A short intro to Graph Neural Networks

Deep Learning
Master degree in Computer Science

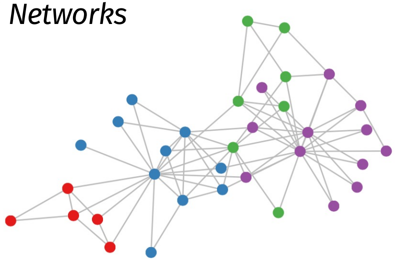
Nicoletta Noceti
Machine Learning Genoa Center – University of Genoa

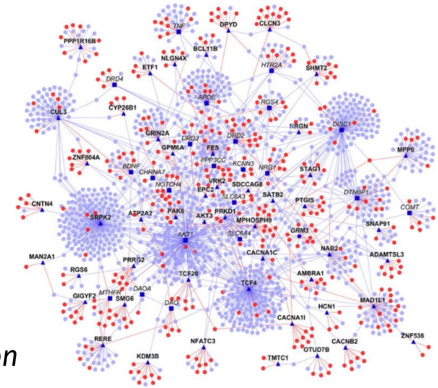
Material

- https://distill.pub/2021/gnn-intro/
- https://www.cs.ubc.ca/~lsigal/532S_2018W2/Lecture18a.pdf
- https://disi.unitn.it/~passerini/teaching/2021-2022/AdvancedTopicsInMachineLearning/slides/GNN/talk.pdf

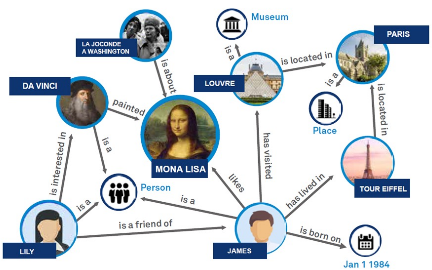
Graphs are everywhere

Social Networks

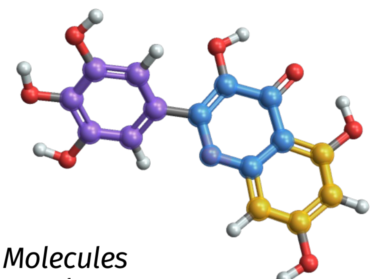




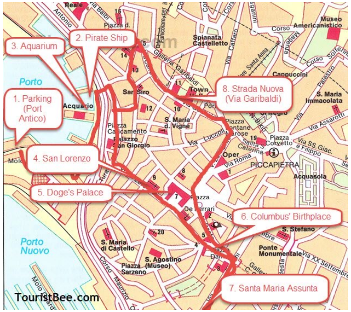
Proteins interaction Networks



Knowledge Graph



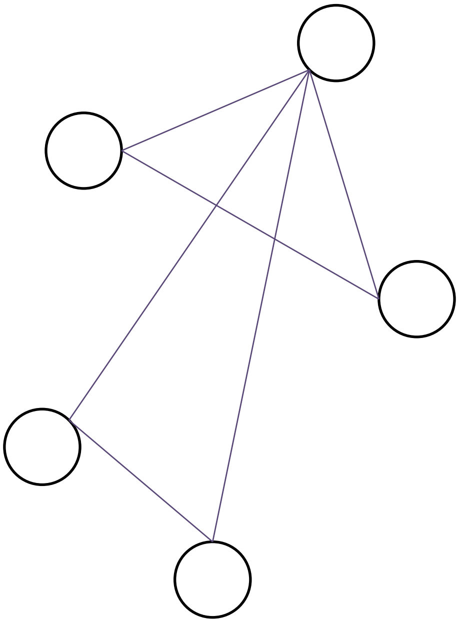
Molecules Graph



Road Maps

But also images can be seen as graphs…

A refresh on graphs



G = (V, E)

V is the set of nodes

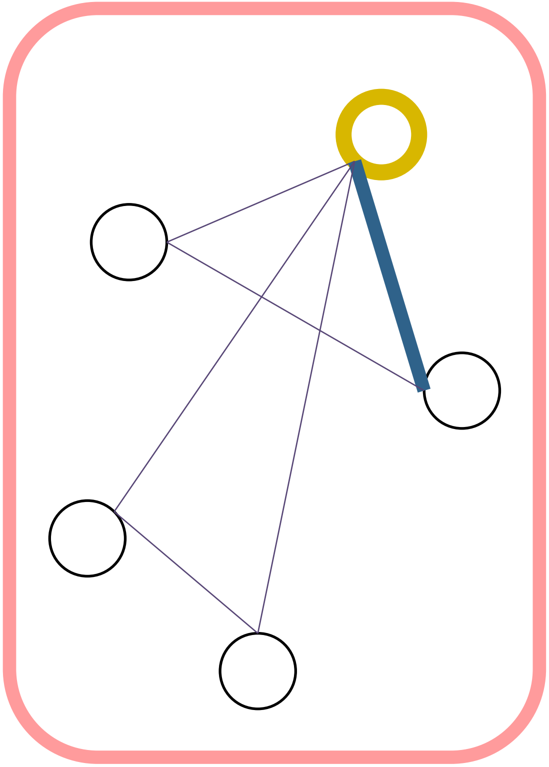
E is the set of edges

Nodes and edges can have attributes

We might also have global graph attributes U

Edges can be directed or not

A refresh on graphs



Node embedding

Edge embedding

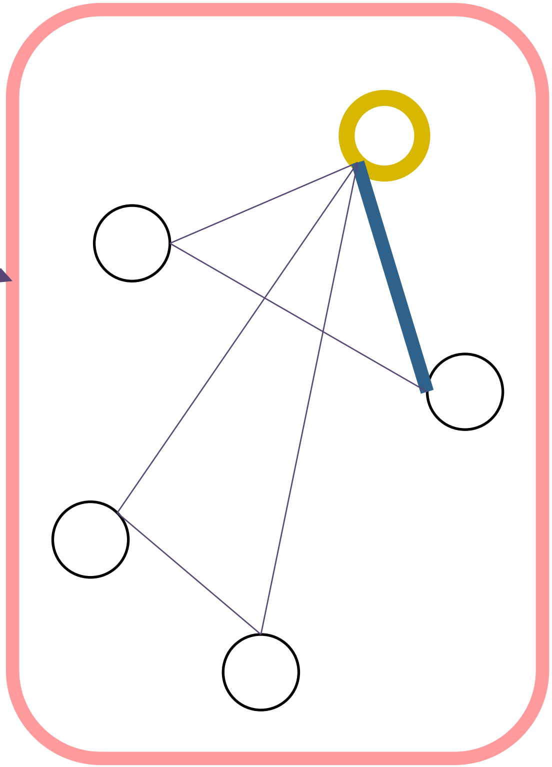
Global embedding

Graph-based learning problems

Graph-level tasks

Goal: predicting a property of an entire graph

→ Analogous to image classification



Examples:

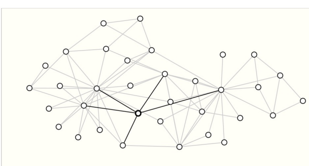
- The graph contains a certain substructure
- The graph is an instance of a certain class

Graph-based learning problems

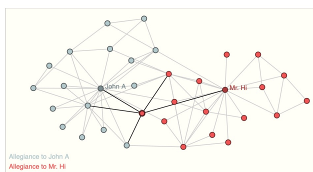
Node-level tasks

Goal: predicting the identity or role of a node

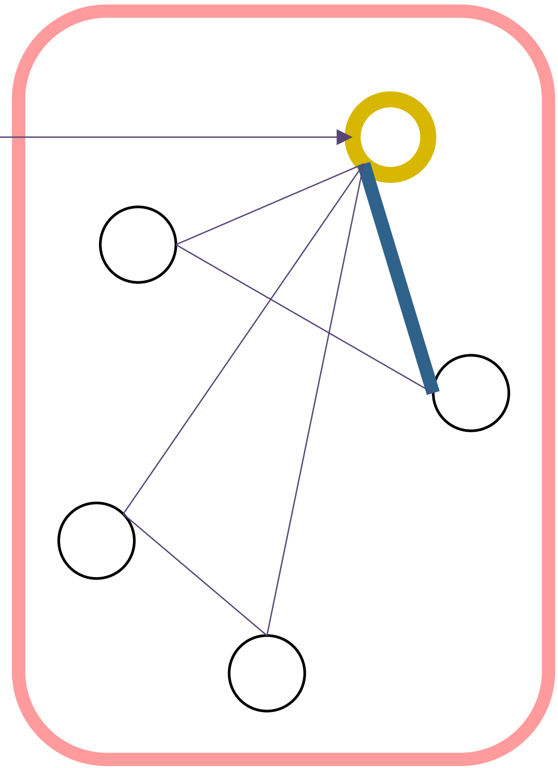
→ Analogous to image segmentation



Input: graph with unlabled nodes



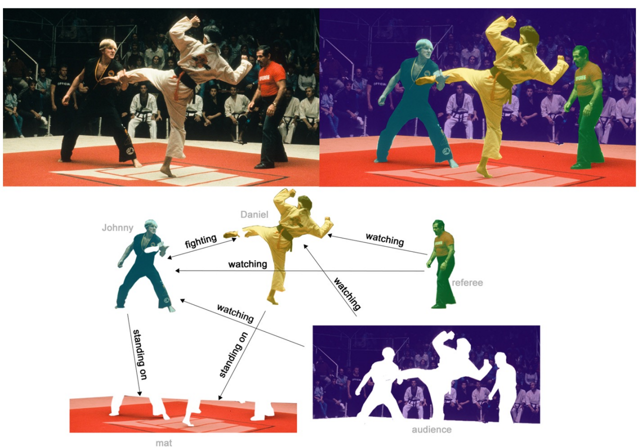
Output: graph node labels

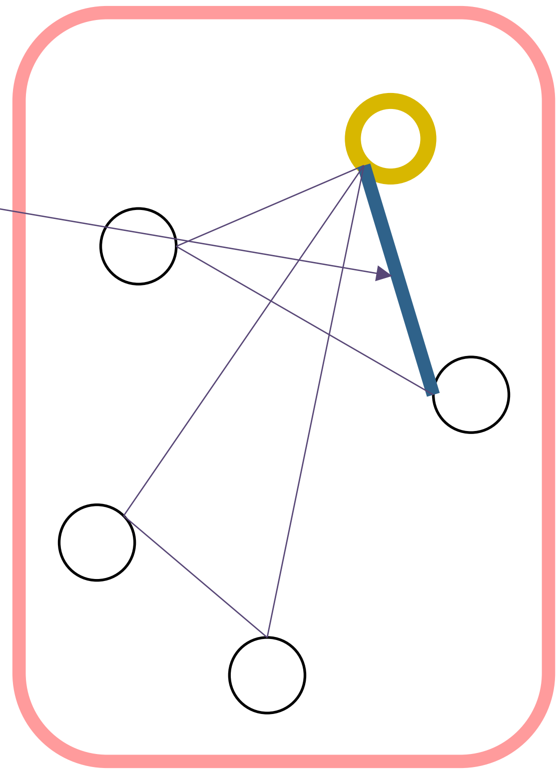


Graph-based learning problems

Edge-level tasks

Goal: predicting the identity or role of an edge





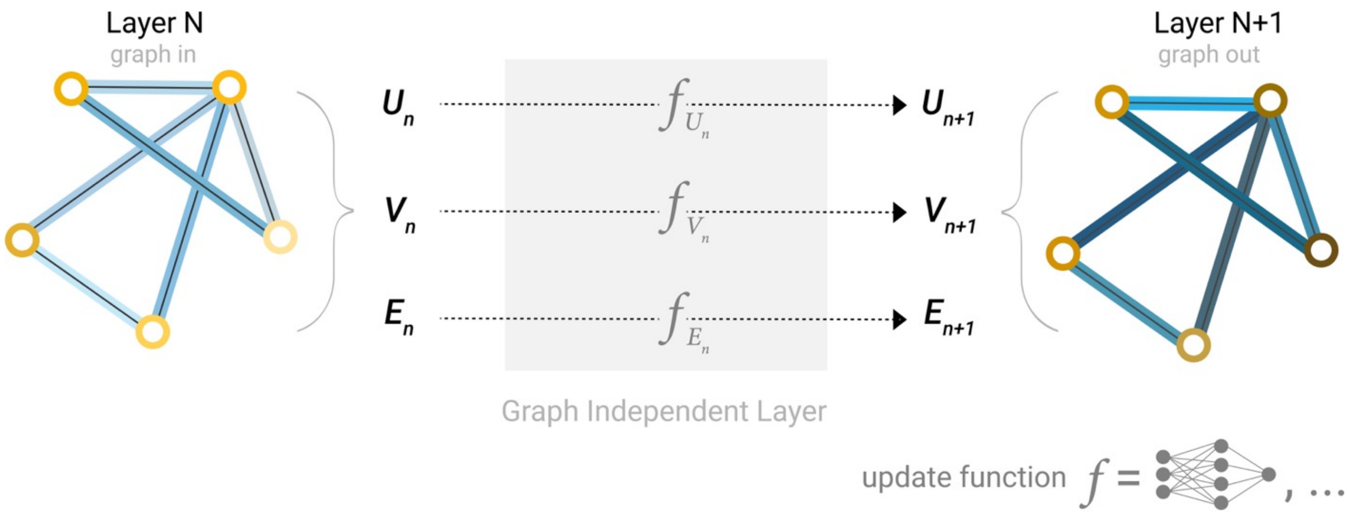
Graph Neural Network

A definition

- A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances)
- GNNs adopt a “graph-in, graph-out” architecture, with information loaded into its nodes, edges and global-context, and progressively transforms these embeddings, without changing the connectivity of the input graph

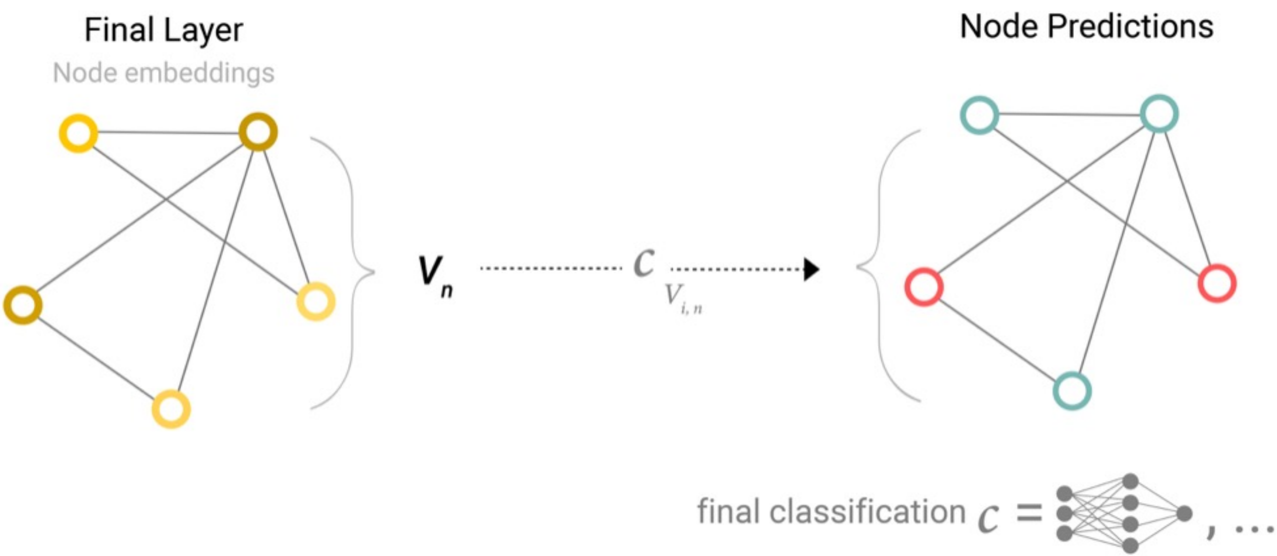
How to encode graphs

Simplest GNN: in each layer, new embeddings are learnt for nodes, edges and the graph, without relying on the graph connectivity



How to make predictions?

If the node already contains information (i.e. it has an embedding) one may simply apply a classifier to each node embedding



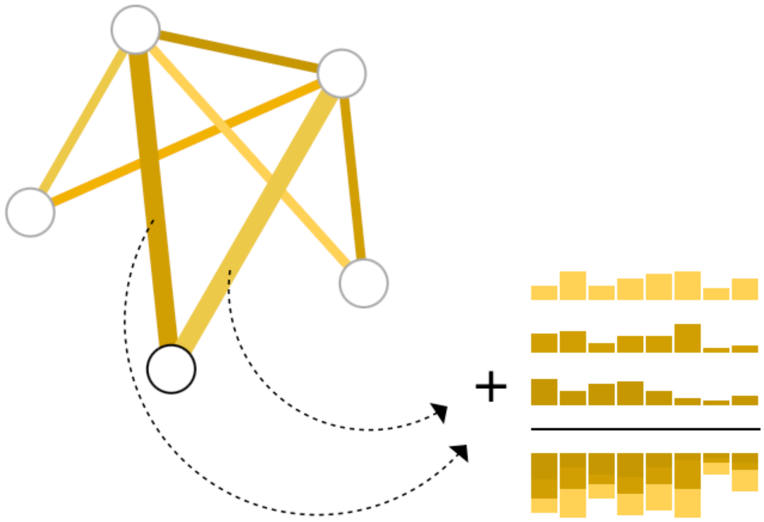
How to make predictions?

However, sometimes nodes do not contain information but still we need to make predictions on them... how to do it?

How to make predictions?

However, sometimes nodes do not contain [enough] information but still, we need to make predictions about them... how to do it?

... By exploiting the edges, with a pooling

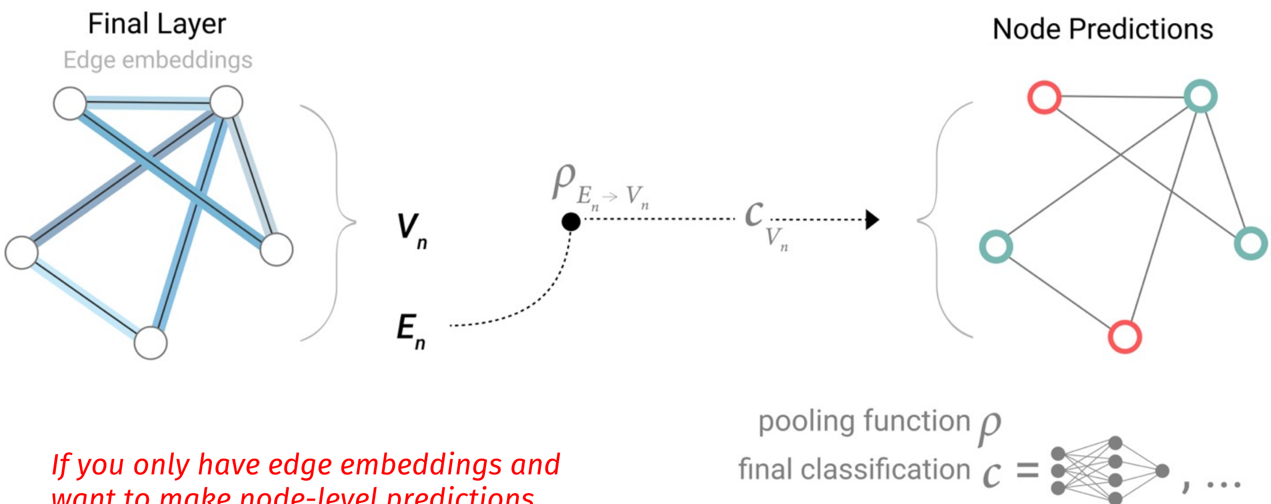


UniGe | MaLGa

How to make predictions?

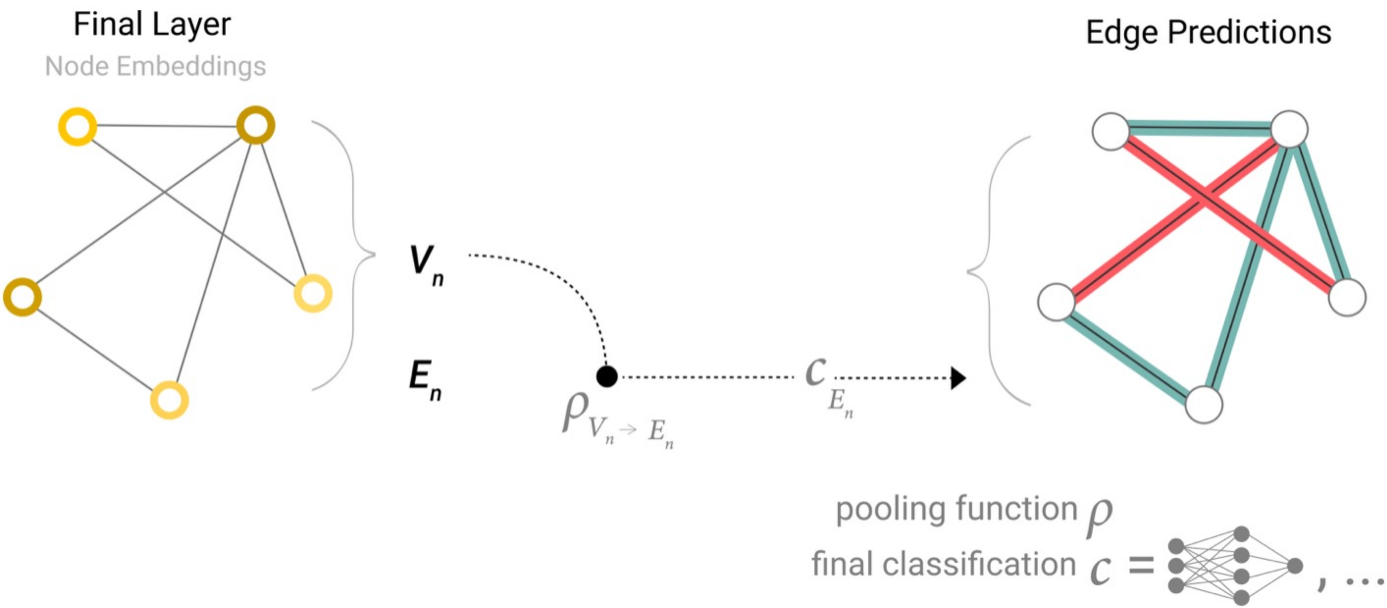
However, sometimes nodes do not contain [enough] information but still, we need to make predictions about them... how to do it?

... By exploiting the edges, with a **pooling**



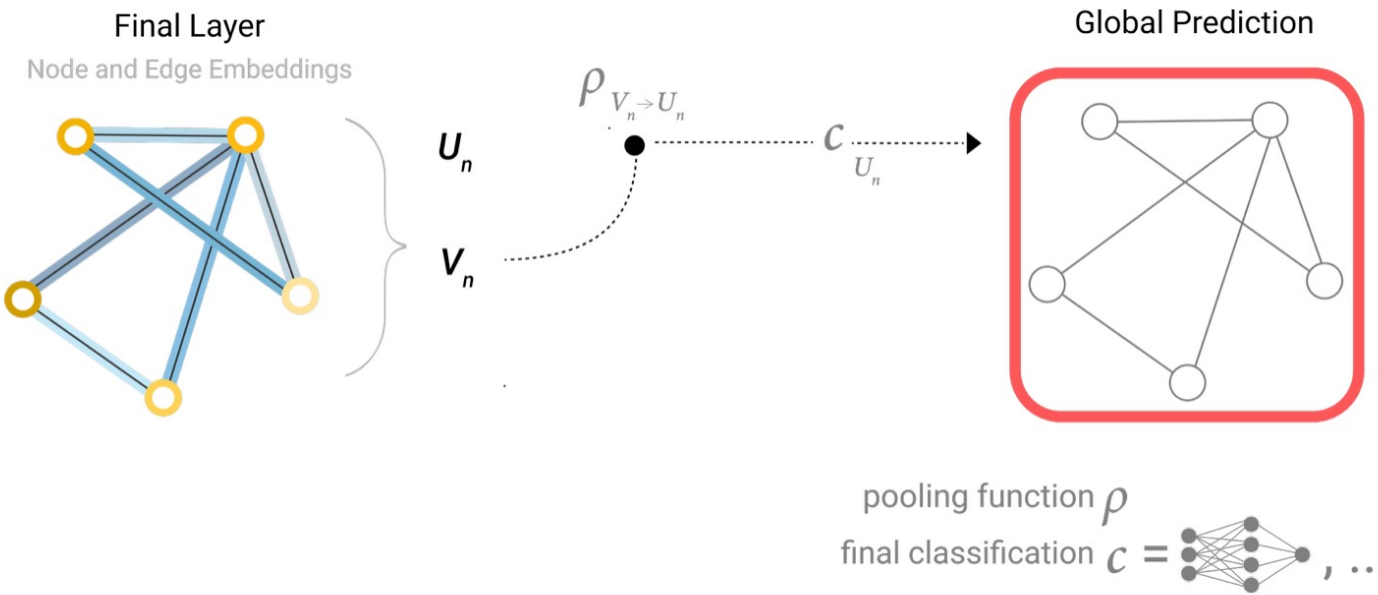
If you only have edge embeddings and want to make node-level predictions...

How to make predictions?



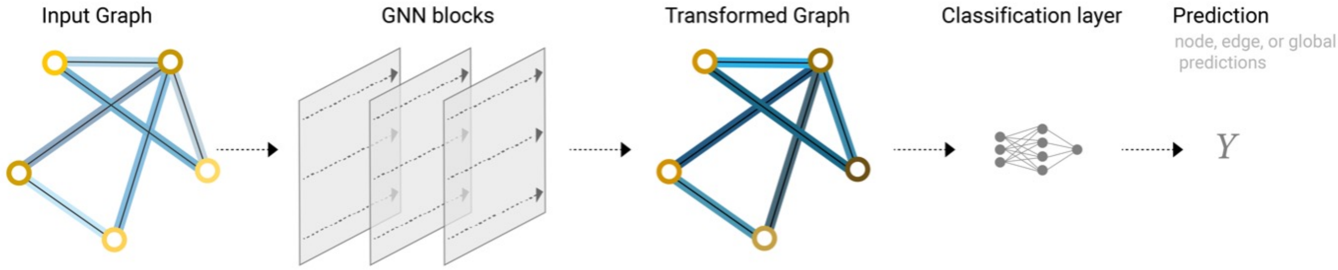
If you only have node embeddings and want to make edge-level predictions...

How to make predictions?



If you only have node embeddings and want to make graph-level predictions...

More in general



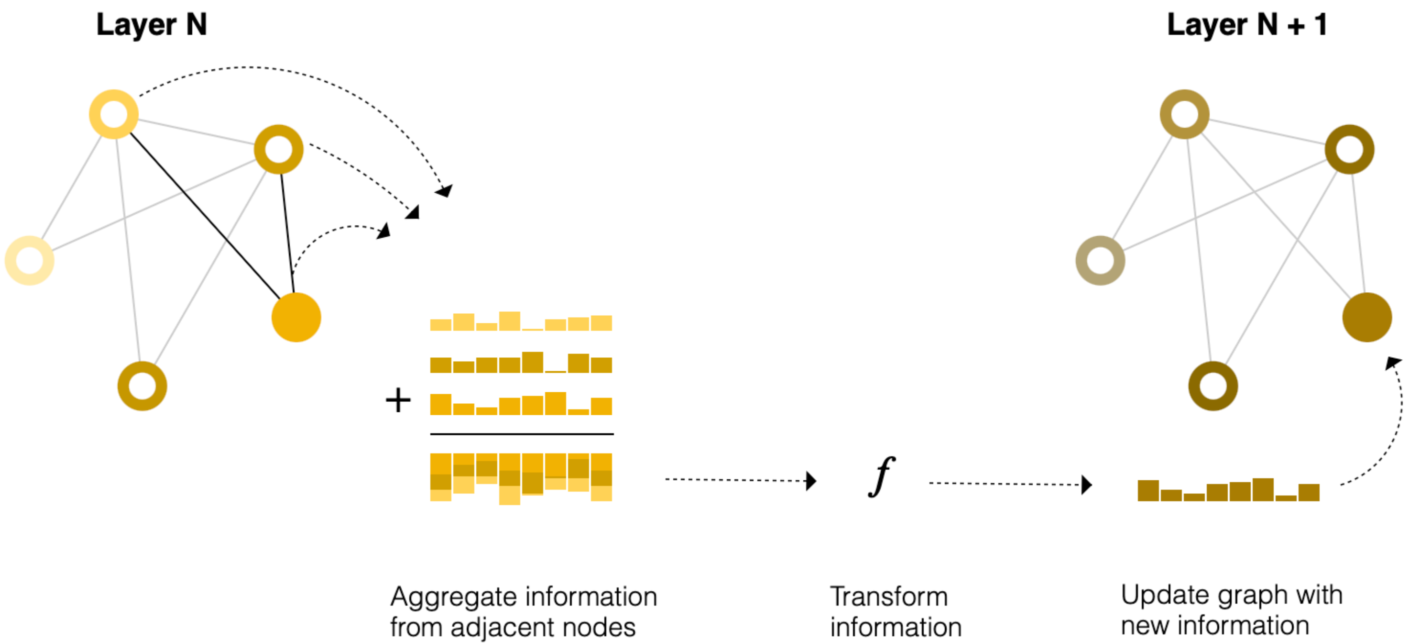
- This pooling serves as a building block for more advanced GNNs
- Connectivity is not used (only in the pooling)

Message passing

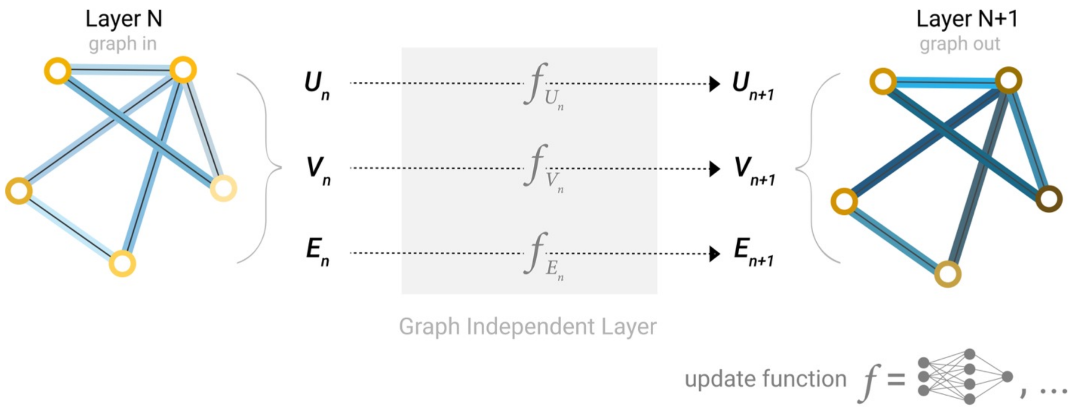
- **It allows to exploit graph connectivity when learning the embeddings, making them aware of the connectivity itself**

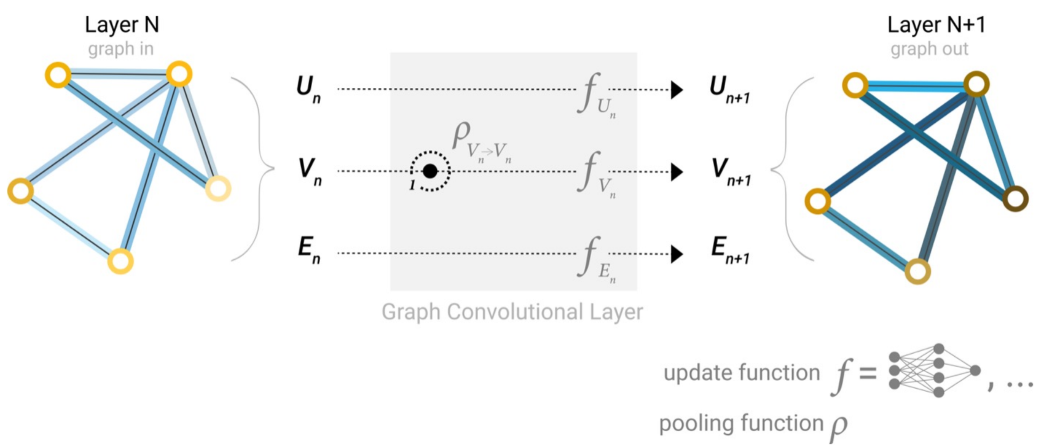
- **It includes different steps:**
  - For each node/edge aggregate the neighbouring [node/edge] embeddings
  - Pooling the embeddings
  - Provide the pooling result to an update function

Message Passing



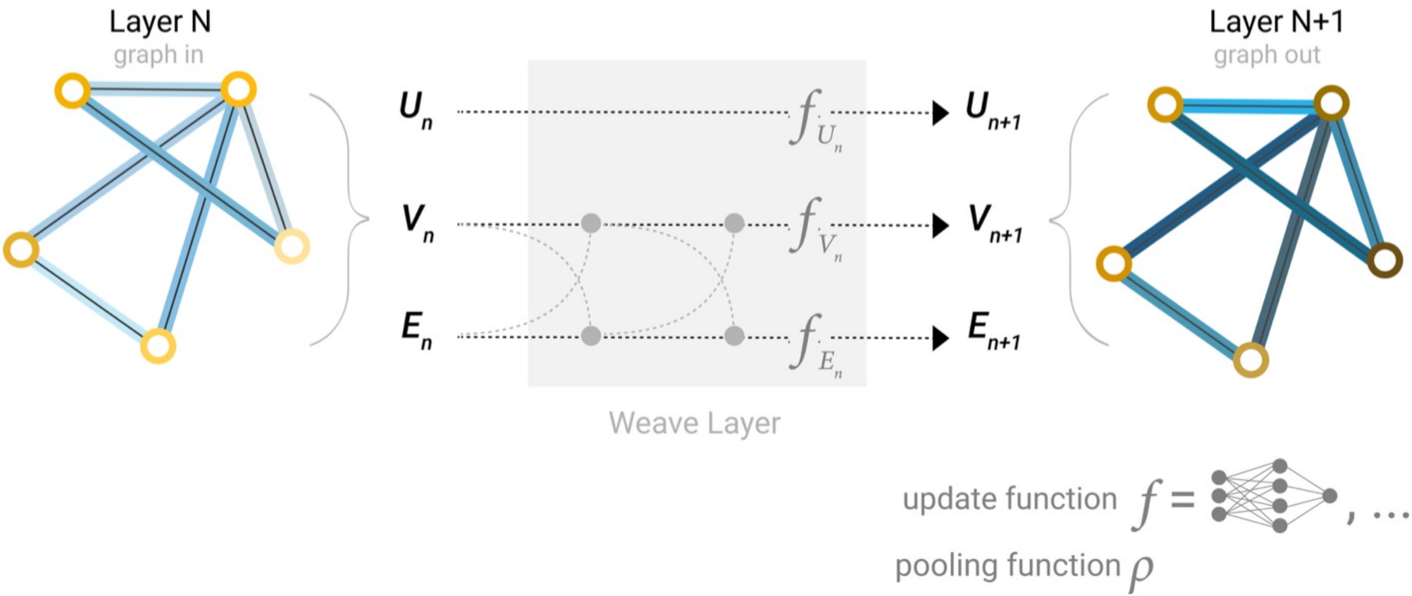
How to encode graphs now?





How to encode graphs now?

- Nodes and edges might have embeddings of different lengths…



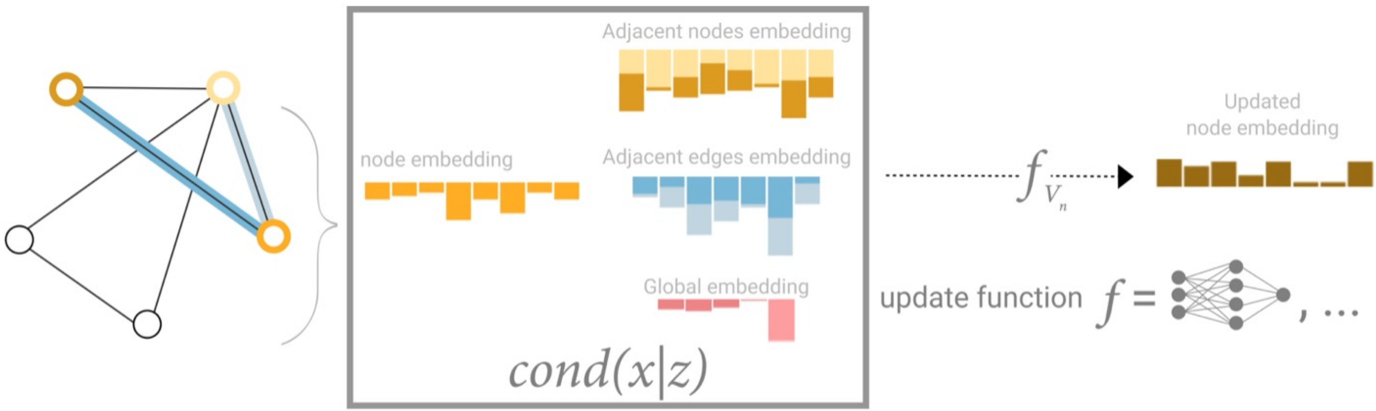
How to encode graphs?

A possible issue

- Nodes that are very far away from each other in the graph might be able to transfer information to one another, even in the presence of multiple layers
- One could allow the nodes to transfer information to all other nodes (regardless the graph connectivity) but this is very expensive for large graphs
- More efficient solution: using a master node

Using a master node

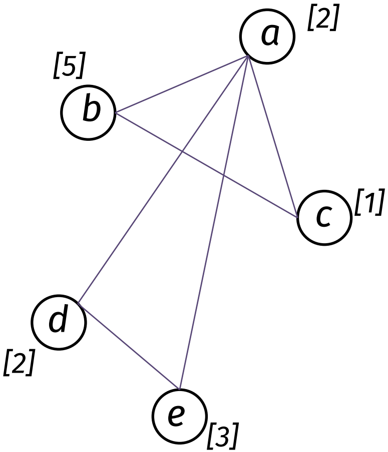
A master node (or context vector) U is a global vector connected to all other nodes and edges in the network → It acts like a bridge between all the graph elements



How to make graph-level predictions

- Using the master node
- Aggregating all the nodes/edges
- In both cases, a final block in the architecture responsible for the classification/regression is needed

Graph convolutions as matrix multiplication

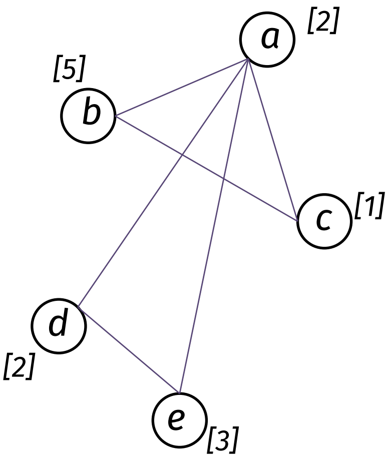


Aggregations based on summation can be obtained by multiplying the adjacency matrix with the node features

	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	1
e	1	0	0	1	0

	x
a	2
b	5
c	1
d	2
e	3

Graph convolutions as matrix multiplication



A multiplication corresponds to a simple sum aggregation

With multiple multiplications we may propagate the information at a greater distance → This is a form of traversing over the graph

Using attention mechanisms

- Not all neighbours have relevant information for a certain node

- The attention mechanism allows to adaptively weight the contribution of each neighbour when updating a node

Using attention mechanisms

A popular way to compute the weights

$$\alpha_{ij} = \frac{f(Wh_i, Wh_j)}{\sum_{j' \in \mathcal{N}(i)} f(Wh_i, Wh_j)}$$

where each weight models the importance of node j for node i as a function of their representations

f is called the attention function (e.g. inner product)

UniGe

MaLGa