MARENGO Matteo
matteo.marengo@ens-paris-saclay.fr

Lab session # 6
ALTEGRAD 2023

11/28/23

1 Question 1

The Erdős–Rényi model is a model that generates random graphs by connecting nodes randomly. Indeed, each edge is included in the graph with a probability p, independent of every other edge.

Therefore, in an Erdős–Rényi graph $G(n, p)$, where n is the number of nodes and p is the probability of an edge being present between any two nodes, the expected degree of a node is $\mathbb{E}[D] = p \times (n - 1)$.

Indeed, it is the probability p of an edge being present between any two nodes multiplied by the number of possible connections a node can have, which is $n - 1$ (since a node cannot have an edge to itself).

Furthermore as it in indicated in [3], $P(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$. It follows a binomial distribution. Then, when we look at the expectation of the binomial distribution, we obtain $\mathbb{E}(D) = p \times (n - 1)$

Finally, for $n = 25$ nodes:

- With edge probability $p = 0.2$:
$$E(D) = 0.2 \times (25 - 1) = 4.8$$
- With edge probability $p = 0.4$:
$$E(D) = 0.4 \times (25 - 1) = 9.6$$

2 Question 2

In graph neural networks, the readout function is used to aggregate information from all nodes in the graph to form a single representation. It is sometimes called a global pooling layer that provides fixed-size representation of the whole graph [4]. As it is shown in Fig 1 of the handbook or in Fig 1 the readout allows to obtain the sum of the node representations. In the Fig 1 of the handbook we can compute the dimension of each layer; Z^0 is $(n, h1)$, Z^1 is $(n, h2)$, z_G is $(3, h2)$, first fc layer is $(3, h3)$ and the output is $(3, n_{class})$.

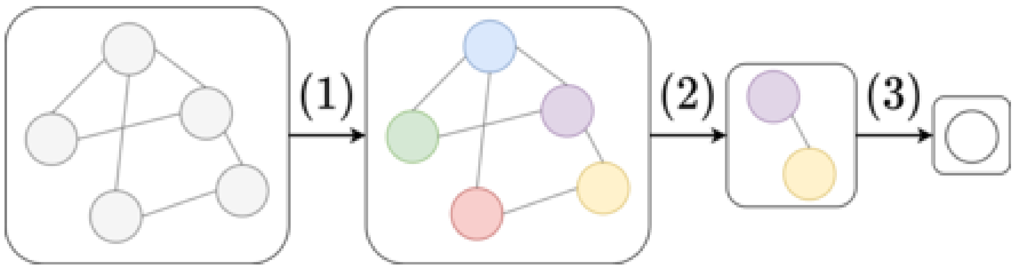


Figure 1: Basic building blocks of a graph neural network. 1) Message Passing Layer 2) Local Pooling Layer 3) Readout layer. Adapted from [4].

What is so convenient with sum or mean is their ability to handle variable-sized input graphs. The final representation of the graph will not depend on the number of nodes.

Fully-connected layers have a fixed size, it means that that they are not suited for graphs of varying sizes. Furthemore, they might not capture the hierarchical and relational structure of graphs.

The sum and mean can also handle the **permutation invariance** property of graphs. Indeed, the graphs can be organised differently than in Figure 1, the index vectors can be different [4]. The ordering of nodes and edges should not impact the final output. Finally, it is also more efficient than fully connected layers. To echo to Fig 1, the Fig 2 also shows how does this readout function works. In [1], they express their thoughts concerning new types of pooling that could be used to preserve intrinsic structures of graphs.

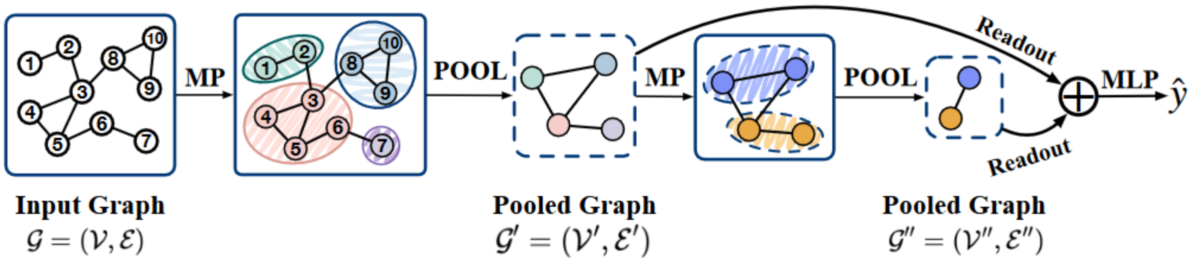

MP
POOL
MP
POOL
Readout
Readout
MLP
$\hat{y}$
Input Graph
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Pooled Graph
$\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$
Pooled Graph
$\mathcal{G}'' = (\mathcal{V}'', \mathcal{E}'')$

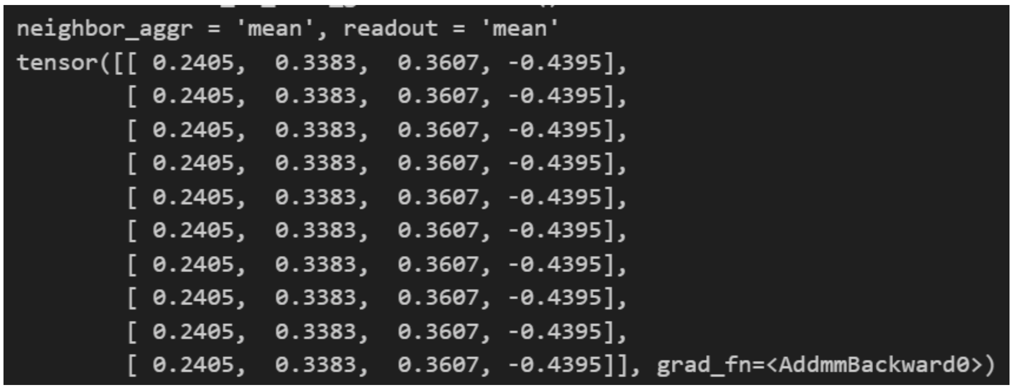
Figure 2: Illustration of graph pooling. Adapted from [1].

If we take the examples of G1, G2, G3, we would need a fully connected layer of input size of three as there are three nodes. It won't be good for G2 as there four nodes and in addition, it is overly complicated as G2 is only a regular graph. It won't be good either for G3 as there are two nodes. In conclusion, the model would need to have multiple different fully connected layers to handle different graph sizes. Therefore sum and mean operations provide a simple and consistent way to aggregate node features into a graph-level representation regardless of the number of nodes or the structure.

3 Question 3

Here are the results for Task 8, it is the vector representation of the 10 cyclic graphs that have been generated after the feedforward pass. There are for each graph a vector of size 4 as the output dimension is 4.

- Fig 3 Neighborhood Aggregation: **Mean** — Readout Function: **Mean**
- Fig 4 Neighborhood Aggregation: **Sum** — Readout Function: **Sum**
- Fig 5 Neighborhood Aggregation: **Sum** — Readout Function: **Mean**
- Fig 6 Neighborhood Aggregation: **Mean** — Readout Function: **Sum**

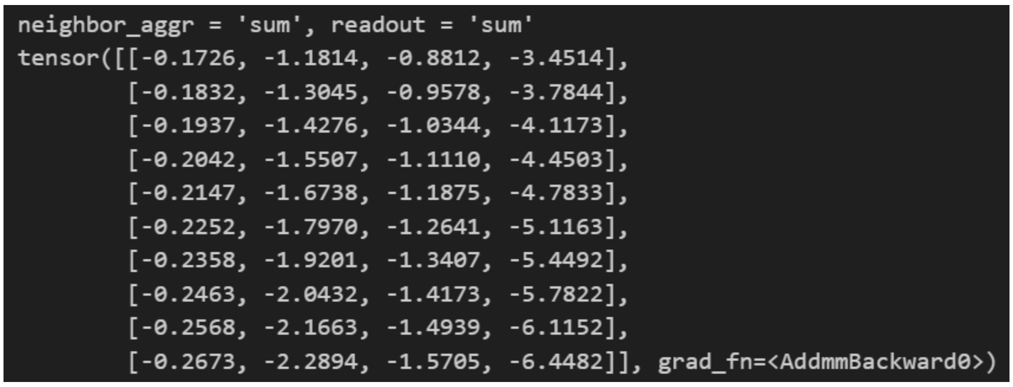


```
neighbor_aggr = 'mean', readout = 'mean'
tensor([[ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395],
        [ 0.2405,  0.3383,  0.3607, -0.4395]], grad_fn=<AddmmBackward0>)
```

Figure 3: Results with mean for neighborhood aggregation and mean for the readout function

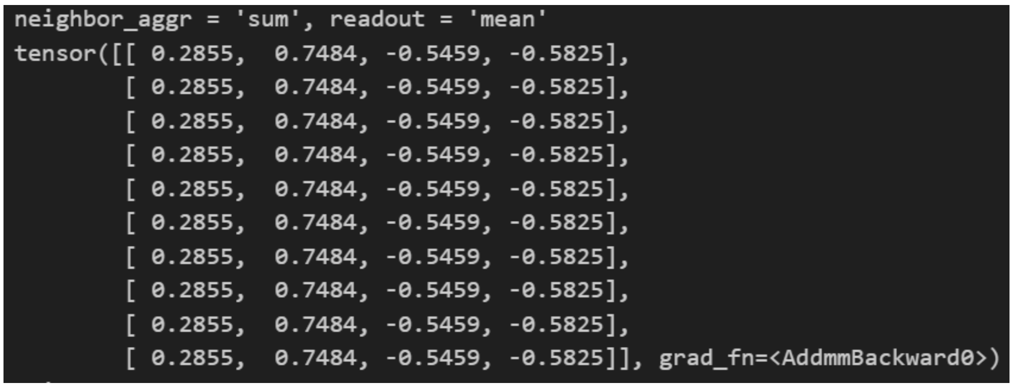
First, we can notice that when the neighborhood aggregation is mean the 10 cyclic graphs cannot be distinguished and that the emerging vectors are the same. In the contrary, when the neighborhood aggregation is sum the 10 cyclic graphs have different emerging vectors, they can be distinguished. It can be explained step by step.

First, all nodes in the cycle graphs are initialized with the **same feature vector** as asked in the task in the lab. Then as the cycle graph is a regular graph, each node receives the same information from its neighborhood. When the mean aggregation method is used, the features from a node's neighbors are summed and then divided by the degree of the node. In the case of cycle graphs, since each node has two neighbors and each neighbors has the same feature, the aggregated feature will be the same for each node after each round of message passing. As the readout phase is also mean it will produce the same final graph representation for each cycle graph.



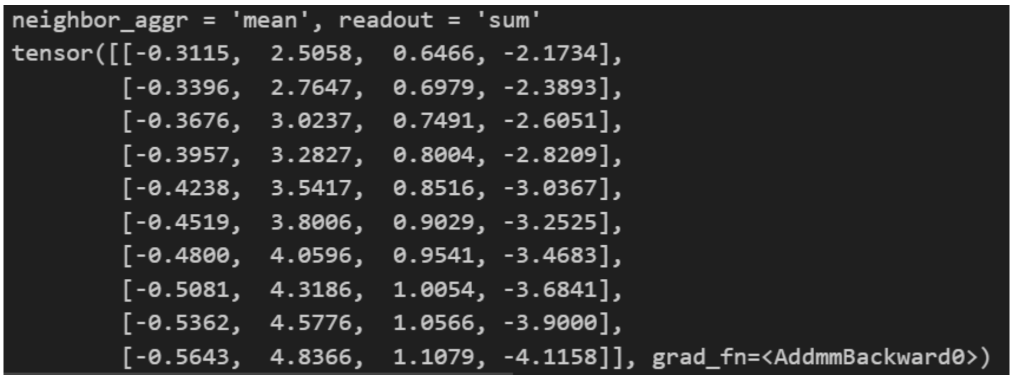
```
neighbor_aggr = 'sum', readout = 'sum'
tensor([[-0.1726, -1.1814, -0.8812, -3.4514],
        [-0.1832, -1.3045, -0.9578, -3.7844],
        [-0.1937, -1.4276, -1.0344, -4.1173],
        [-0.2042, -1.5507, -1.1110, -4.4503],
        [-0.2147, -1.6738, -1.1875, -4.7833],
        [-0.2252, -1.7970, -1.2641, -5.1163],
        [-0.2358, -1.9201, -1.3407, -5.4492],
        [-0.2463, -2.0432, -1.4173, -5.7822],
        [-0.2568, -2.1663, -1.4939, -6.1152],
        [-0.2673, -2.2894, -1.5705, -6.4482]], grad_fn=<AddmmBackward0>)
```

Figure 4: Results with sum for neighborhood aggregation and sum for the readout function



```
neighbor_aggr = 'sum', readout = 'mean'
tensor([[ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825],
        [ 0.2855,  0.7484, -0.5459, -0.5825]], grad_fn=<AddmmBackward0>)
```

Figure 5: Results with sum for neighborhood aggregation and mean for the readout function



```
neighbor_aggr = 'mean', readout = 'sum'
tensor([[-0.3115,  2.5058,  0.6466, -2.1734],
        [-0.3396,  2.7647,  0.6979, -2.3893],
        [-0.3676,  3.0237,  0.7491, -2.6051],
        [-0.3957,  3.2827,  0.8004, -2.8209],
        [-0.4238,  3.5417,  0.8516, -3.0367],
        [-0.4519,  3.8006,  0.9029, -3.2525],
        [-0.4800,  4.0596,  0.9541, -3.4683],
        [-0.5081,  4.3186,  1.0054, -3.6841],
        [-0.5362,  4.5776,  1.0566, -3.9000],
        [-0.5643,  4.8366,  1.1079, -4.1158]], grad_fn=<AddmmBackward0>)
```

Figure 6: Results with mean for neighborhood aggregation and sum for the readout function

When both aggregation function and readout function is sum, the features from a node's neighbors are simply **added together** without any normalization by the degree of the node. This is different from the 'mean' aggregation, where the sum is divided by the number of neighbors. In the readout phase, when the sum function is used it aggregates the features of all nodes in a graph by summing them. Since the nodes in larger graphs would have accumulated more feature information the final sum for each graph will be different, reflecting the size of the graph. Larger cycle graphs will have larger aggregates feature values compared to smaller ones.

To conclude, when the readout function is mean the graph representation will be the same for the 10 graphs, the sum readout have to be used. Furthermore, with the sum readout function, the vector representation will increase as the graph size increases, this trend is well observed in Fig 6 and Fig 4.

4 Question 4

We are looking for two non-isomorphic graphs G_1 and G_2 that will never be distinguished by the GNN model which uses the sum operator both for neighborhood aggregation and as the model's readout function.

An example of these graphs is depicted in Fig 7. Indeed, these two graphs are non-isomorphics. It is proven computationally (with the is.isomorphic function from networkX library).

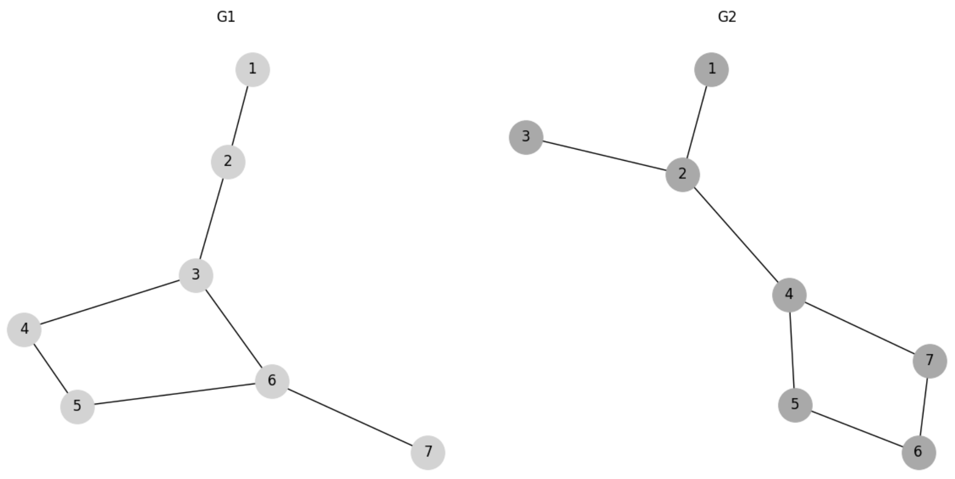
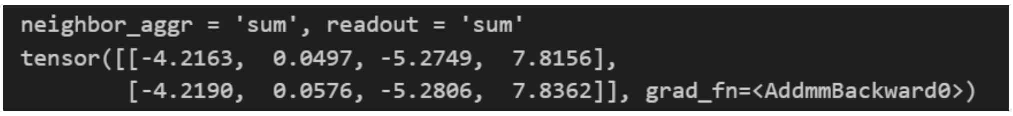


Figure 7: Two non-isomorphic graphs G1 and G2 which can never be distinguished by the GNN model which uses the sum operator both for neighborhood and aggregation and as the model's readout function.

Now to proove that there won't be distinguised by GNN using sum operator, we can compute it with the same method as for Task 11 and visualize that the output, the representations of the two graphs are the same, there cannot be distinguisable (as it is shown in Fig 8). Furthermore, it was the expected result as far as the two graphs have the same degree sequence (1,2,3,2,2,3,1) for G_1 and (1,3,1,3,2,2,2) for G_2.



```
neighbor_aggr = 'sum', readout = 'sum'
tensor([[-4.2163,  0.0497, -5.2749,  7.8156],
        [-4.2190,  0.0576, -5.2806,  7.8362]], grad_fn=<AddmmBackward0>)
```

Figure 8: Output of the GNN with G1 and G2, the representing vectors are the same.

In conclusion, as stated in [4] and visualized by our results, GNNs are not more expressive than the Weisfeiler-Leman Graph Isomorphism Test (heuristic test for existence ofan isomorphism between two graphs [2]). As in our example, different graph structures cannot be distinguised by GNNs. This is not powerful enough. New architectures are looked nowadays to tackle this issue.

References

[1] Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities, 2023.

[2] A. A. Weisfeiler, B. Yu.; Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. In *Nauchno-Technicheskaya Informatsia*, 1968.

[3] Wikipedia. Erdos-rényi model. In *https://en.wikipedia.org/wiki/Erdos-Rényi-model*, 2023.

[4] Wikipedia. Graph neural network. In *https://en.wikipedia.org/wiki/Graph*$_{n}eural_{n}etwork$, 2023.