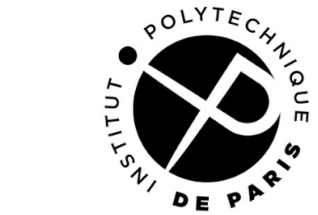

INSTITUT
POLYTECHNIQUE
DE PARIS

ALTeGraD

Project Report

M2 Data Science 2023-2024

Victor Gertner, Irshath Nagouri

1 Introduction

The goal of this data challenge, involving both text and graph data, is to rank molecules represented as graphs based on a given textual query. To tackle this task effectively, a dual-model approach is proposed. One sub-model handles graph data, while another processes text data. The challenge is to train both sub-models simultaneously, aiming to encode graph and text data into a shared latent space. However, limited GPU memory on Kaggle presents a size constraint. Thus, it is crucial to carefully analyze the trade-off between the sizes of the graph and text encoders to ensure optimal model performance within the resource constraints imposed by the Kaggle GPU.

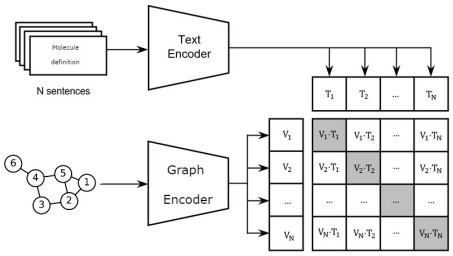

FIGURE 1 – Architecture of our model

Data :

Here is one exemple of graph found in the dataset and its textual data associated.

— **Textual description** : Glutaric acid is an alpha,omega-dicarboxylic acid that is a linear five-carbon dicarboxylic acid. It has a role as a human metabolite and a Daphnia magna metabolite. It is a conjugate acid of a glutarate(1-) and a glutarate.

— **Graph Data :**

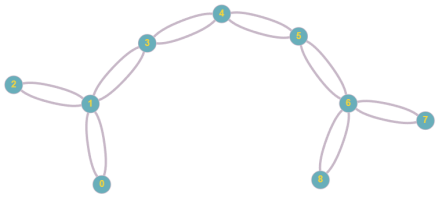

FIGURE 2 – Graph associated

As we can see the description is about the structure of our molecule and at the same time about its properties. Moreover here we have a simple molecule that has only 8 atoms, but in the dataset we find graph with way more nodes(250) and edges. Also the example shown only has words in terms of description but one can find in the dataset the use of atoms name inside the text to better describe the structure of the data. In terms of data, we are given node features where each feature represent a certain atom. We need to leverage this in our architecture.

In terms of number, we possess 26 410 training data, 3 303 validation data and also 3 303 test data.

2 Graph Encoder

We first started by tackling the graph encoder part. We needed to find a graph neural network architecture that would be able to really differentiate the different molecules we have.

We present below the different encoders we tested.

GIN :

Our first idea was to go for a model that has good performance with respect to the W-L test. Indeed, as seen the the class, a significant challenge highlighted in working with graphs is the task of discriminating non-isomorphic graphs. Thus, we looked in the literature and found that in [5], they proposed an architecture that is as powerful as the W-L graph isomorphism test. To do so, they propose a new strategy based on the mechanism of GNN. The classic mechanism of a GNN is given by the equation below :

$$a_v^{(k)} = \text{AGGREGATE}(k) \odot \left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\}, \quad h_v^{(k)} = \text{COMBINE}(k) \left\{ h_v^{(k-1)}, a_v^{(k)} \right\}$$

And the choice of the Aggregate and Combine function are essential to how the GNN is going

to perform in a certain task. In the paper, they prove that they can find two injective functions for aggregation and combine such that the discriminative power is the same as the W-L test. And in order to find those functions, they propose to train MLP to learn those. We can even take one MLP, as a MLP can learn the composition of functions. It gives us this :

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)})\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

In this formula, ϵ determines the importance of the target node compared to its neighbors (it has the same importance if $\epsilon = 0$). It can be a learnable parameter or a fixed scalar. This model permitted us to have better performance than the baseline in the same number of epochs, but we struggle to have really high performance with it, and it has a lot of configuration possible knowing that the MLP can be of any type. Maybe we just didn't found the best MLP for this challenge.

Attentive FP :

Another architecture we tried is known as Attentive FP [7]. The Attentive FP (Fingerprint) network introduces a novel graph neural network architecture specifically tailored for molecular representation in chemical compounds.

The Attentive FP architecture consists of two stacks : one stack for atom embedding, and another for full molecule embedding. Through careful design, the model embeds atoms and bond features using graph attention mechanisms within atom embedding layers.

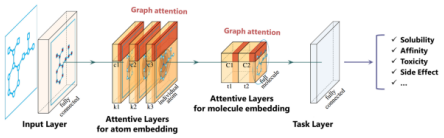

FIGURE 3 – Architecture of Attentive FP

Let's first detail how the atom embedding step works. In this process, each atom is represented by an initial state vector, generated by considering the atom's characteristics such as its symbol, degree, formal charge... Notably, the vectors of atomic features and neighboring atomic features may differ in length, requiring a linear transformation and nonlinear activation to unify their dimensions. Let's denote the initial state vector of a target atom v by h_v^0. The Attentive FP will produce a series of messaging phases and readout phases to obtain the final state vector of each atom. This process is explained by the equations below for $k \geq 1$.

Messaging :

$$C_v^{k-1} = \sum_{u\in N(v)} M^{k-1}(h_u^{k-1}, h_v^{k-1}) \qquad (1)$$

Readout :

$$h_v^k = GRU^{k-1}(C_v^{k-1}, h_v^{k-1}) \qquad (2)$$

Where C_v^{k-1} is the attention context of the atom v from the set of its neighbors $N(v)$. These equations describe how each atom representation at layer $k-1$ denoted by h_v^{k-1} go through an attention mechanism calculated in the messaging phase (1) enables the target node (or atom) to capture the most relevant features from its neighbors to compute an attention context C_v^{k-1}. This context alongside the state vector from the previous layer h_v^{k-1} are fed into a GRU (Gated Recurrent Unit) as shown in (2) to output a new vector state h_v^k. The GRUs allow the retaining and filtering of relevant information thanks to simplified update and reset gates. This structure helps to achieve a molecular representation where links from different atoms still make a difference if they are relevant to a given molecular structure.

In the Attentive FP network, the transition from atom to molecular embedding involves introducing a virtual supernode that connects to all atoms. Applying the same attention mechanism used for individual atoms to this supernode allows the model to selectively aggregate atom embeddings, creating a comprehensive representation for the entire molecule. This approach ensures a cohesive and effective strategy for encoding structural information, contributing to the model's predictive performance in chemical tasks.

But in the end, this network was not retained in our final model as it required more memory than what was available on the platform. We decided to work with more lightweight networks afterwards.

GAT, our final model :

So this last paper even if we couldn't really have it due to memory issues, made us go towards GAT [6].

A GAT is defined this way :

$$h'_i = \sigma\left(\left(\sum_{j\in N_i} \alpha_{ij} W h_j\right)\right) \tag{3}$$

Where σ is a non linearity and we have :

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\tilde{a}^T[Wh_i||Wh_j]\right)\right)}{\sum_{k\in N_i}\exp\left(\text{LeakyReLU}\left(\tilde{a}^T[Wh_i||Wh_k]\right)\right)} \tag{4}$$

There are several parameters one can tune in a GAT. Firstly, we decided to focus on the number of heads we could have in our model, given the memory allowed by Kaggle. The idea of heads here is the same that the one proposed by 'Attention is all you need'.We acheived at maximum to put 25 heads for each gat. However, we attained our best performance only using 20 heads and not 25. We might explain this difference by the fact that we can't train it for as much hours as we can and that having more than here 20 heads, would result in a model under fitting our data because it was too complex. Also, we found that it was better not to aggregate the output of the GATs but better concatenate each head in order to retain the maximum of information that was gathered.

Moreover, we tried to add dropout in our architecture, but it didn't enhance our performance, we might suspect that we needed even more epochs in order for the model to achieve better performance than the non-dropout model.

Also, we tried to do skip connection in between the layer for the pooling where all the results of each layer where concatenated, but the results weren't performing so we decided not to keep this idea.

Final graph encoder :

Our best performing model is 3 GAT with each 20 heads and separated by a ReLU activation function and then two linear layer in order to project the embeddings into the same space as the one from the text encoder.

Pooling layer :

So now that we tackled the issue of node embedding, we need to go from a graph where each node is embedded in a certain space to a graph embedding. This is what the pooling layer is about.

We tried multiple pooling layer, from the classical global pooling layer proposed (mean,max,sum) to more sophisticated one such as SAGPooling [4] (Self-Attention Graph Pooling) which tries to leverage the idea of attention mechanism in the pooling by using a score of the form

$$Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}^{-\frac{1}{2}}D^{-\frac{1}{2}}X\Theta_{att}) \tag{5}$$

Where σ is the activation function (e.g., tanh), $\tilde{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix with self-connections (i.e., $\tilde{A} = A + I_N$), $\tilde{D} \in \mathbb{R}^{N\times N}$ is the degree matrix of $\tilde{A}$, $X \in \mathbb{R}^{N\times F}$ is the input features of the graph with N nodes and F-dimensional features, and $\Theta_{att} \in \mathbb{R}^{F\times 1}$ is the only parameter of the SAGPool layer.

We thought that this type of polling would actually be really efficient in our case : our nodes can be very different in terms of importance in a molecule, hence having a pooling that takes that in account was an idea we looked a lot forward. However, in both global and hierarchical settings, we couldn't make it perform better than any other pooling.

This is why we decided to stick to the max pooling layer that was used in our best performing model.

3 Text Encoder

In the baseline model, a Distill BERT was given. As shown before, we have a very specific dataset where the text related to molecular properties and we know that the main purpose

of BERT isn't to perform well on very specific data. Hence, we tried to look towards other models. We decided to keep looking at BERT-like architecture, since we are trying to do text embeddings for a non generative task, we need to use as much context as we can. So using a bidirectional model is the best way to capture most of the information contained in the textual data.

Here are all the textual encoders that we implemented or tried to implement :

Distill BERT :

Distill BERT is a good alternative for us, since we are limited by Kaggle's GPU memory. Moreover, as stated by its paper, it reduces the size of a BERT model by 40%, while retaining 97of its language understanding capabilities and being 60% faster. The speed factor is also really important for us, because as far as this we only talked about the GPU memory, but Kaggle as also a time limit, and having a model that is possible to train quickly is a key factor for us. Indeed, if the model is too big, it's going to take a lot of time to train, then we have a debate : should we try to train it more, or should we consider another model.

BioMegatron :

So we found the model BioMegatron in [1], it's a BERT-Like model derived from Megatron that use larger pre-training text corpus. The issue with this model is that it is huge in terms of parameters : 345 million, 800 million or 1.2 billion parameters, and we weren't able to bring it up forward on Kaggle with our graph encoder architecture. One way we tried to solve this was trying to parallelize our model on the two T1 that Kaggle provides, by putting on one GPU our graph encoder and on the other GPU the text encoder, but even this wasn't sufficient in order to load such a model...

BioBert :

So in order to be able to get a model that would fit into our architecture we went into BERT-sized models, and we found BioBert [3]. It's a BERT like model that has been pretrain on large-scale biomedical corpora. To do so, they extract data from PubMed (PubMed abstracts and PMC full-text articles). In the paper, they show that pre-training BERT on biomedical corpora largely improves its performance. Especially in a QA setup. As we saw our project can be seen as close to a QA setup, but instead of retrieving textual data it retrieves a graph. Anyway, our model can leverage the information that this pretraining permitted in order to have more precise embeddings. We know that our task is more on the chemistry side than really biomedical, but when we looked in the litterature for Chemistry adapted Bert, the proposed model are mainly models that are only pretrained on SMILES such as ChemBert or Bert loves Chemistry. However, our text data goes way further than only SMILES. This is why we tried a model like BioBert over some models like ChemBert... However, when having the same training time as our Graph Encoder + Distillbert, we couldn't acheive to make it perform better. We might be able to explain this by the fact that DistillBert has less parameters, hence it's easier for us to train it on a quite low number of epochs. Moreover, our vocabulary is very specific about chemistry whereas BioBert as encounter any vocabulary towards biomedical data which is really broad.

SciBert :

SciBert [2] is a state-of-the-art, pretrained language model designed specifically for handling scientific text. Developed as an extension of the BERT architecture, SciBert addresses the scarcity of high-quality labeled data in the scientific domain by leveraging unsupervised pretraining on a vast and diverse corpus of scientific publications. Its unique focus on scientific language nuances positions SciBert as a valuable resource for researchers and practitioners seeking to enhance natural language processing capabilities in scientific contexts. The results we obtained using SciBert were unfortunately not significantly better than the results we got with Distill Bert. This can be explained by the fact that SciBert is not specialized in chemistry. It is trained on a large set of scientific topics, including for example computer science. But this model is a BERT based

model. So it required much more memory than a simple Distill Bert while not outperforming it.

Final textual encoder :

With what was said right above, we decided to use Distill-BERT as our text.

4 Training setup

So in order to set up the training, we used the main code proposed in the baseline, but we did a few modification of the training hyperparameters.

Batch size :

Firstly, we decided to modify the size of the batch, we know that having a smaller batch induce noise and is often providing better performance. So we tried to go for a batch size of 16, and also a smaller batch size of 8. But this last size was an issue for us, as a smaller batch size increase the training time, our model would take too much time to train for each epochs. This is why we settled for a batch size of 16.

Scheduler We also decided to go for experimentation also on our learning rate which is a key parameter. in order to do so, we tried 3 setups, firstly just a simple linearLR from pytorch, which decrease at each epoch the learning rate. And then we went for a more sophisticated one, the CosineLR from timm. We did variation on this one concerning the periodicity and also if having a decay between each periods was usefull or not. We found the best of those was the CosineLR without decay, which permitted us to obtain our best performance.

Loss :

So we first used the loss provided in the baseline which is a simple contrastive loss making sure that a graph embedding is close to its text embedding (doing it both ways hence the transpose and two BCE).

We then tried to implement another type of contrastive loss which is the triplet loss that tries to leverage more information about our data : positive and negative examples, but we couldn't find a way to really get the positive and negative label, and the loss couldn't be implemented.

Training Time :

We also found that a way to leverage a really good performance was a long training : we first started to only train our model for 10/15 epochs. But after, we tried to make them train way longer, and we had our first best performance after a training of 35 epochs which took us roughly 10 hours. Then we tried to train this model for again 35 epochs, and this is how we achieved our best performance for a total of 23 hours of training.

5 Conclusion

In this challenge, we developed a complete pipeline to tackle the problem of molecule ranking given textual data. As we faced several difficulties such as memory management or the choice of the hyperparameters, we managed to experiment several models for each step of the pipeline. Our final model, consisting of 3 layers of GAT in the graph encoder part and the Distill BERT text encoder, achieved a satisfying tradeoff between performance and memory availability. The final result highlights a score of 0.75 in the public leaderboard.

Références

[1] Evelina Bakhturina Raul Puri Mostofa Patwary Mohammad Shoeybi Raghav Mani Hoo-Chang Shin, Yang Zhang. Biomegatron : Larger biomedical domain language model. 2020.

[2] Arman Cohan Iz Beltagy, Kyle Lo. Scibert : A pretrained language model for scientific text. 2019.

[3] Sungdong Kim Donghyeon Kim Sunkyu Kim Chan Ho So Jaewoo Kang Jinhyuk Lee,

Wonjin Yoon. Biobert : a pre-trained biomedical language representation model for biomedical text mining. 2019.

[4] Jaewoo Kang Junhyun Lee, Inyeop Lee. Self-attention graph pooling. 2019.

[5] Jure Leskovec Stefanie Jegelka Keyulu Xu, Weihua Hu. How powerful are graph neural networks ? 2020.

[6] Arantxa Casanova Adriana Romero Pietro Liò Yoshua Bengio Petar Veličković, Guillem Cucurull. Graph attention networks. 2017.

[7] Xiaohong Liu Feisheng Zhong Xiaozhe Wan Xutong Li Zhaojun Li Xiaomin Luo Kaixian Chen Hualiang Jiang Mingyue Zheng Zhaoping Xiong, Dingyan Wang. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. 2017.