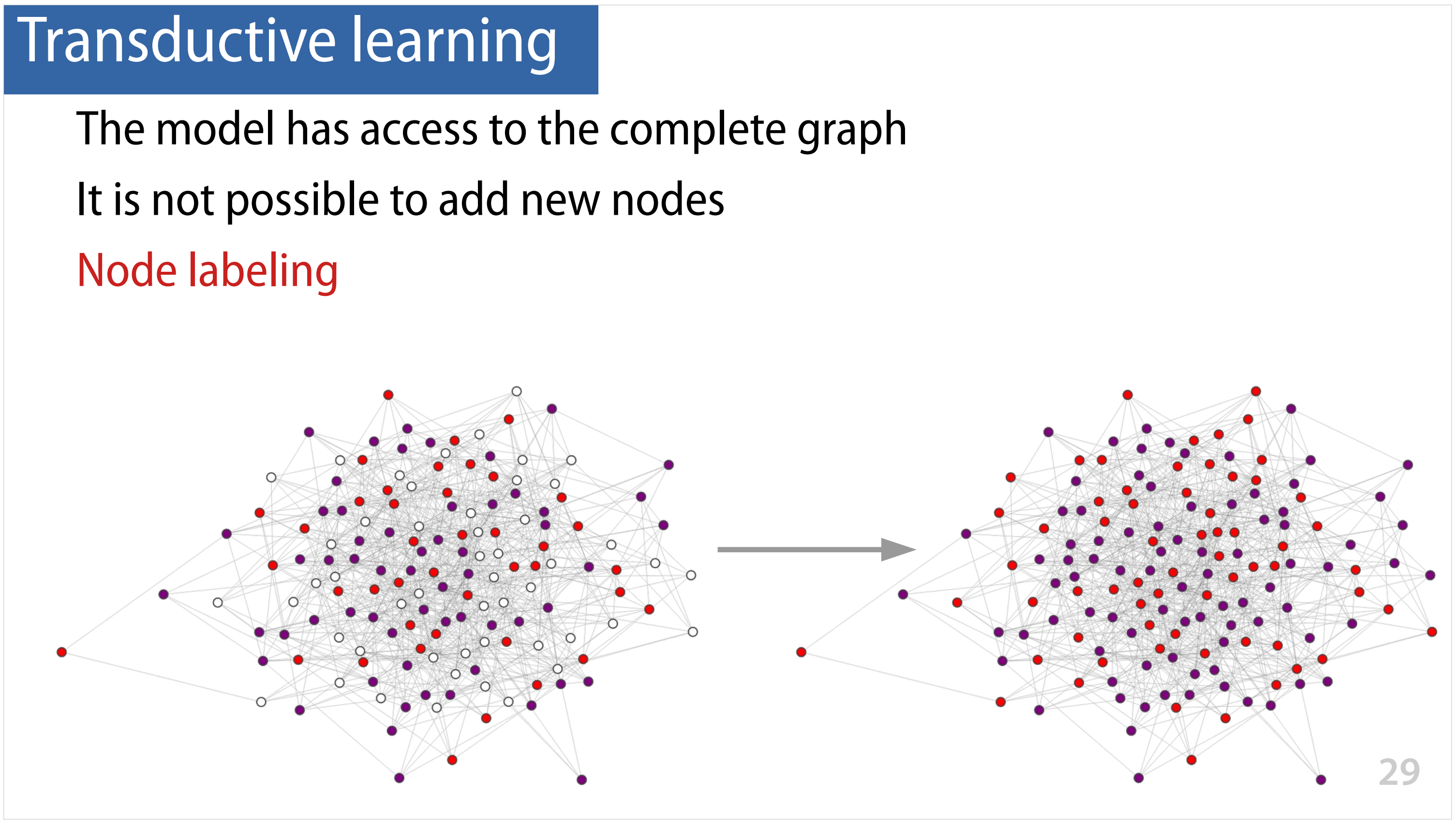


- Features stored in nodes/edges/graphs are not easily processed.
- We transform the features into a vector in the latent space (**Dimension is a hyperparameter**).
- The embedding has to be suited for the task → **Learnable.**



Now a few words on embeddings.

An embedding is a dense representation of the features of the graph. The size of the vector is an hyperarameter of the model that have to be tuned.
The idea of training a model, is to get the best representation possible to solve our problem.
In the first case, we want the nodes that have a similar environment to be close in the embedding space.
The second example is more suited for problems for which the proximity of the nodes are important.

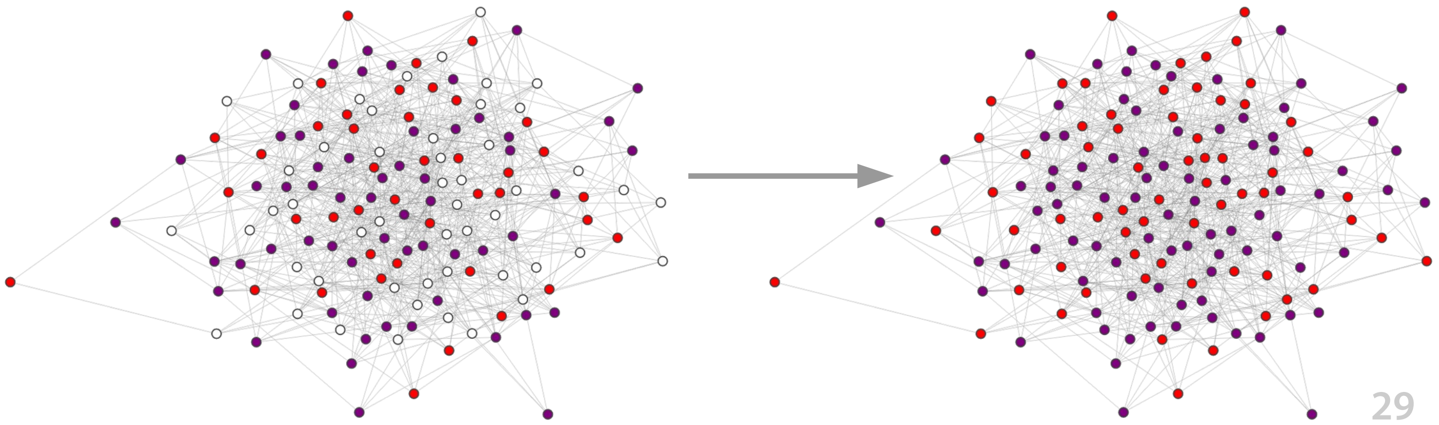


Transductive learning

The model has access to the complete graph

It is not possible to add new nodes

Node labeling



In transductive learning, the problem is restricted to one (usually large) graph. Everything is learned on this graph.

The specificity is that during the training, the nodes (or edges) of the test set are seen by the network.

There are some limitations, as the impossibility to integrate new nodes during the training.

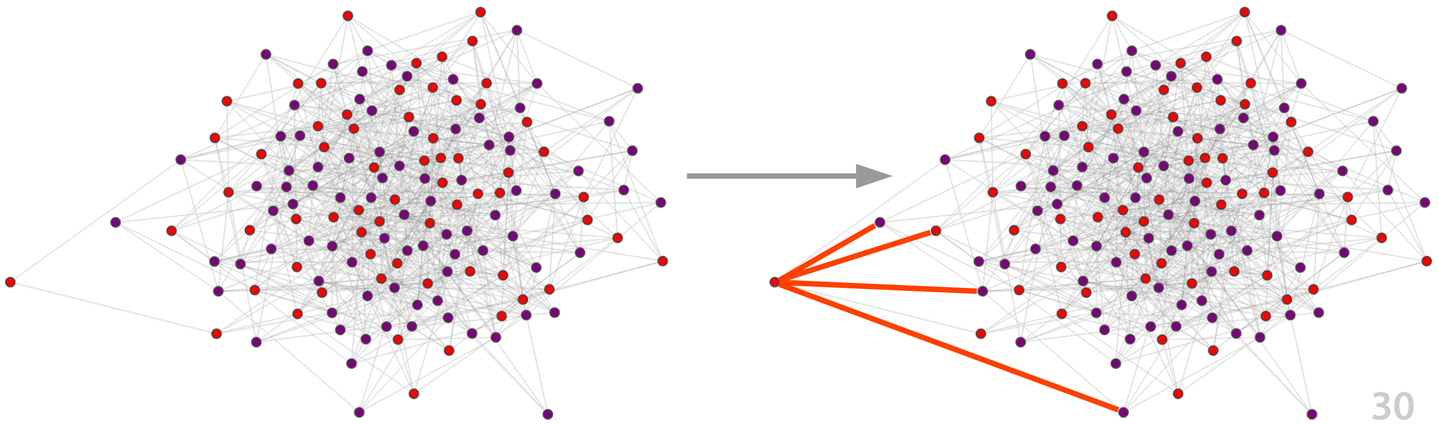
If we take social network, it can be used to :
- Label nodes (find bots in a network)
- Find new edges (give new propositions for contacts)

Transductive learning

The model has access to the complete graph

It is not possible to add new nodes

Find new edges



In transductive learning, the problem is restricted to one (usually large) graph.
Everything is learned on this graph.

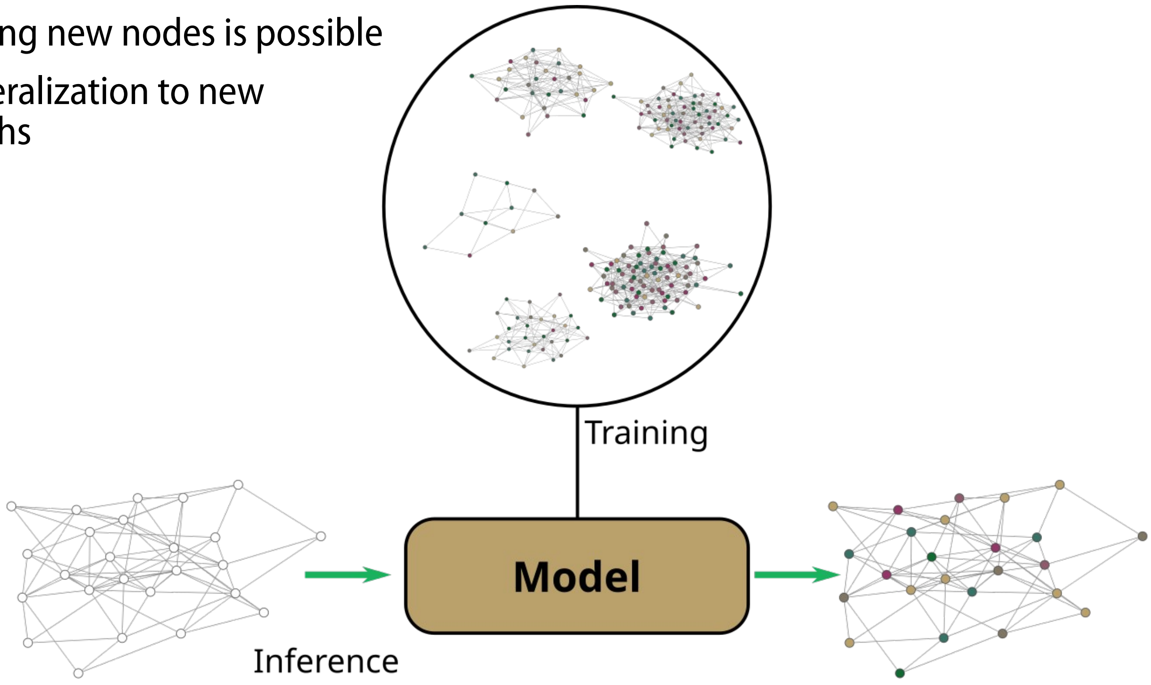
The specificity is that during the training, the nodes (or edges) of the test set are seen by the network.

There are some limitations, as the impossibility to integrate new nodes during the training.

If we take social network, it can be used to :
- Label nodes (find bots in a network)
- Find new edges (give new propositions for contacts)

Inductive learning

- The model has access only to a part of the graph (train set)
- Adding new nodes is possible
- Generalization to new graphs



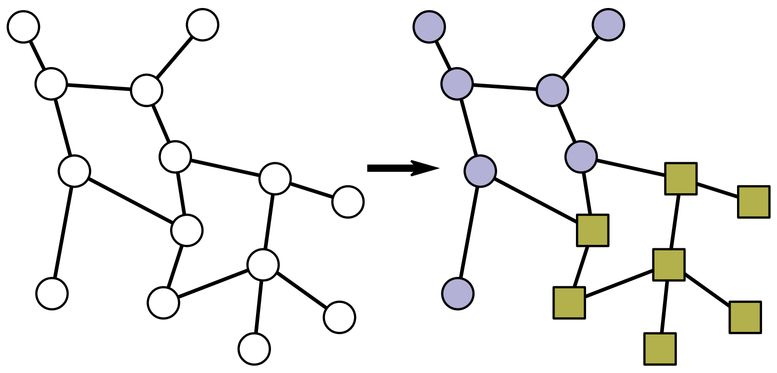
Inductive learning is closer that what we have already seen during the training.

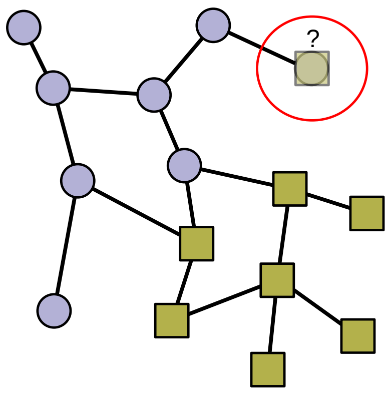
We have a training set with a lot of graphs from which the model learns it parameters. Then this model is used for inference.

It is possible to have graphs with different sizes and the model will be able (hopefully) to generalize on new graphs.

Tasks on nodes

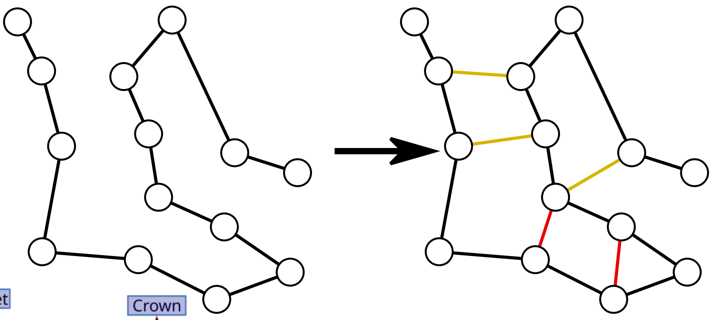
- Labeling nodes in a graph (clustering)
  - Find topic of a research paper (CORA, etc)
  - Find bots in a social network
  - …
- Labeling new nodes
- Perform regression

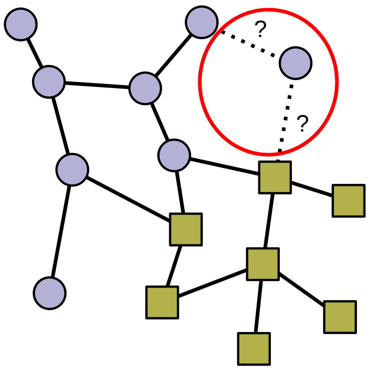


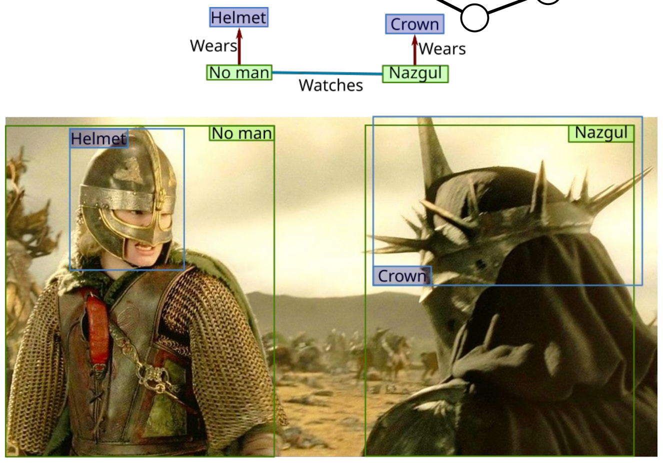


Tasks on edges

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



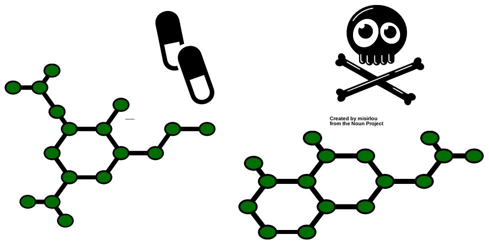


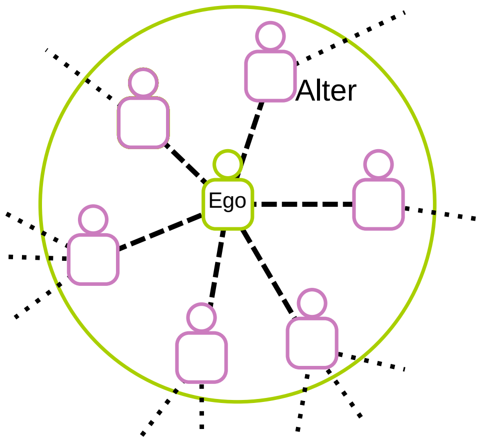


G. Zhu et al., “Scene Graph Generation: A Comprehensive Survey.” arXiv, Jun. 22, 2022. doi: 10.48550/arXiv.2201.00443.

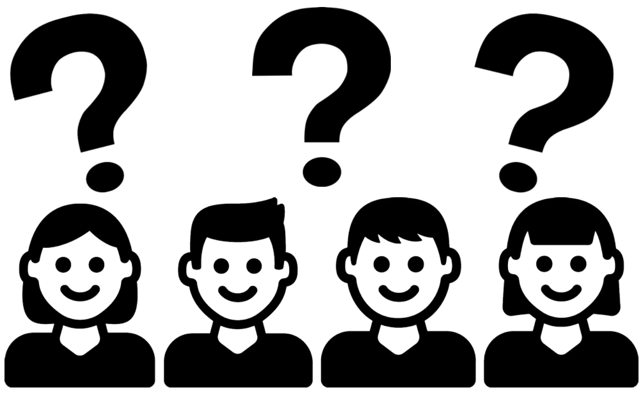
Tasks on graphs

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

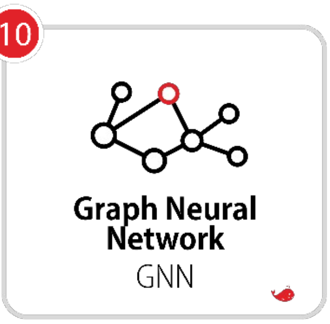




Question break



Roadmap



9.1 **Graphs are everywhere**

- Complex data structures
- Basics of graph theory

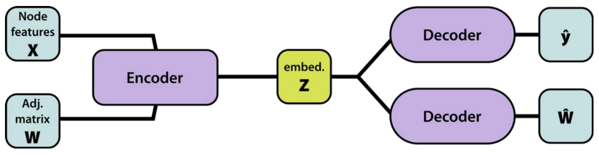
9.2 **Learning on Graphs**

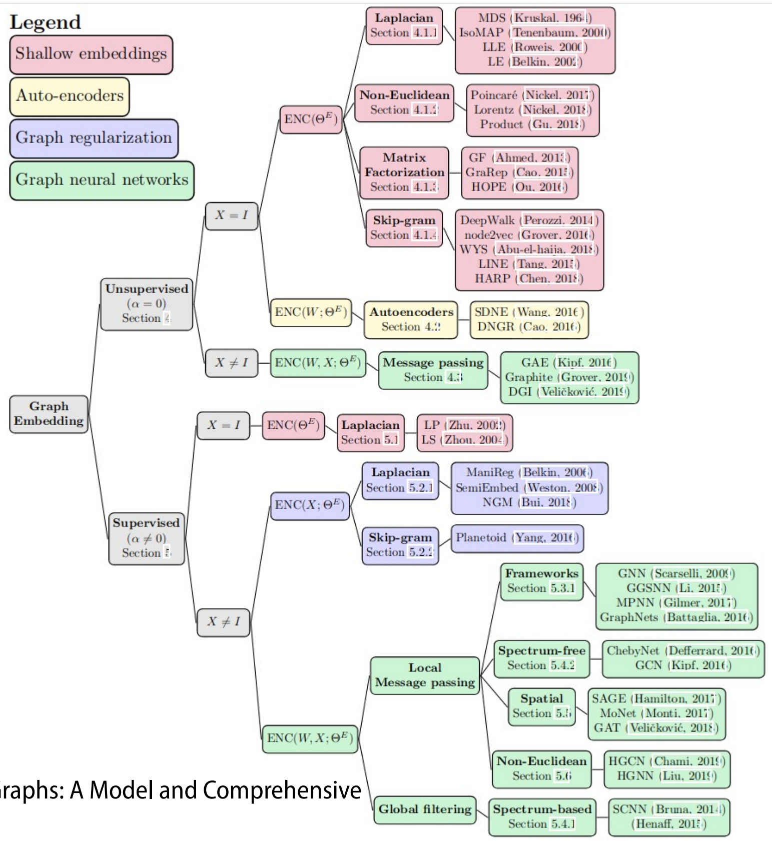
- Graph embedding
- Transductive and inductive learning
- Tasks on graph learning

9.3 **A few examples**

- Taxonomy of methods
- Graph convolution
- Message passing
- Graph Transformer

Taxonomy of methods





Legend
Shallow embeddings
Auto-encoders
Graph regularization
Graph neural networks
Graph Embedding
Unsupervised (α = 0) Section 4
Supervised (α ≠ 0) Section 5
X = I
X ≠ I
ENC(Θ^E)
ENC(W; Θ^E)
ENC(W, X; Θ^E)
ENC(X; Θ^E)
Laplacian Section 4.1.1
MDS (Kruskal, 1964) IsoMAP (Tenenbaum, 2000) LLE (Roweis, 2000) LE (Belkin, 2002)
Non-Euclidean Section 4.1.2
Poincaré (Nickel, 2017) Lorentz (Nickel, 2018) Product (Gu, 2018)
Matrix Factorization Section 4.1.3
GF (Ahmed, 2013) GraRep (Cao, 2015) HOPE (Ou, 2016)
Skip-gram Section 4.1.4
DeepWalk (Perozzi, 2014) node2vec (Grover, 2016) WYS (Abu-el-haija, 2018) LINE (Tang, 2015) HARP (Chen, 2018)
Autoencoders Section 4.2
SDNE (Wang, 2016) DNGR (Cao, 2016)
Message passing Section 4.3
GAE (Kipf, 2016) Graphite (Grover, 2019) DGI (Veličković, 2019)
Laplacian Section 5.1
LP (Zhu, 2002) LS (Zhou, 2004)
Laplacian Section 5.2.1
ManiReg (Belkin, 2006) SemiEmbed (Weston, 2008) NGM (Bui, 2018)
Skip-gram Section 5.2.2
Planetoid (Yang, 2016)
Frameworks Section 5.3.1
GNN (Scarselli, 2009) GGSNN (Li, 2015) MPNN (Gilmer, 2017) GraphNets (Battaglia, 2016)
Local Message passing
Spectrum-free Section 5.4.2
ChebyNet (Defferrard, 2016) GCN (Kipf, 2016)
Spatial Section 5.5
SAGE (Hamilton, 2017) MoNet (Monti, 2017) GAT (Veličković, 2018)
Non-Euclidean Section 5.6
HGCN (Chami, 2019) HGNN (Liu, 2019)
Global filtering
Spectrum-based Section 5.4.1
SCNN (Bruna, 2014) (Henaff, 2015)

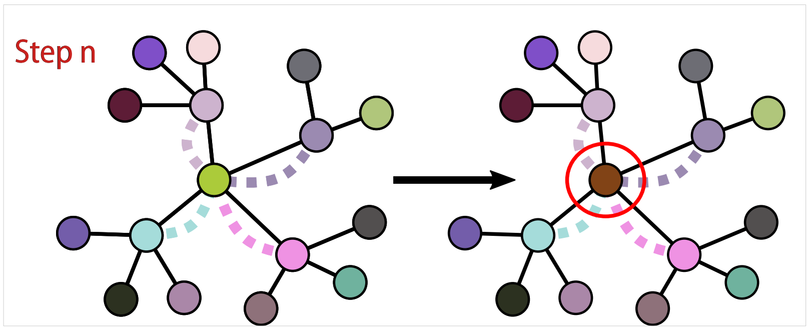
I. Chami, S. Abu-El-Haija, and B. Perozzi, "Machine Learning on Graphs: A Model and Comprehensive Taxonomy".

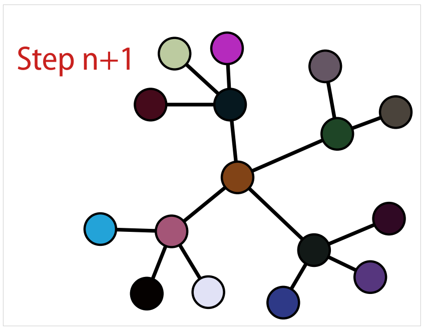
Chami and collaborators have written an interesting review of the different kinds of methods being used for learning on graph.

Depending on the task you are doing, you need to find the correct methodology.

Graph convolution

- Just like for images we can learn from neighborhood with a convolution.





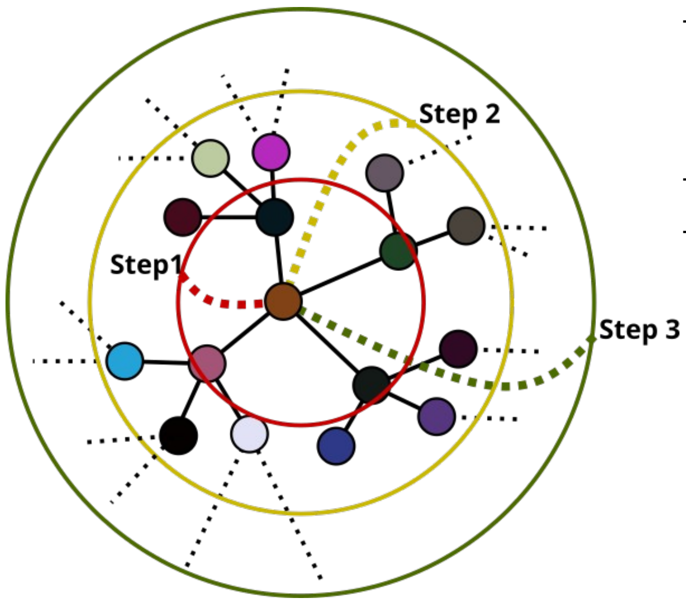
- A bit more complex since the number of neighbors is unlikely to be constant.
- We want the operator to be permutation invariant.

In a similar way as for pictures, we can define a convolution for graph.
The main difference is due to the fact that we do not have a rigid structure and we have to find the neighbors. It is done by using the adjacency matrix which holds this information.

The value of the node is updated with the values of the neighbors with an operator that is permutation invariant.
We can choose for example the sum operator.

At one step all the nodes get their values updated.

Graph convolution



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

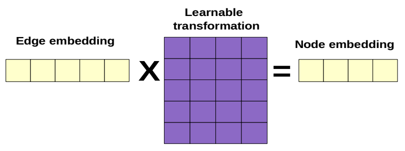
To get information of other nodes than the neighbors, we can make several steps of convolution.
Each additional convolution will get information for neighbors that are one step further in the path.

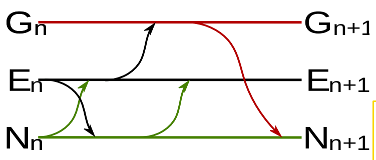
An hyperparameter to choose is the number of convolution to perform. When this number increases, the calculation becomes more and more heavy. We can choose a cutoff.

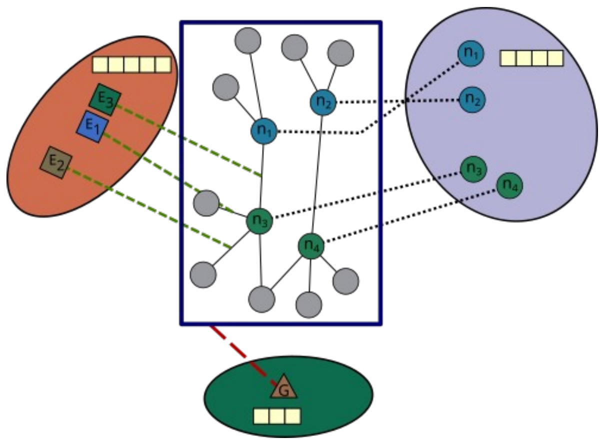
Increasing the number of convolution step can lead to a problem of information dilution. We get a lot of information from the closest neighbors but this decreases with the distance in the path,
To overcome this problem, it is possible to use a virtual node which is connected to all other node and serves as a shortcut to get the information. We still need to be careful because the cost increases with the size of the graph.

Message passing

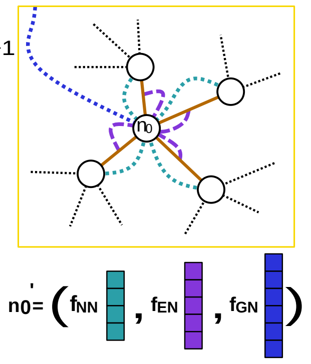
- We have embeddings for each part of the graph (possibly different vector sizes).
- Each part can learn from the others via a transformation.







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



Here we present an algorithm useful for passing information from different elements of the graphs (nodes, graph and edges).

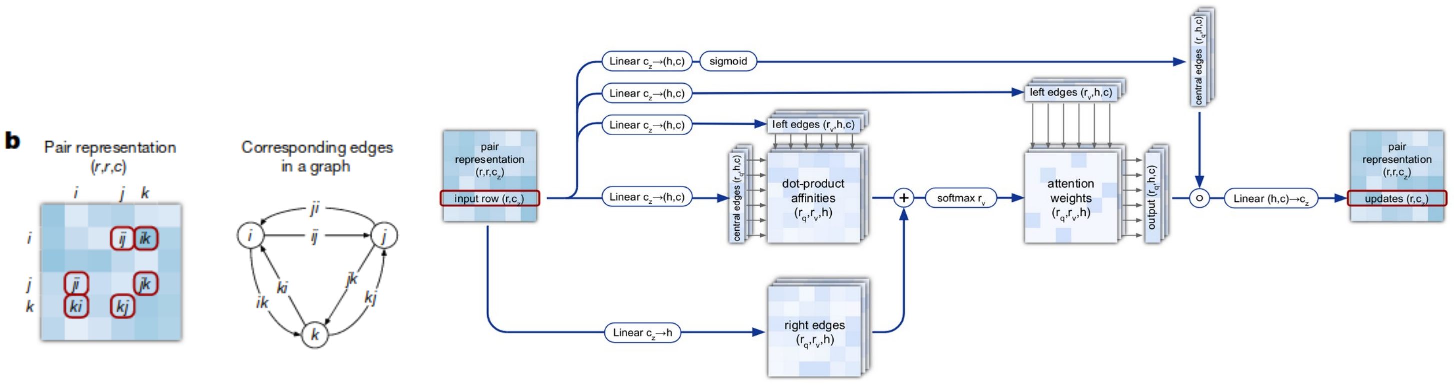
We presented the node embedding in a precedent part of the course. This is a way to condense the information on the node.
Here we want to get information from edges and the graph. As for the nodes we need to have the information encoded in a dense vector.
The embedding size for the edges and for the graph are also hyperparameter and do not have to be equal to the node embedding size.

At the end the value of the node will have the size of the embedding space of the nodes. It is necessary to define a transformation for the other parts so that they are compatible with the nodes.
To do so we define a matrix with learnable values that will transform an, for instance, edge information to something that can be understood by the nodes.

Once we have all the transformations done, the message (or value) of the node that will be sent to the other nodes is an aggregation of all the vectors.

It is possible to share information between all the parts of the graph by defining a correct transformation.

Alphafold transformer



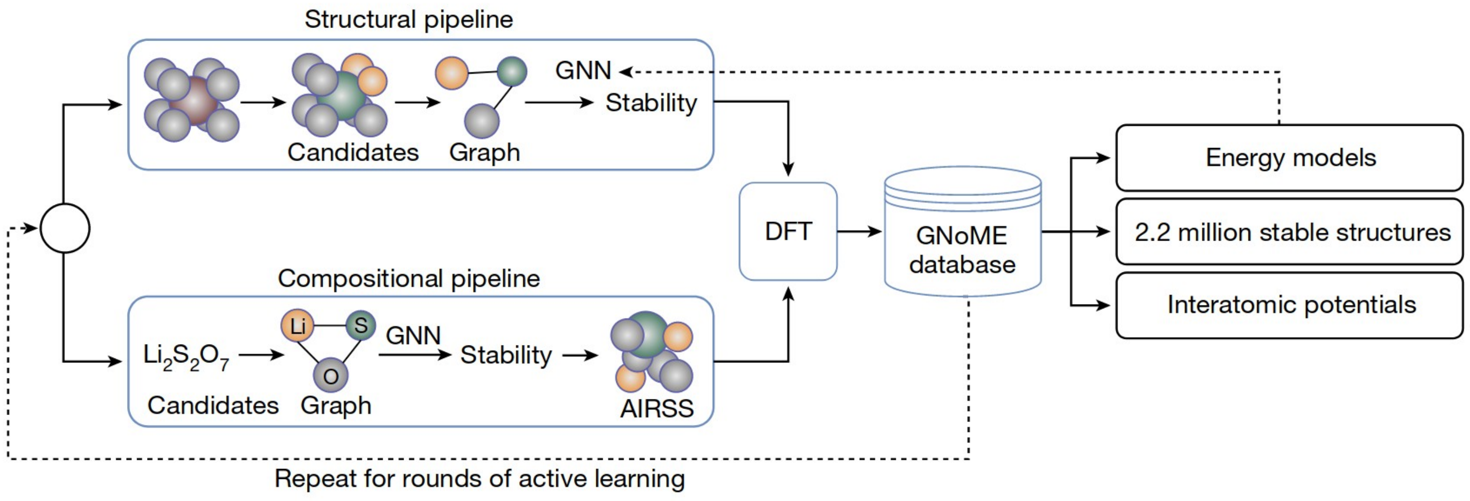
Supplementary Figure 7 | Triangular self-attention around starting node. Dimensions: r: residues, c: channels, h: heads

J. Jumper et al., Highly accurate protein structure prediction with AlphaFold, Nature, vol. 596, no. 7873, Art. no. 7873, Aug. 2021, doi: 10.1038/s41586-021-03819-2.

An example of use of graph in Alphafold.
It is a model to find the structure of proteins for the sequence of aminoacids.

GNoME

Generation of novel crystal structures

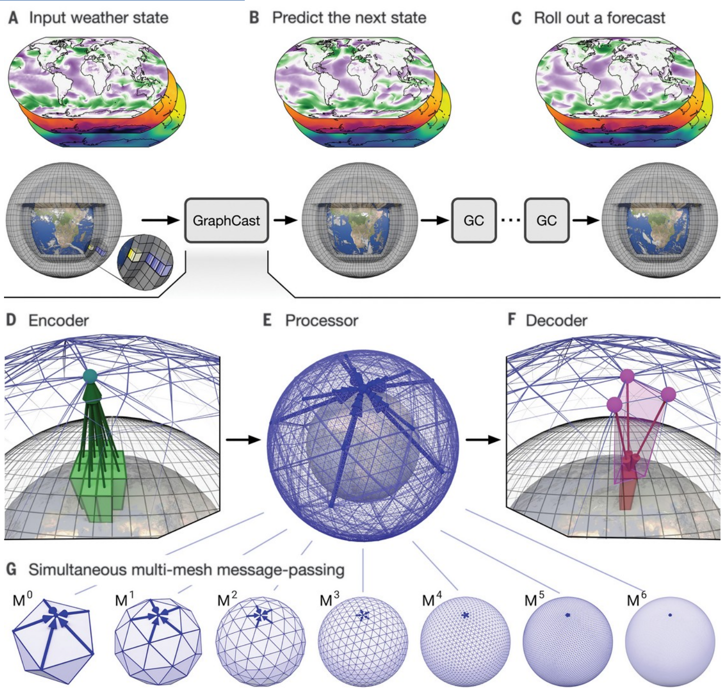


Merchant, A., Batzner, S., Schoenholz, S.S. et al. Scaling deep learning for materials discovery. Nature 624, 80–85 (2023). https://doi.org/10.1038/s41586-023-06735-9

▪

GraphCast

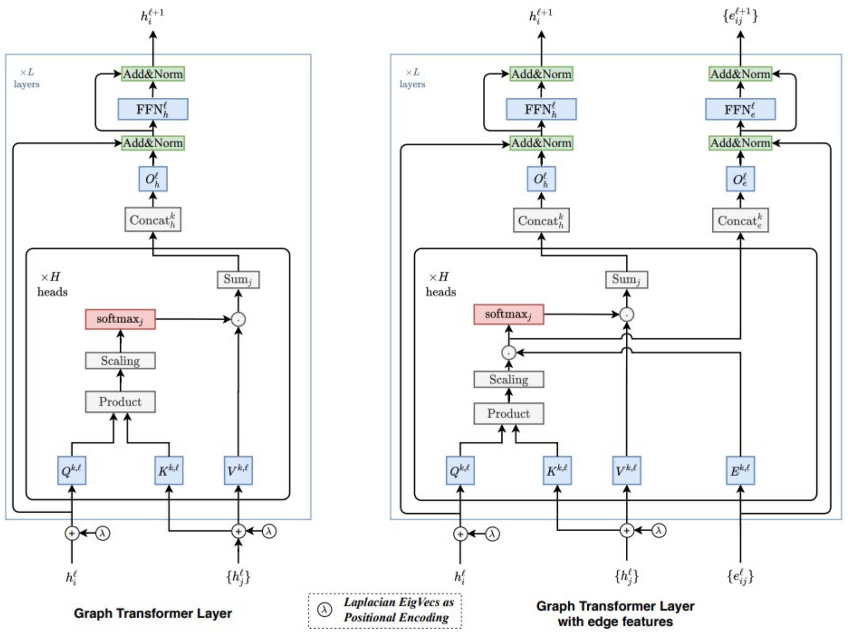
Prediction of the weather with temporal graphs



Remi Lam et al. Learning skillful medium-range global weather forecasting.Science382,1416-1421(2023).DOI:10.1126/science.adi2336

▪

Graph Transformer Network

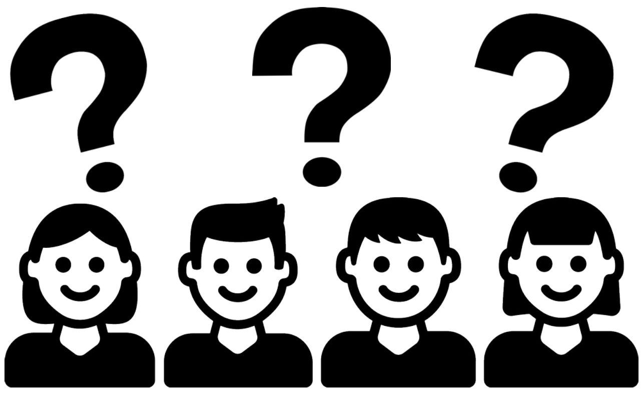


Graph Transformer Layer
Laplacian EigVecs as Positional Encoding
Graph Transformer Layer with edge features

Dwivedi, Bresson A Generalization of Transformer Networks to Graphs 2020, https://arxiv.org/abs/2012.09699

It is possible to add attention to our graph models.

Question break



Resources

Libraries

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

→ https://logconference.org/
→ https://ogb.stanford.edu/

Tutorials

→ https://antoniolonga.github.io/Pytorch_geometric_tutorials/
→ https://docs.dgl.ai/tutorials/blitz

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

- **Chami, S. Abu-El-Haija, and B. Perozzi, "Machine Learning on Graphs: A Model and Comprehensive Taxonomy".**
- 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).
- Dwivedi, Bresson "A Generalization of Transformer Networks to Graphs" 2020, https://arxiv.org/abs/2012.09699
- G. Zhu et al., "Scene Graph Generation: A Comprehensive Survey." arXiv, Jun. 22, 2022. doi: 10.48550/arXiv.2201.00443
- Merchant, A., Batzner, S., Schoenholz, S.S. et al. Scaling deep learning for materials discovery. Nature 624, 80–85 (2023). https://doi.org/10.1038/s41586-023-06735-9
- Remi Lam et al. Learning skillful medium-range global weather forecasting.Science382,1416-1421(2023).DOI:10.1126/science.adi2336