What is a graph ?

Node and edges to represent entities

Main concept : Permutation invariant?

| GNN | CNN | Transformer |
| --- | --- | --- |
| A GNN is made of successive layers. A GNN layer represents a node as the combination (aggregation) of the representations of its neighbours and itself from the previous layer (message passing), plus usually an activation to add some nonlinearity. | CNN can be seen as a GNN with fixed neighbour sizes (through the sliding window) and ordering (it is not permutation equivariant). | Comparison to other models: AA [Transformer](https://arxiv.org/abs/1706.03762v3) without positional embeddings can be seen as a GNN on a fully-connected input graph. |

Different type of graphs

| Method | Aggregation andMessage passing | | When to use |
| --- | --- | --- | --- |
| [Graph Convolutional Networks](https://tkipf.github.io/graph-convolutional-networks/) | averages the normalised representation of the neighbours for a node | |  |
| [Graph Attention Networks](https://petar-v.com/GAT/) | learn to weigh the different neighbours based on their importance (like transformers); | |  |
| [GraphSAGE](https://snap.stanford.edu/graphsage/) | samples neighbours at different hops before aggregating their information in several steps with max pooling. | |  |
| [Graph Isomorphism Networks](https://arxiv.org/pdf/1810.00826v3.pdf) | aggregates representation by applying an MLP to the sum of the neighbours' node representations. | |  |

### **GNN shape and the over-smoothing problem**

At each new layer, the node representation includes more and more nodes.

A node, through the first layer, is the aggregation of its direct neighbours. Through the second layer, it is still the aggregation of its direct neighbours, but this time, their representations include their own neighbours (from the first layer). After n layers, the representation of all nodes becomes an aggregation of all their neighbours at distance n, therefore, of the full graph if its diameter is smaller than n!

If your network has too many layers, there is a risk that each node becomes an aggregation of the full graph (and that node representations converge to the same one for all nodes). This is called the oversmoothing problem

This can be solved by :

* scaling the GNN to have a layer number small enough to not approximate each node as the whole network (by first analysing the graph diameter and shape)
* increasing the complexity of the layers
* adding non message passing layers to process the messages (such as simple MLPs)
* adding skip-connections.

Graph Transformer

The most recent approach is [*Pure Transformers are Powerful Graph Learners*](https://arxiv.org/abs/2207.02505) (Kim et al, 2022), which introduced TokenGT. This method represents input graphs as a sequence of node and edge embeddings (augmented with orthonormal node identifiers and trainable type identifiers), with no positional embedding, and provides this sequence to Transformers as input. It is extremely simple, yet smart!

<https://huggingface.co/blog/intro-graphml>

In Python, we can easily build a GCN using PyTorch:

**import** torch

**from** torch **import** nn

**class** **GCN**(nn.Module):

**def** **\_\_init\_\_**(self, \*sizes):

super().\_\_init\_\_()

self.layers = nn.ModuleList([

nn.Linear(x, y) **for** x, y **in** zip(sizes[:-1], sizes[1:])

])

**def** **forward**(self, vertices, edges):

*# ----- Build the adjacency matrix -----*

*# Start with self-connections*

adj = torch.eye(len(vertices))

*# edges contain connected vertices: [vertex\_0, vertex\_1]*

adj[edges[:, 0], edges[:, 1]] = 1

adj[edges[:, 1], edges[:, 0]] = 1

*# ----- Forward data pass -----*

**for** layer **in** self.layers:

vertices = torch.sigmoid(layer(adj @ vertices))

**return** vertices

‘https://distill.pub/2021/gnn-intro/