Data Challenge report – Course ”Advanced Learning for TExt and GRAph Data” – M2 Data Science

1 Introduction

Advancements in machine learning on graphs have significantly impacted chemistry, particularly in predicting molecular, enzymatic, and protein properties. This research area, driven by its potential for industrial and medical applications, seeks innovative solutions to overcome the limitations of traditional methods in classifying and quantifying molecular properties. The introduction of Natural Language Processing (NLP) technologies, notably transformers in 2017 [1], has paved new pathways for addressing these challenges. This project aims to leverage the advancements in NLP through the integration of graph and text embedding models, developing a pipeline for accurately describing molecular properties in plain text to enhance molecular property prediction.

The potential of transformer-based models, combined with graph-based approaches, offers promising opportunities for molecular property prediction. This synergy aims to address the challenges of molecular property prediction by incorporating structural and stereochemical information, facilitating advancements in drug discovery and other applications.

2 Challenge description

The aim of the challenge is to map molecule graphs with their plain text properties description. We work with a method called contrastive learning [2], which consists in embedding both structured data in a way that the matching inputs are close one from another in the embedding space. Over the training, the embedding weights are modified in order to pull the matching data embedding and push the different elements apart. Then by evaluating the distance between all the embedding we can deduce a solution to the matching problem.

2.1 Presentation of the dataset

For the purpose of simplicity we will not explore in this section the way data is stored and linked, we simply want to provide a quick reminder about the data that will enter the pipeline for better understanding of the following sections.

In total we have 102981 graphs representing molecules. The data we have does not present all the characteristics of the molecules, simply the atoms (nodes) embedding and their links (which do not contain any information apart from their existence). These graphs are splitted into 4 categories already defined: train (26408 graphs), validation (3301 graphs), test (3301) and unlabeled. The 4th category is not explicitly defined it is simply all the remaining graphs.

We also have molecules properties descriptions, that are in plain text. They are split into train, validation and test, where for the first two categories we can access the matching graph in the respective split. The goal of the project is to match as perfectly as possible the graphs and the descriptions of the test datasets.

2.2 Metrics

We use a classical metric for matching problem which is the mean reciprocal rank (MRR). It does not only evaluate the accuracy of the matching (in terms of best guess only), but computes the score based on the rank attributed to the correct description for each graph (or vice-versa). For a sample of queries Q, the computation formula is

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i} \tag{1}$$

with rank_i the rank of the correct prediction for the i-th query.

3 Our approach

The task we want to achieve is, given a text query and list of molecules, to retrieve the molecule corresponding to the query, this is a classification task. The molecules are represented as graphs, and contain no complementary textual information. Therefore, the model has to be able to tackle two types of representations: texts and graphs. This is why we have to train two different encoders, each one suited for its attributed type of data representation, and then to combine them thanks to contrastive learning.

To handle this challenge, our strategy has been to start with methods that are simple and practical to implement, and move on to techniques that are more complex and time-consuming. We have focused on the Graph Encoder architecture, as the Text Encoder already supplied, BERT [3], has been widely proven in the literature.

3.1 The chosen model

The choice of the Graph Isomorphism Network (GIN) for our graph encoder was motivated by its foundational principle, which leverages the Weisfeiler-Lehman (WL) graph isomorphism test. The WL test is a classical algorithm for graph isomorphism that iteratively aggregates and hashes neighboring node labels to identify unique structures within graphs. This process effectively captures the topological information of graphs, allowing for a distinction between non-isomorphic structures. GIN, proposed by Xu et al. [4], extends this concept by learning an injective function to aggregate neighborhood features. This approach ensures that the representation power of GIN is at least as powerful as the WL test in distinguishing graph structures, making it highly effective for encoding molecular graphs where the precise arrangement of atoms and bonds is crucial.

The graph isomorphism operator from Xu et al. [4] is given as follows

$$x'_i = h_\theta\left((1+\epsilon)\cdot x_i + \sum_{j\in\mathcal{N}(i)} x_j\right)$$

or

$$X' = h_\theta\left((A + (1+\epsilon)\cdot I)\cdot X\right),$$

where h_θ denotes a neural network, i.e. an MLP, and ϵ is a learnable parameter or a fixed scalar that can be used to weigh the importance of a node's own features versus the features of its neighbors.

GIN addresses a critical limitation in previous graph neural networks (GNNs), which struggled to capture the full complexity of graph structures due to overly simplistic aggregation functions. By optimizing the aggregation function to be as expressive as the WL test, GIN ensures that graph-level representations are sensitive to even subtle differences in local graph topologies. This capability is particularly important in our task, as retrieving specific molecules from a textual query requires a high degree of accuracy in encoding graph representations.

The decision to employ GIN over other GNN variants was also influenced by its simplicity and efficiency. Unlike more complex models that might require extensive computational resources or intricate architectural adjustments, GIN provides a straightforward yet powerful framework for learning graph representations. This balance between simplicity and expressiveness aligns with our strategy of implementing practical and effective solutions, making GIN an ideal choice for our graph encoder architecture.

Therefore, our final model includes three GIN layers. We used the `conv.GINConv` class from `torch-geometric`, which is exactly the implementation of the Xu et al. [4] network. This allowed us to improve our model by adding simple yet effective techniques. First, Batch Normalization helps in stabilizing the learning process and improving convergence by ensuring that the distribution of the inputs to activation functions remains more consistent during training.

Also, by randomly setting a fraction of input units to 0 at each update during training time, Dropout forces the network to learn robust features that are useful in conjunction with many different random subsets of the other neurons. After running multiple tests, we chose a 0.1 ratio, and we placed it between the hidden layers.

The final stages of our architecture are crucial for transforming the graph-encoded features into a representation suitable for downstream tasks. This transformation is achieved through a sequence of global mean pooling, non-linear activation, regularization, and linear transformation steps.

The global mean pooling operation aggregates node features across the entire graph to produce a single vector representation of the graph. This step is essential for tasks that require a graph-level output, as it condenses the information distributed across all nodes into a unified form. Mathematically, for a graph G with nodes $\{v_1, v_2, \ldots, v_n\}$, the global mean pooled feature x_G is computed as:

$$x_G = \frac{1}{n} \sum_{i=1}^{n} x_{v_i}$$

where x_{v_i} represents the feature vector of node v_i. This operation ensures that the output is invariant to the graph's size and node order, making it particularly suitable for analyzing molecular structures where the overall composition is more significant than the specific arrangement of atoms.

Following pooling, the features undergo a transformation through a fully connected layer, and are then passed through a Rectified Linear Unit (ReLU) activation function. The ReLU activation introduces non-linearity, enabling the model to capture complex relationships in the data.

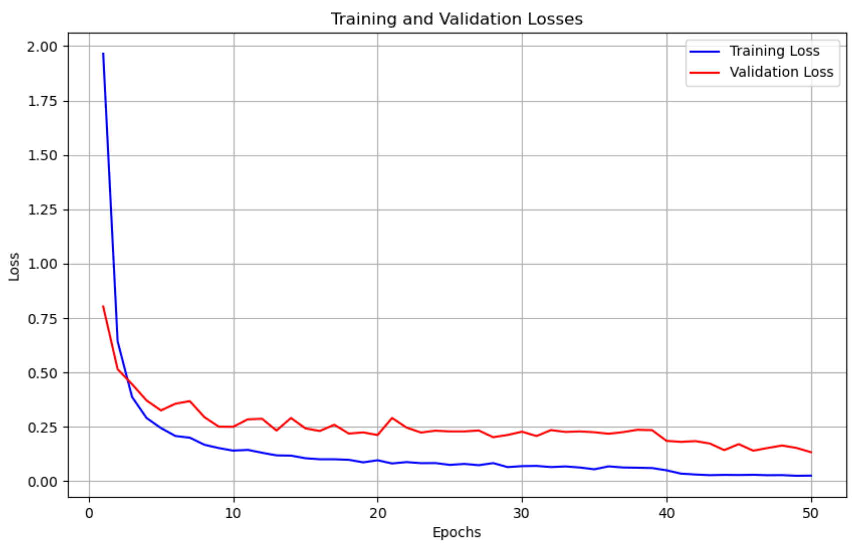

Figure 1: After the initial sharp decline, both losses begin to flatten out, indicating that the model is starting to converge. The training loss continues to decrease gradually, suggesting the model is still learning but at a reduced rate. The validation loss also decreases slightly but remains relatively flat, which is a good sign that the model is not overfitting significantly.

The processed features are then passed through another fully connected layer, which maps them to the desired output dimensionality. This layer serves as the final step in the model, producing the output vector x.

After multiple tests, we chose a dynamic learning rate as we found that model learning tended to slow down or even stagnate.The resulting training is not optimal, but it has been improved compared to when we used a constant learning rate.

Finally, to ensure the reproducibility of our results and the stability of our model's training process, we adopted a deterministic approach by setting fixed seeds for random number generation and configuring PyTorch's backend behavior. This approach is critical in deep learning experiments, where even slight variations in initialization or computational procedures can lead to notably different outcomes.

Our chosen model allowed us to achieve a 0.78 score on Kaggle, and we obtained the presented train and validation loss curves 1. It took us almost 10 hours, using the Planetary Computer Hub, which provided us with a T4 GPU, to train our model on 50 epochs.

3.2 The other models

During the challenge, we tried multiple architectures presented in class, such as GraphSage [5], Graph Attention Network (GAT) [6], Graphormer [7], adding a Positional Encoding layer [8], and even the Weisfeiler and Leman go Hyperbolic Network (WLHN) [9]. Kaggle scores obtained with these models are presented in 1.

Yet promising, since these methods achieve state-of-the-art performance in graph encoding tasks, we got way better results with way more simple architectures. An explanation for that was a difficulty we had to tackle during the whole challenge: computational resources.

Network	Kaggle Score	Epochs
GraphSage	0.48	10
GAT	0.54	15
Graphormer	0.4	30
Positional Encoding	0.65	30
WLHN	0.45	50

Table 1: These scores were obtained after different numbers of epochs, depending on the time and computational resources we had at our disposal.

In these papers, the model was trained for hundreds of epochs, a grid search was done to keep the optimal hyperparameters, and the authors had the time and computational resources to run these tests.

We also explored pre-training tasks in order to use the large amount of unlabeled graphs provided. However we did not quite succeed in finding a suitable unsupervised learning task in order to improve our training performance. Our difficulty essentially resulted in finding a task that would help us in finding a representation linked to molecule properties without using any notion of properties in our pre-training. We essentially worked with graph reconstruction but this was creating graphs that were not viable molecules, which could be the reason behind the fact that it didn't work.

Our code is available on the following `Github repository`: `https://github.com/marionchadal/ALTeGraD-2024-Data-Challenge-Molecule-Retrieval-with-Natural-Language-Queries.git`

Biblography

[1] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. 2017, pp. 5998–6008. URL: `https://arxiv.org/abs/1706.03762`.

[2] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding". In: *arXiv:1807.03748* (2018). URL: `https://arxiv.org/abs/1807.03748`.

[3] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[4] Keyulu Xu et al. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2018).

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[6] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[7] Chengxuan Ying et al. "Do transformers really perform badly for graph representation?" In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28877–28888.

[8] Grégoire Mialon et al. “Graphit: Encoding graph structure in transformers”. In: *arXiv preprint arXiv:2106.05667* (2021).

[9] Giannis Nikolentzos, Michail Chatzianastasis, and Michalis Vazirgiannis. “Weisfeiler and Leman go Hyperbolic: Learning Distance Preserving Node Representations”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 1037–1054.