Molecular Property Prediction on HIV replication inhibition

A Preprint

Thibault Monsel
CentraleSupélec
thibault.monsel@supelec.fr

Antoine Paris
CentraleSupélec
antoine.paris@student.ecp.fr

March 15, 2021

Abstract

Graph-structured data appears frequently in domains like chemistry, natural language semantics, socials networks, knowledge bases and biology. In the recent decade, we have seen a surge in research on graph representation learning. These advances have led to a new state-of-the-art results in many domains. Graph Neural Network (GNNs) have shown to be effective models for predictive tasks on graph-structured data compared to other types of neural networks.

1 Introduction

The human immunodeficiency virus (HIV) are two species of lentivirus that infect humans. Over time, they cause acquired immunodeficiency syndrome (AIDS) a condition in which progressive failure of the immune system allows life-threatening opportunistic infections and cancers to thrive [22]. One antiviral strategy is to block as much as possible HIV replication. In this study, we focus on the classification of molecules that can inhibit HIV replication. Hence our classification task is binary, either molecules curb HIV proliferation or don't.

2 Dataset

We are using the *ogbg-molhiv* dataset that comes from Standford's Open Graph Benchmark (OGB) [12] website for graph machine learning. All the molecules are pre-processed using RDKit [6]. Each graph represents a molecule, where nodes are atoms, and edges are chemical bonds. Input node features are 9-dimensional, containing atomic number and chirality, as well as other additional atom features such as formal charge and whether the atom is in the ring or not. Input edge features are 3-dimensional, containing bond type, stereoisomerism and its conjugation. More information is available on OGB.

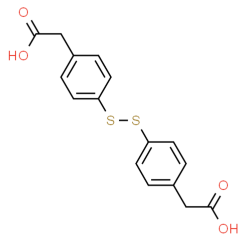


Figure 1: HIV molecule inhibitor

Table 1: *ogbg-molhiv* metadata

#Graphs	#Nodes per graph	#Node features.	#Edges per graph	#Edge Features	%Positive labels
41,127	25.5	9	27.5	3	3.5

3 Theoretical groundwork of GNNs

3.1 Graph and GNN formalism

Formally, a graph $\mathscr{G} = (\mathcal{V},\mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between these nodes. We denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{E}$. One way to represent a graph is through an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. One can extend the edge notation to include an edge or relation type τ, e.g $(u, \tau, v) \in \mathcal{E}$ and we can define one adjacency matrix $\mathbf{A}_\tau$ per edge type. We call such graph *multi-relational* and the entire graph can be summarized by an adjacency tensor $\mathcal{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{R}|\times|\mathcal{V}|}$ where $\mathcal{R}$ is the set of relations [7].

The *graph neural network* (GNN) formalism is a general framework for defining deep neural networks on graph data. The key idea is that we want to generate a representations of nodes that actually depend on the structure of the graph, as well as any feature information we might have.

3.2 Neural Message Passing

The basic graph neural network (GNN) can be motivated in a variety of ways. The same fundamental GNN model has been derived as a generalization of convolution to non-Euclidian data [1], as well as by the analogy to class graph isomorphism tests [11]. Regarless of the motivation, the defining feature of a GNN is that it uses a form of *neural message passing* in which vector messages are exchanged between nodes and updates using neural networks [5].

Below we will describe how we can take an input graph $\mathscr{G} = (\mathcal{V},\mathcal{E})$, along with a set of node features $\mathbf{X} \in \mathbb{R}^{d\times|\mathcal{V}|}$ and use this information to generate node embeddings $\mathbf{z_u}, \forall u \in \mathcal{V}$.

During each message-passing iteration in a GNN, a *hidden embedding* $\mathbf{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated according to information aggregated from u's graph neighborhood $\mathcal{N}(u)$. The message-passing update can be expressed as follows :

$$\begin{aligned}\mathbf{h}_u^{(k+1)} &= UPDATE^{(k)}(\mathbf{h}_u^{(k)}, AGGREGATE^{k}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \\ &= UPDATE^{(k)}(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)})\end{aligned}$$

where $UPDATE$ and $AGGREGATE$ are arbitrary differentiable function (i.e, neural networks) and $\mathbf{m}_{\mathcal{N}(u)}$ is the "message" that is aggregated from u's graph neighborhood $\mathcal{N}(u)$.

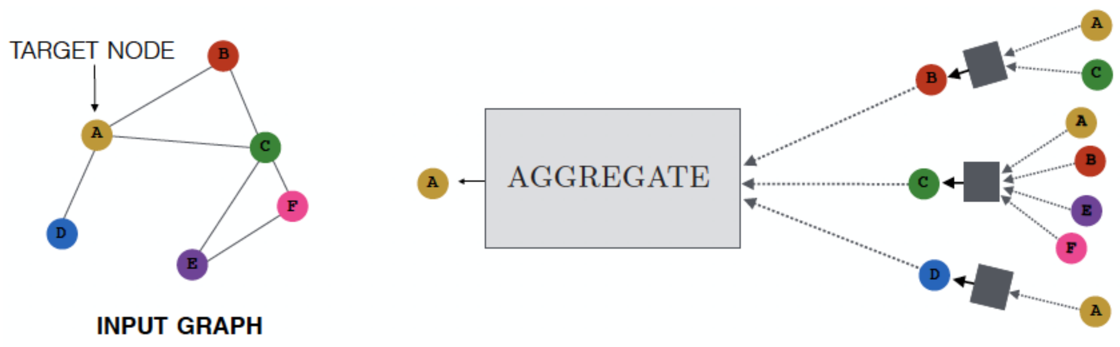


Figure 2: Overview of how a single node aggregates messages from its local neighborhood. The model aggregates messages from A's local graph neighbors (i.e., B, C, and D), and in turn, the message coming from these neighbors are based on information aggregated from their respective neighborhoods, and so on.

At each iteration k of the GNN, the $AGGREGATE$ function takes as input the set of embedding of the nodes in u's graph neighborhood $\mathcal{N}(u)$ and generate a message $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ based on this aggregated neighborhood information. The update function $UPDATE$ then combines the message $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ with the previous embedding $\mathbf{h}_u^{(k-1)}$ of node u to generate the update embedding $\mathbf{h}_u^{(k)}$. The initial embeddings at $k = 0$ are set to the input feature of all nodes, i.e, $\mathbf{h}_u^{(0)} = \mathbf{x_u}, \forall u \in \mathcal{V}$. After running K iterations of the GNN message passing, we can use the output of the final layer to define the embedding for each node, ie. [8],

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}$$

3.3 Applying the theory

In practice, to translate the *neural message passing* paradigm, one must define the $UPDATE$ and $AGGREGATE$ functions. For example, the most basic GNN framework is a simplification of the model proposed by Merkwirth and Lengauer [17].

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v$$
$$UPDATE(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \sigma(\mathbf{W}_{self}\mathbf{h_u} + \mathbf{W}_{weight}\mathbf{m}_{\mathcal{N}(u)})$$

where $\mathbf{W}_{weight}$, $\mathbf{W}_{self}$ are trainable parameters and $\sigma(.)$ is a non-linearity.

From this elementary GNN type of neural network layer, many other types were developped such as Graph convolutional networks (GCNs) [13], the Graph Attention Network (GAT) [21], etc...

4 Experiments

4.1 Prior work on *ogbg-molhiv*

Many new state-of-the-art novel architectures and new graph-related techniques have been introduced recently. For instance, FLAG (Free Large-scale Adversarial Augmentation on Graphs) iteratively augments node features with gradient-based adversarial perturbations during training, and boosts performance at test time [15]. Trang Pham et al. introduced a method to augment an attributed graph with a virtual node that is bidirectionally connected to all existing nodes [19]. Finally, Soheil Kolouri et al. presented a new type of graph embedding with their Wasserstein Embedding for Graph Learning (WEGL) model [14].

4.2 Our experimental approach

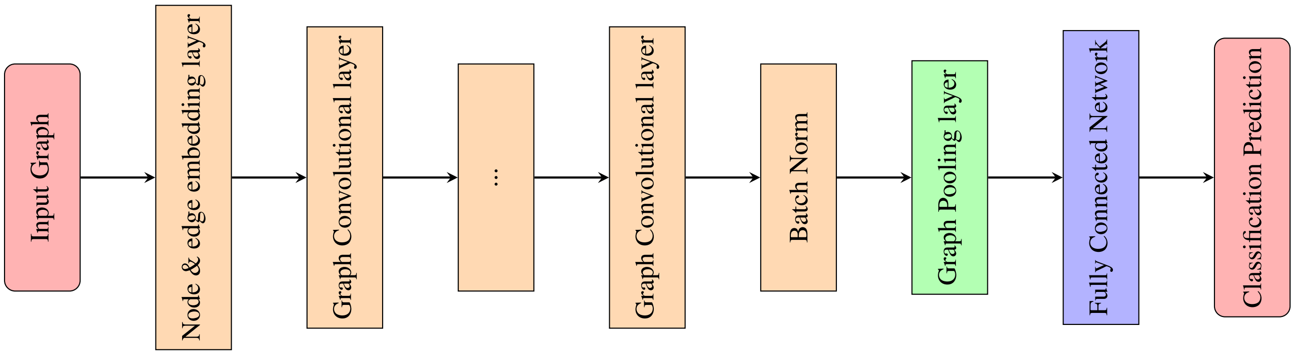
In this project, our goal is to study and discover different types of GNNs architectures. We will use the *ogbg-molhiv* dataset in two ways with and without its edge features in order to compare and discuss results. The code is open-sourced and written with the Pytorch Geometric package [3], a framework built on top of Pytorch.

Our neural network will be composed of 2 blocks. The first block will learn the graph embedding thanks to specific types of layer e.g. graph convolutional layers. The second is a fully connected network for the classification. The project focuses on the analysis of the performance of the first block and its hyperparameters. Purposely, we won't scrutinize the node & edge embeddings provided by the OGB python package.

All models were trained with ADAM optimizer using a learning rate $l_r = 0.05$, 25 epochs and a binary cross entropy loss while hyperparameters were fine-tuned (#graph convolution aggregator, #global pooling layer and #network depth). Models were trained with the training set, tuned with the validation and eventually evaluated with the test. Ten runs were done to obtain our test results.

We propose to investigate the following type of network where we will try to understand the model's influence on the hyperparameters mentionned above.

Figure 3: Neural network description


Input Graph
Node & edge embedding layer
Graph Convolutional layer
...
Graph Convolutional layer
Batch Norm
Graph Pooling layer
Fully Connected Network
Classification Prediction

4.2.1 GNNs without edge features

We decided to use Christopher Morris and al [18] findings in GNNs as our elementary graph convolutional layer called $GraphConv$ in the Pytorch Geometric library. Hyperparameters mentionned below are attributes that define our model in our code.

Set Aggregator : In order to view the influence of set aggregators ($AGGREGATE$ function in 3.2) in our graph convolutional layer we used the fixed hyperparamters $in\_channel = 100$, $number\_hidden\_layer = 2$, $hidden\_out\_channel = 64$, $out\_channel = 32$, $pool\_layer = "sort"$ and $k = 15$.

Each set aggregator is defined mathematically as follows with u's graph neighborhood $\mathcal{N}(u)$ and updated embedding $\mathbf{h}_u$.

$$\textit{sum} : \mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v \qquad \textit{mean} : \mathbf{m}_{\mathcal{N}(u)} = \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v \qquad \textit{max} : \mathbf{m}_{\mathcal{N}(u)} = \max_{v \in \mathcal{N}(u)} \mathbf{h}_v$$

Table 2: Set Aggregator test results

Set aggregator	ROC-AUC
add	0.5847 ± 0.0496
mean	0.5910 ± 0.0376
max	$\mathbf{0.6320 \pm 0.0305}$

The *max* set aggregator with our fixed hyperparameter yields the best performance. This may be explained by the fact that our graph data are molecules. Indeed, the atom with the *max* aggregator only takes the most "important" message from his neighborhood. Layer after layer, each atom updates their own representation with only one nearby atom that influences it the most.

Global Pooling : Then, we tried out different global pooling layers in our network. Hyperparameters were fixed to $in\_channel = 100$, $number\_hidden\_layer = 2$, $aggregator = "max"$, $hidden\_out\_channel = 64$ and $out\_channel = 32$.

Each global pooling layer is defined mathematically as follows with u's embedding $\mathbf{h}_u$ and $\mathbf{z}$ the graph's representation defined by $\mathscr{G} = (\mathcal{V},\mathcal{E})$. The global sort pooling layer is defined in [23].

$$\textit{add} : \mathbf{z} = \sum_{u \in \mathcal{V}} \mathbf{h}_u \qquad \textit{max} : \mathbf{z} = \max_{u \in \mathcal{V}} \mathbf{h}_u \qquad \textit{mean} : \mathbf{z} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{h}_u$$

Table 3: Graph Pooling test results

Graph pooling	ROC-AUC
sort (k=15)	0.6320 ± 0.0305
add	0.7007 ± 0.0485
max	0.7198 ± 0.0291
mean	$\mathbf{0.7238 \pm 0.0186}$

Since we have four $GraphConv$ layers each node has information from their 4-hop neighborhood. Respectively *add*, *mean*, *max* and *sort* pooling layer would pool the whole information of the graph, the mean information of the graph, the most important node of molecule and an combination of k most important nodes. The *mean* pooling summarizes the best the underlining chemical property of our compound.

Network Depth :Afterwards, the network's depth graph convolution layer impact was checked out with the following hyperparameters $in\_channel = 100$, $aggregator = "max"$, $hidden\_out\_channel = 64$, $out\_channel = 32$ and $pool\_layer = "mean"$.

Table 4: Network depth test results

# Layers	ROC-AUC
2	$\mathbf{0.7679 \pm 0.0107}$
3	0.7518 ± 0.0251
4	0.7357 ± 0.0401
5	0.7079 ± 0.0392
6	0.6880 ± 0.0557

Depending on the number of graph convolutional layer, nodes aggregate information from their surrounding. For instance, if we have k layers then each node will have their representation updated with information that comes from their k-hop neighborhood. An illustration of the process is found on Figure 2 in 3.2. Our experimental values point to one specific phenonemon found in GNNs called *over-smoothing*. The essential idea is that after several iterations of GNN message passing, the representation for all nodes in the graph can be very similar to one another. This is problematic because it make its hard to build deeper GNN model [9].

4.2.2 GNNs with edge features

We also tried a different approach by training a model using Relational Graph Convolutional layers [20] which takes the edge features into account. However the RGCN model is much slower to train than the standard GNN model, even with a GPU and we probably did not have the computation power to train the model sufficiently to get better results, as the model kept improving when we stopped him after several hours of training. This can be explained by the message passing expression :

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{\tau \in \mathcal{R}} \sum_{v \in \mathcal{N}_\tau(u)} \frac{\mathbf{W}_\tau \mathbf{h}_v}{\mathbf{f_n}(\mathcal{N}(u), \mathcal{N}(v))}$$

where $\mathbf{f_n}$ is a normalization function, u's neighborhood $\mathcal{N}(u)$, $\mathbf{W}_\tau$ a trainable parameter, $\mathbf{h}_v$ v's embedding and τ a relation type in $\mathcal{R}$. The RGCN approach drastically increase the number of parameter hence the training time [10]. The results we obtained for this model are also less precise because we could not afford to train the model on the same number of run we did with the GraphConv Networks and evaluated each model over 3 runs .

We first compared the influence of the number of layers, using a max aggregator and a $mean$ pooling layer and the same hyperparameters seem above.

Table 5: RGCN depth test results

# Layers	ROC-AUC
2	0.6312 ± 0.0601
3	0.5256 ± 0.0545
4	$\mathbf{0.6786 \pm 0.0523}$
5	0.6510 ± 0.0703

As the model took a long time to train and did not produce the same kind of results that the $GraphConv$ did, we dediced to not evaluate the impact of the type of aggregator or pooling layer we used but to try a model with combinations of $GraphConv$ layers and RGCN layers. We tried 2 types of models :

The first one we added a $GraphConv$ between the RGCN layers and the pooling layer.

The second one we inspired us of the architecture of the Inception model used in image recognition, that chooses directly which type of convolutionnal layer it uses. So for each RGCN layer, we added a parallel $GraphConv$ layer and we concatenated the outputs of the 2 model before passing it to the next layer.

Table 6: RGCN depth test results

# Layers	ROC-AUC
Adding a GraphConv layer	0.7074 ± 0.0422
Inception inspired	0.6887 ± 0.0475

The results are then slightly better than previous ones, but it not really significant due to the poor precision of our metrics and the improvement is probably caused by the better performance of the GraphConv layers on the problem.

4.3 Literrature comparison

In this section, we compare our best model with and without edge attributes with other published models found in the litterature taken from [4] and [2].

Table 7: Test results with no edge attributes

# Models	ROC-AUC
GCN-E	0.7607 ± 0.0097
GATEDGCN-E	0.7765 ± 0.0050
GIN-E	0.7558 ± 0.0140
HIMP-GNN	$\mathbf{0.7880 \pm 0.0082}$
Ours	0.7679 ± 0.0107

Table 8: Test results with edge attributes

# Models	ROC-AUC
GCN	0.7606 ± 0.0097
GIN	0.7558 ± 0.0140
PNA	$\mathbf{0.7905 \pm 0.0132}$
Ours	0.7074 ± 0.0422

Models that incorporate edge features yield slightly better results. Our model with no edge attributes performed good compored to *SOTA* models. However, we weren't able to fully leverage the expressive power of the multi-relational data because of the limited computational power for our RGCN model.

4.4 Conclusion

This project enabled use to discover the new field that is Graph Neural Networks and its implementation thanks to the Pytorch Geometric framework. It gave us a better understanding of the general mindset on how to implement a GNN and also how the graph data is represented thanks to the OGB platform. The *ogbg-molhiv* dataset also helped us discover other potential bottlenecks that deep learning has to offer for example an unbalanced dataset and choosing a good loss function. To improve our current models, we could also play on other hyperparameters and try to reduce problematic phenonemons in GNNs. For instance, layers less prone to oversmoothing seem to be a good strategy to explore was introduced in [16]. Many other graph convolutional layers could be tested and employed. Depending on training speed, we could try out larger layers and augment the number of epochs in order to reduce the standard deviation of our results.

References

[1] Joan Bruna et al. *Spectral Networks and Locally Connected Networks on Graphs*. 2014. arXiv: 1312.6203 [cs.LG].

[2] Gabriele Corso et al. *Principal Neighbourhood Aggregation for Graph Nets*. 2020. arXiv: 2004.05718 [cs.LG].

[3] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG].

[4] Matthias Fey, Jan-Gin Yuen, and Frank Weichert. *Hierarchical Inter-Message Passing for Learning on Molecular Graphs*. 2020. arXiv: 2006.12179 [cs.LG].

[5] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

[6] Greg Landrum et al. *Open-source cheminformatics*. Version 0.20.2. 2006. URL: https://www.rdkit.org.

[7] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 3–4.

[8] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 3–4.

[9] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), p. 58.

[10] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), p. 63.

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI].

[12] Weihua Hu et al. *Open Graph Benchmark: Datasets for Machine Learning on Graphs*. 2021. arXiv: 2005.00687 [cs.LG].

[13] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[14] Soheil Kolouri et al. "Wasserstein Embedding for Graph Learning". In: *arXiv e-prints*, arXiv:2006.09430 (June 2020), arXiv:2006.09430. arXiv: 2006.09430 [cs.LG].

[15] Kezhi Kong et al. *FLAG: Adversarial Data Augmentation for Graph Neural Networks*. 2020. arXiv: 2010.09891 [cs.LG].

[16] Meng Liu, Hongyang Gao, and Shuiwang Ji. "Towards Deeper Graph Neural Networks". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining* (July 2020). DOI: 10.1145/3394486.3403076. URL: http://dx.doi.org/10.1145/3394486.3403076.

[17] C. Merkwirth and T. Lengauer. "Automatic generation of complementary descriptors with molecular graph networks". In: *Journal of Chemistry Inf. Model* (2005).

[18] Christopher Morris et al. *Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks*. 2020. arXiv: 1810.02244 [cs.LG].

[19] Trang Pham et al. *Graph Classification via Deep Learning with Virtual Nodes*. 2017. arXiv: 1708.04357 [cs.LG].

[20] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML].

[21] Petar Veličković et al. "Graph Attention Networks". In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=rJXMpikCZ.

[22] Wikipedia. *HIV — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=HIV&oldid=1006111089. [Online; accessed 14-February-2021]. 2021.

[23] Muhan Zhang et al. "An End-to-End Deep Learning Architecture for Graph Classification". In: *AAAI*. 2018, pp. 4438–4445.