Molecule Retrieval with Natural Language Queries

Lucas Gascon
MVA
lucas.gascon@polytechnique.edu

Hippolyte Pilchen
MVA
Hippolyte.pilchen@polytechnique.edu

Pierre Fihey
IP Paris
pierre.fihey@polytechnique.edu

Abstract

By enabling the extraction of vector representations from graph-structured data, Graph Neural Networks (GNNs) make it possible to study such data structures more efficiently and to relate them to other types of data, especially textual ones.

The aim of this article is to use these encoding techniques on graphs representing chemical molecules. These encoding will make it possible to retrieve the textual data corresponding to each of these molecules. This can be achieved by co-training a text encoder and a graph encoder using contrastive learning. Through contrastive learning, the model learns to map similar text-molecule pairs closer together in the learned representation space while pushing dissimilar pairs apart. In this article, we present the various experiments we've been able to carry out on the architecture of these encoders, as well as on how to train them jointly.

1 Introduction

Natural language analysis has seen exceptional growth since the introduction of Transformers in 2017 [10], with new language models with exceptional performance. They are adaptable to capture characteristics of any type of textual data. In particular, we have seen the development of language models adapted to scientific texts ([5], [2]). On the other hand, developments in graph neural networks have made it possible to achieve the state of the art in many classification and prediction tasks for graph data. The application of these GNNs to molecular graphs has enabled the development of drug discovery and chemical property analysis tools that have rapidly outperformed older machine learning models ([12], [6],[11],[13]).
The aim of our model is to leverage all these deep learning tools to obtain a successful match between the embeddings generated from graphs and those generated from the text queries that characterize them. The challenge is not just to obtain independent embeddings for each of these data types. In fact, we want the embedding generated from a graph to contain information related to the embedding generated from the text that characterizes it. As we will see in the dataset study, the text provides information on the chemical properties of molecules, their atomic structures and other chemical information. All this information must be found in the embeddings generated by the graph encoder. By jointly training our graph neural network with a text encoder, the aim is for the GNN to be able to capture this specific information, present in other forms in molecular graphs. In this work, we will therefore perform contrastive learning, where the model learns to map text-molecule pairs closer together in the learned representation space while pushing dissimilar pairs apart. All our codes are available in our Github.

2 Dataset

In this section, we describe in detail the dataset used in our study. This dataset is made up of several different types of data, which will be used for the joint learning of our model.

On the one hand, we have a dataset of molecular graphs. Each molecule is represented as a graph, with the nodes representing the atoms and the edges the chemical bonds between them. Each node contains information in the form of a 300-dimensional embedding that characterize the atoms that make up molecules. By analyzing this dataset, made up of 29 709 graphs, we can extract a few characteristics that will be useful for understanding the problem and setting up our model. We can see that we are dealing with moderate-sized graphs, with an average of 32 nodes. However, these sizes are highly diversified: the largest graph is made up of 536 nodes, while the smallest is made up of 2. Above all, these graphs are not very dense, with an average of 33 edges. These observed characteristics are totally logical in the context of our study. Molecule sizes can vary greatly, and atoms can only create a limited number of bonds between themselves.

On the other hand, we have a dataset composed of textual data containing chemical descriptions of the molecules. These descriptions contain information on the chemical structure, atomic configurations, functional groups present, and sometimes the biological roles or applications of the compounds. These are highly technical descriptions, so using a language model on such data will require specific training to capture the specifics.

The graphs and text descriptions of the same molecule are linked to the same index to enable training and validation of our model. Our dataset is therefore made up of pairs of textual data and graphs for each molecule, as we can see below.

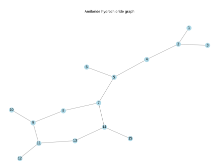
Graph representation of Amiloride hydrochloride	Textual description of Amiloride hydrochloride
	*Amiloride hydrochloride is a hydrochloride obtained by combining amiloride with one molar equivalent of hydrochloric acid. It has a role as a diuretic and a sodium channel blocker. It contains an amiloride(1+).*

Table 1: Representation of a molecular graph with its corresponding text query

We divide these 29 709 pairs into a training dataset (90%) and a validation dataset (10%).

2.1 The evaluation metric: Mean Reciprocal Rank

The metric used to evaluate our submissions is the Label Ranking Average Precision (LRAP). Since there is only one relevant label per sample, this metric is equivalent to the Mean Reciprocal Rank (MRR).

$$MRR = \frac{1}{n\_samples} \sum_{i=1}^{n\_samples} \frac{1}{rank_i} \qquad (1)$$

, with $n\_samples$ the number of text descriptions and $rank_i$ the position of the relevant graph in the list of similarity scores (in descending order). MRR is a widely used evaluation metric in information retrieval and recommendation systems. It assesses the effectiveness of a ranking algorithm by quantifying how well a system prioritizes relevant results, with higher MRR ($\in [0, 1]$) values indicating better performance.

3 Model's architectures

To achieve our objective of retrieving the molecule corresponding to a given text query, our model must logically be made up of two encoders. A graph encoder and a text encoder. These two models must provide embeddings of the same dimension, so that they can be compared with each other using a similarity measure. As described in the hand-out, cosine similarity is designed to compare embedding vectors similarity, we tried other pairwise similarity measure (euclidean and dot product) but cosine similarity gives the best results since it normalizes the vectors and therefore enables to compare the angle between embeddings on the unit sphere.

3.1 Graph Encoder

For the creation of a graph encoder, we studied several different architectures in order to take into account the specific characteristics of molecular graphs.
Given the reasonable size of the graphs in the dataset, we decided to implement a first block of 3 graph convolution layers to efficiently capture the information from each node while avoiding over-smoothing and over-fitting. Then, independently of the first block we added one GATConv layer (convolutional layer with attention mechanisms). These three graph neural networks layers are followed by four linear layers with ReLU activations. This framework of model comes from several empirical observations at the beginning of the challenge: 3 convolution layers seemed optimal due to the size of molecules while attention was also performing very well, increasing the number of linear layers enabled to experiment dropout and also to learn more complex representation of the graphs.

It's in the composition of the first block that we decided to experiment with several different architectures in order to best extract the information contained in the graphs. We will therefore briefly present the layers we have decided to experiment with.

3.1.1 Convolutional layers

Given the rather simple architecture of the graphs at our disposal, we first turned our attention to **Graph Convolutional Networks** (GCN). GCNs operate by aggregating information from neighboring nodes using a weighted average. Their simplicity and low computational cost make them an interesting solution for our model, insofar as training is performed on a rather large database.
The simplicity of GCNs may, however, limit the study of molecular graphs as, for each GCN layer, each node will only contain information from its neighboring nodes. Even if it means that each atom will receive information down to its third neighbors, it may not be enough for the detection of complex functional groups or specific motifs. The GNN may need to consider higher-order information for each atom. That's why we decided to experiment with a **Chebyshev Spectral Graph Convolution** [3]. This architecture is composed of ChebConv layers that use a spectral approximation based on Chebyshev polynomials. These layers enable more flexible convolution filters to be set up, allowing information from more distant nodes to be captured for each node. While these layers are more flexible and can therefore capture more complex patterns and functional groups, the computational cost is also higher.

3.1.2 Attention layers

While this aggregation of information based on a simple average can be very effective, the addition of an attention mechanism can be particularly interesting for a more precise understanding of the interactions between each atom in the molecule. An atom properties may in fact be better defined by some of its neighbors than by others, particularly when defining **functional groups** or extracting chemical properties. Above all, the heterogeneity of the graphs at our disposal calls for a dynamic and adaptable computational model. **Graph Attention Networks** (GATs) were introduced in 2018 [7] to add the attention mechanism to the study of graphs, and the application of these graph-based Neural Networks to molecular graphs very quickly showed relevant results in the recognition of complex intramolecular patterns. By weighting the information provided at each node with an attention score according to the importance of each neighbor, GATs enable a more adaptive understanding of different chemical properties. The final representation of each atom will therefore contain information from every atom influencing it, down to its fourth neighbors in our case, which can be really relevant in an analysis of chemical properties. Moreover, the attention mechanism allows flexibility, as it can adapt to different topologies and sizes of molecules. This makes the model robust and capable of handling various types of molecular structures, which is important given the dataset at our disposal.

These GNN layers all present interesting characteristics for our study, and we therefore decided to compare each of these architectures in order to retain the most optimal one. We were also able to experiment with other GNN layers derived from the three we presented, to see if any improvements could be made.

3.1.3 Batch Normalization et Subpooling

In each of our experiments, a BatchNormalization function is applied to the output of each convolutional layer. Batch normalization simply consists in normalizing each feature within the batch in order to stabilize learning by reducing covariance shift for instance. This function ensures that the inputs to each layer follow the same distribution. This considerably stabilizes training [4].
Finally, as the output of the last GAT layer consists of one embedding per node, a global pooling layer is required to aggregate the information from each node within a single embedding. This enables the model to provide a single embedding containing all the information needed to compare molecular graphs with textual data. Several factors need to be taken into account when choosing the pooling function, and the literature logically recommends the use of sum pooling for molecular graph analysis [1]. This is the pooling function that preserves the most information in a molecule study, especially molecule size, which is really important in our study. We therefore decided to add this *global add pool* at the end of our model to obtain a relevant embedding in the right dimension.

3.2 Text Encoder

We opted for a BERT model to encode textual data, utilizing its Transformer-based architecture and notable for its bidirectional attention mechanism. As the BERT model is pre-trained on very large quantities of data, it

already has in-depth knowledge of natural language. Above all, its architecture allows the model to be finetuned on specific data to capture the particularities of that data. This results in a highly efficient model on specific data, at a lower computational cost.
Given the highly technical descriptions we have available, this finetuning capability is very important in our study, as it will enable our model to generate embeddings capturing information on the chemical properties of molecules, the functional groups that make them up and atomic configurations. The joint training of this language model with the Graph Neural Networks model should enable us to capture this precise information in textual data.

We therefore decided to finetune two models based on the BERT architecture: the DistillBERT model, which retains the performance of the classic BERT model with 40% fewer parameters thanks to the "knowledge distillation" process, and the SciBERT model, a BERT model finetuned on scientific corpora to capture their specificities [2]. In particular, we hope that this second model will adapt more easily to our highly technical data during finetuning. These text encoders finally provide an embedding containing all the information on the text data and which is in the right dimension for comparison with graph embeddings.

4 Pipeline and results

4.1 Learning

Most of our choices come from both empirical observations and intuitions about the data studied.

Batch size is the number of graphs and text queries used to train the model before updating the weights. Increasing the batch size during learning enables to have better results. By comparing a larger number of text queries with a greater variety of graphs, the model can develop a more nuanced understanding of the connections between a graph and a text query, as opposed to randomly selecting a pair, which would be the case when working with a batch size of two. Furthermore, a large batch size speeds up convergence. The main issue with increasing the batch size is that the GPU has a rather limited memory (RAM) and the larger the batch size, the more samples are being propagated through the GNN in the forward pass. One of the main challenge of our project was to increase the batch size during learning while preventing "out of memory" errors. We explored multiple strategies, including gradient accumulation, where weights are updated after aggregating gradients from several batches; gradient checkpointing, which reduces memory usage by not storing all gradients at the cost of higher computational effort through gradient recomputation; and layer freezing. The latter proved to be the most effective, significantly enhancing batch size, hence it became our primary focus.

Performing layer freezing at the beginning of training leaded to poor results. Therefore, we thought of a particular learning method. First, all the layers require a gradient and we set the batch size to a small number (between 30 and 80 depending on the BERT architecture mostly and on the convolutional layers). After a fixed number of epochs we increase the batch size and we freeze the first transformer block of the text encoder. Then, we iteratively freeze the other blocks until the end. The text encoder, since it has already been trained, starts with a pretty good representation of natural language, it "only" needs to grasp the links with graph embeddings. Finally, we freeze the convolutional layers of the graph encoder and the attention layer, leaving only the linear layers unfrozen. The batch size increases according to different policies: first we add 4 of batch-size at each update and after all the BERT layers are freezed we perform an interpolation between a the current value and a final one. Great care must be taken on the policy used for freezing the layers and increasing the batch size in order not to reach the total memory capacity of the GPU.

4.2 Training losses

We explored different loss functions. Initially, we employed a symmetrical loss that builds on the principles of contrastive loss, enhanced with cross-entropy. This approach involves comparing each graph embedding with every text embedding within the batch but does not compare graph embeddings with each other or text embeddings with each other. The objective is to increase the similarity between embeddings of matching pairs of (graph, text) while decreasing it for non-matching pairs.

We added a temperature parameter to this loss function. This parameter helps adjust the similarities before applying the softmax operation in the contrastive learning setup. The role of the temperature parameter is to control the sharpness of the softmax function's output distribution, affecting the similarities between embedding pairs.

Ultimately, we re-implemented and evaluated the Lifted Structured Loss[8], which exhaustively evaluates all pairwise interactions within a batch. From a mathematical standpoint, the Lifted Structured Loss aims to simultaneously minimize the distance between similar pairs while maximizing the separation between dissimilar pairs, covering the entire batch. This goal is achieved through a structured loss mechanism that employs a "lifting" technique, transforming the pairwise distance matrix into a structured configuration. In this setup, the contribution of each pair is recalibrated in relation to the collective context of the batch, ensuring that the loss calculation reflects the overall data structure within a batch, rather than isolating individual pair or triplet connections.

The role of the margin parameter is crucial as it defines the required distinctiveness between dissimilar and similar pairs. Essentially, the margin sets a baseline separation that must be achieved between dissimilar pairs, compelling the model to develop more distinctive features, thereby enhancing its discriminative ability.

In our experiments with Lifted Structured Loss, we varied the distance metric, including tests with cosine distance and mean squared error. However, the most effective results were achieved with the L2 norm.

4.3 Other parameters

4.3.1 Dropout

We noticed that our experiments often led to overfitting on the training dataset. To mitigate this, we introduced Dropout layers (with $p = 0.2$) between the linear layers in the last block of the GraphEncoder model.

	With Dropout	**Without Dropout**
GAT	0.7273	0.6802
GCN	0.7393	0.758
ChebConv	0.7194	0.743

Table 1: Dropout Layers Ablation Studies Across Models: 100-Epoch Training Performance Measured by LRAP on Validation Data

Table 1 shows that, although the results for GraphEncoders based on GCN and ChebConv were not entirely satisfactory, incorporating Dropout was advantageous for GAT models that employ attention mechanisms and are prone to overfitting.

4.3.2 Hyperparameters

Furthermore, we tested the impact of different hyperparameters. For these experiments we used the given loss (cross-entropy loss) as contrastive loss. We were looking for insightful behavior in order to quickly get the best hyperparameters for our best architecture and training procedure. First, we found that the **embedding dimension** has a great impact on the final metric (LRAP). In high-dimensional spaces, comparing two vectors using cosine similarity becomes more challenging due to the "curse of dimensionality". As the number of dimensions increases, the vectors tend to spread out, and the angle between them becomes less informative, most vectors are nearly orthogonal to each other. This phenomenon makes it difficult to distinguish between vectors that might be intuitively similar in lower dimensions. Nevertheless, the output dimension needs to be high enough to capture all the information from the text query and the graph. Therefore, after several experiments, we set the output dimension to 200.

Secondly, the **learning rate** affects convergence of the model. We highlight in Fig.1 that too high learning rate leads to a very slow convergence and the flow gets stuck in local minima. The same goes for the LRAP (Fig. 2) which plateau at lower values for small learning rate. To combine accuracy and fast convergence we found that $5*10^{-5}$ is a rather good value for the learning rate.

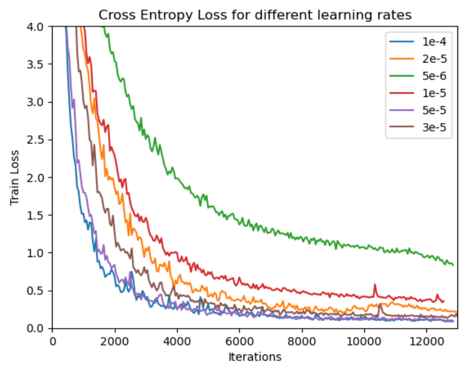


Figure 1: Impact of the learning rate on the training loss of our Model with the Cross-Entropy as contrastive loss. (Model with ChebConv and DistillBERT, without dropout and with the initial cross-entropy loss with a temperature of 0.5).

Finally, we looked at the weight decay. Adjusting the weight decay of the optimizer helps prevent overfitting by penalizing excessively large weights, reducing model complexity, and promoting better generalization to unseen data during training. After a few experiments, we set it to 10^{-2}.

4.4 Final results

Even though, SciBERT is specialized in scientific natural language it does not lead to the best results in the ranking task. Indeed, the train loss is quickly minimized and reaches a low value but it seems to be an overfitting since the LRAP of the model with SciBERT remains smaller than the one of DistillBERT as shown in Fig.3.

After all our experiments, we trained a model which reached 0.7976 of LRAP on the test set. We used ChebConv layers as the first three convolutional layers, DistillBERT and the rest of the architecture is described in Section 3. We freezed layers each two epochs and increased batch size from 80 to 220 by steps of 4. The model is trained using the objective function Lifted Structured loss described in [9] with a margin of 1 and the L-2 norm as distance. The learning rate was set to $5 * 10^{-5}$ and the output dimension to 200.

5 Conclusion

This article examines the fusion of Graph Neural Networks (GNNs) and Natural Language Processing to effectively encode chemical molecules. By jointly training graphical and textual encoders through contrastive learning, we achieved meaningful vector representations that bridge molecular structures and their textual descriptions. The findings underscore the critical role of selecting the appropriate GNN architecture and the positive effects of Dropout layers and hyperparameter tuning on model generalization.

Implementing the Lifted Structured Loss function enabled a comprehensive and efficient training approach, enhancing model performance. Our final model, leveraging ChebConv layers and SciBERT with strategic freezing of layers and batch size adjustments, demonstrated significant results on the test set, showcasing our method's potential for drug discovery and chemical analysis applications. This research lays the groundwork for future exploration into optimizing GNNs and language models for specialized fields.

References

[1] Jana M. Weber Martin Grohe Manuel Dahmen Kai Leonhard Alexander Mitsos Artur M. Schweidtmann, Jan G. Rittig1. Physical pooling functions in graph neural networks for molecular property prediction. 2022.

[2] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *EMNLP*. Association for Computational Linguistics, 2019.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[5] Sungdong Kim Donghyeon Kim Sunkyu Kim Chan Ho So Jinhyuk Lee, Wonjin Yoon and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. 2019.

[6] Joel P. Arrais Luis H.M. Torres, Bernardete Ribeiro. Few-shot learning with transformers via graph embeddings for molecular property prediction. *ELSEVIER*, 2023.

[7] Arantxa Casanova Adriana Romero Pietro Lio Yoshua Bengio Petar Velic ˇkovic , Guillem Cucurull. Graph attention networks. *ICLR*, 2018.

[8] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep Metric Learning via Lifted Structured Feature Embedding, November 2015. arXiv:1511.06452 [cs].

[9] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding, 2015.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[11] Kil To Chong Waqar Ahmad, Hilal Tayara. Attention-based graph neural network for molecular solubility prediction. 2023.

[12] Weiqi Luoa Liangda Fanga Zhao-Rong Laic Jun Wang Xian-bin Yea, Quanlong Guana. Molecular substructure graph attention network for molecular property identification in drug discovery. *ELSEVIER*, 2022.

[13] Sławomir Mucha Krzysztof Rataj Jacek Tabor-Stanisław Jastrzebski Łukasz Maziarka, Tomasz Danel. Molecule attention transformer. 2020.

A Additional plots

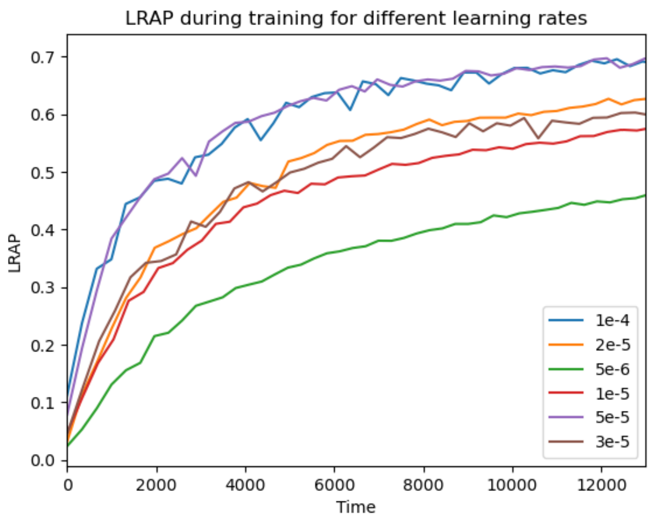


Figure 2: Impact of the learning rate on the LRAP during the training of our Model with the Cross-Entropy as contrastive loss. (Model with ChebConv and DistillBERT, without dropout and with the initial cross-entropy loss with a temperature of 0.5).

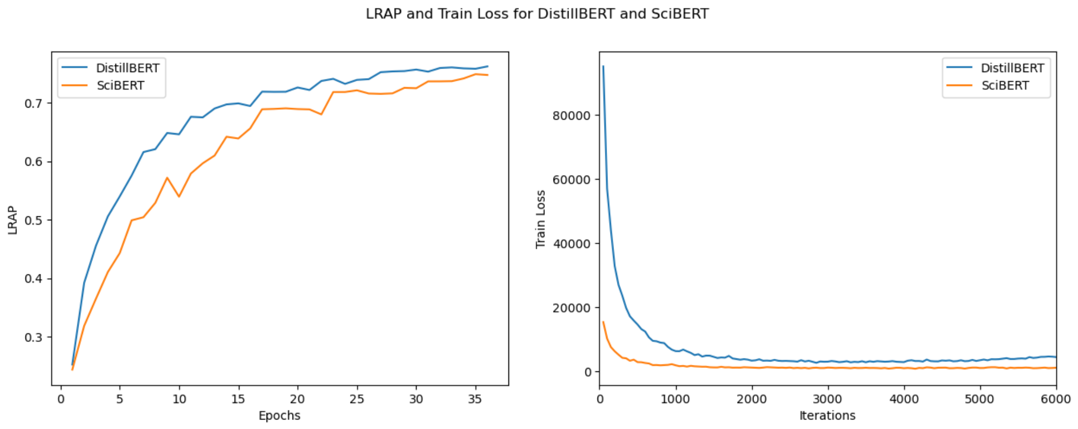


Figure 3: Comparison of SciBERT and DistillBERT trained with the Lifted Structured Loss (with L2 distance and $alpha = 1$) without dropout and with ChebConv layers. Left: evolution of the LRAP during learning. Right: evolution of the training loss during learning.