Molecule Retrieval with Natural Language Queries

Mohamed Aymen Bouyahia, Mustapha Bedoui, Axel Dumont

Février 2024

Introduction / Problem Presentation

Main idea : Bring text embedding and graph embeddings of molecules closer together pairwise.

Argyssfrywff: Ala-Arg-Gly-Tyr-Ser-Ser-Phe-Arg-Tyr-Trp-Phe-Phe is an oligopeptide composed of L-alanine, L-arginine, glycine, L-tyrosine, L-serine, L-serine, L-phenylalanine, L-arginine, L-tyrosine, L-trytophan, L-phenylalanine and L-phenylalanine joined in sequence by peptide linkages.

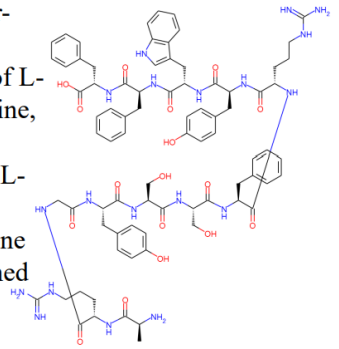


Figure – Example query that is predicted correctly

Baseline

The baseline given co-trains a text encoder and a graph encoder through contrastive learning

Dataset

- **train.tsv** : A tsv file that contains the training data (26408 samples) in the following format : CID, Description
- **val.tsv** : A tsv file that contains the validation data (3301 samples) in the following format : CID, Description
- **token_embedding_dict.npy** : A dictionary mapping molecule tokens to their embeddings.
- **test_text.txt** : A text file contains 3301 textual descriptions from the test dataset.
- **test_cids.txt** : A text file contains 3301 cid (graph ID) from the test dataset.
- **./data/raw/** : A folder contains 102981 cid.graph files.

Dataset

```
['167161',
 '53477645',
 '90546',
 '444796',
 '25203706',
 '135563708',
 '73271',
 '91666374',
 '23421197',
 '54730539',
 '49792034',
 '22035237',
 '25201443',
 '52951747',
 '51351667',
 '25164000',
 '5889665',
 '91861779',
 '5281252',
 '7567127',
 '6741',
```

(a) test_cids.txt

	Xanthurenate is a quinolinemonocarboxylate that is the conjugate base of xanthurenic acid, obtained by deprotonation of the carboxy group. It has a role as an animal metabolite. It is a conjugate base of a xanthurenic acid.
0	4,4-dimethylcholest-8(14)-en-3beta-ol is a 3be...
1	Glas#24 is an ascarosyloxycarboxylic acid beta...
2	5'-(N(6)-L-lysine)-L-tyrosylquinone is an L-ly...
3	Diphenyl ether is an aromatic ether in which t...
4	4-hydroxy-6-(13-methyl-2-oxononadecyl)pyran-2-...
...	...
3295	(18S)-resolvin E1 is a resolvin that is (6Z,8E...
3296	(4-hydroxyphenyl)acetaldehyde is an alpha-CH2-...
3297	Acephate is a phosphoramide that is methamidop...
3298	L-tryptophanamide(1+) is the conjugate acid of...
3299	6-bromoindirubin-3'-oxime is a member of the c...

3300 rows × 1 columns

(b) test_text.txt

	25244316	UDP-alpha-D-galactofuranose(2-) is a UDP-D-galactofuranose(2-) in which the anomeric centre of the galactofuranose moiety has alpha-configuration. It is a conjugate base of an UDP-alpha-D-galactofuranose.
0	46931111	2-acetamido-2-deoxy-3-O-(4-deoxy-alpha-L-threo...
1	743	Glutaric acid is an alpha,omega-dicarboxylic a...
2	6151363	All-trans-4-oxoretinal is a retinoid that is a...
3	7344	Ethyl 2-hydroxypropanoate is the ethyl ester o...
4	170780	Alstonine(1+) is an organic cation resulting f...
...	...	...
26402	3014955	3-chlorobenzoate is a chlorobenzoate that is t...
26403	10315720	NK372135B is a dinitrile that is butanedinitri...
26404	52921823	(14Z,17Z,20Z,23Z,26Z)-dotriacontapentaenoic ac...
26405	53355130	7-epi-zeaenol is a macrolide that is a C-7 epi...
26406	5282451	Pitavastatin calcium is the calcium salt of pi...

26407 rows × 2 columns

Figure – train.tsv

State of the Art - Text2mol

Carl Edwards, ChengXiang Zhai, Heng Ji already imagined this model to get molecules from natural queries :

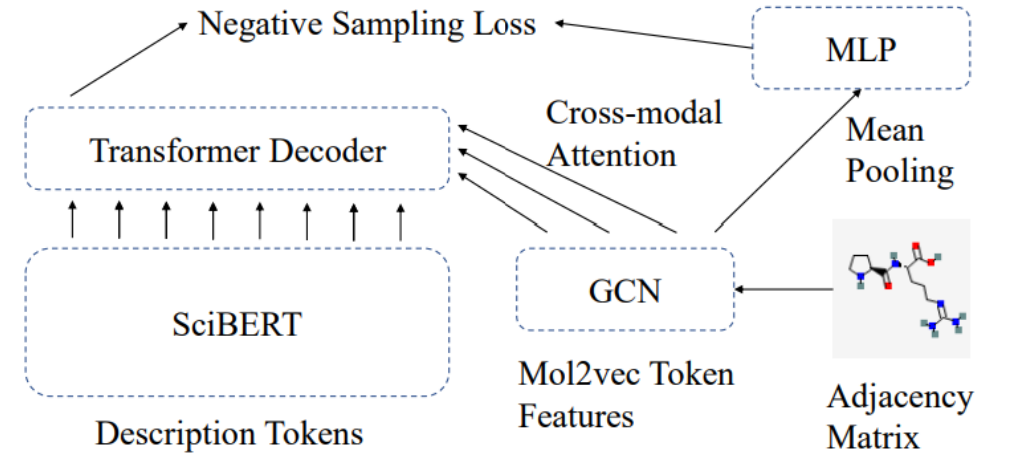


Figure – Text2mol architecture

Takeout

Interesting to understand more completely what could be implemented. We based off our hyperparameters following their experiments.

Table of contents

1. Our Work
   - GAT
   - GIN
   - Implementation
2. Results
   - Loss
   - Final LRAP
3. Ideas to go further
   - Computational Limitations
   - GraphSAGE

Graph Attention Networks

Introduced in 2017 by Veličković et al., GATs relies on a the idea that some nodes are more important than others.
⇒ In this context, we talk about *self-attention* (and not just attention) because inputs are compared to each other.

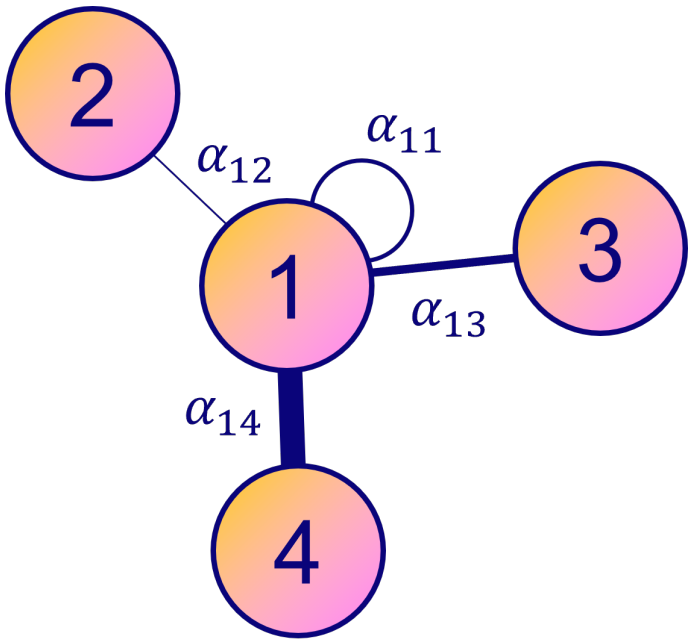


Figure – GAT illustration

In the previous figure, self-attention calculates the importance of nodes 2, 3, and 4's features to node 1. We denote $\alpha_{i,j}$ he importance of node j's features to node i. The GAT layer calculates the embedding of node 1 as a sum of attention coefficients multiplied by a shared weight matrix W :

$$h_1 = \sum_{j \in \mathcal{N}_i} \alpha_{1,j} W x_j$$

To get the attention coefficients :

- Linear Transformation
- Activation
- Softmax
- Multi-head Attention

Other possibility, Graph Isomorphic Networks

GIN was designed by researchers trying to maximize the **representational power** of a GNN. Xu et al. designed a new aggregator that is proven to be as good as the Weisfeiler-Lehman test to distinguish isomorphic graphs in 2018. The result was to learn two injective functions with an MLP :

$$h_i = MLP\left((1+\varepsilon)x_i + \sum_{j \in \mathcal{N}_i} x_j\right)$$

For the global pooling, we then use the sum and concatenate :

$$h_G = \sum_i h_i^0 || \cdots || \sum_i h_i^k$$

Text Encoder

We decided to implement Transformer Encoder Layers as well as an MLP to compute a weighted sum and learn the weights. We also froze the parameters of the pretrained model.

Main Idea

We are trying to reduce the number of parameters that are to be learned and thus reduce the size of the model

GAT Encoder

```python
class GraphEncoder(nn.Module):
    def __init__(self, num_node_features, nout, nhid, heads=1):
        super(GraphEncoder, self).__init__()
        self.nhid = nhid
        self.nout = nout
        self.heads = heads
        self.relu = nn.LeakyReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.ln = nn.LayerNorm(nout)
        self.bn = BatchNorm(nhid * self.heads)

        # Define the GAT layers
        self.conv1 = GATv2Conv(num_node_features, nhid, heads=self.heads)
        self.conv2 = GATv2Conv(nhid * self.heads, nhid, heads=self.heads)
        self.conv3 = GATv2Conv(nhid * self.heads, nhid, heads=self.heads)
        self.mol_hidden1 = nn.Linear(nhid * self.heads, nhid)
        self.mol_hidden2 = nn.Linear(nhid, nout)
```

```python
def forward(self, graph_batch):
    x = graph_batch.x
    edge_index = graph_batch.edge_index
    batch = graph_batch.batch

    # Apply the GAT layers
    x = self.conv1(x, edge_index)
    #x = self.bn(x)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.conv2(x, edge_index)
    #x = self.bn(x)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.conv3(x, edge_index)
    #x = self.bn(x)
    x = self.relu(x)
    x = self.dropout(x)

    x = global_mean_pool(x, batch)
    x = self.mol_hidden1(x).relu()
    x = self.mol_hidden2(x)
    x = self.ln(x)

    return x
```

- `GatConv`: $e_{ij} = \text{LeakyReLU}(W_{att}^{t}[\mathbf{W}x_i \parallel \mathbf{W}x_j])$
- `Gatv2Conv`: $e_{ij} = W_{att}^{t}\text{LeakyReLU}(\mathbf{W}[x_i \parallel x_j])$

Figure – Difference between GAT and GATv2

GIN Encoder

```python
class GIN(torch.nn.Module):
    """GIN"""
    def __init__(self, num_node_features, nout, nhid):
        super(GIN, self).__init__()
        self.conv1 = GINConv(
            nn.Sequential(nn.Linear(num_node_features, nhid),
                          nn.BatchNorm1d(nhid), nn.ReLU(),
                          nn.Linear(nhid, nhid), nn.ReLU()))
        self.conv2 = GINConv(
           nn.Sequential(nn.Linear(nhid, nhid), nn.BatchNorm1d(nhid), nn.ReLU(),
                          nn.Linear(nhid, nhid), nn.ReLU()))
        self.conv3 = GINConv(
            nn.Sequential(nn.Linear(nhid, nhid), nn.BatchNorm1d(nhid), nn.ReLU(),
                          nn.Linear(nhid, nhid), nn.ReLU()))
        self.lin1 = nn.Linear(nhid*3, nhid*3)
        self.lin2 = nn.Linear(nhid*3, nout)
```

```python
def forward(self, graph_batch):
    x, edge_index, batch = graph_batch.x, graph_batch.edge_index, graph_batch.batch

    # Node embeddings
    h1 = self.conv1(x, edge_index)
    h2 = self.conv2(h1, edge_index)
    h3 = self.conv3(h2, edge_index)

    # Graph-level readout
    h1 = global_add_pool(h1, batch)
    h2 = global_add_pool(h2, batch)
    h3 = global_add_pool(h3, batch)

    # Concatenate graph embeddings
    h = torch.cat((h1, h2, h3), dim=1)

    # Classifier
    h = self.lin1(h)
    h = h.relu()
    h = F.dropout(h, p=0.5, training=self.training)
    h = self.lin2(h)

    return h
```

Text Encoder

```python
class TextEncoder(nn.Module):
    def __init__(self):
        super(TextEncoder, self).__init__()
        self.model_type = "Transformer"

        # Use pretrained BERT model as encoder
        self.encoder = AutoModel.from_pretrained("allenai/scibert_scivocab_uncased")

        self.nhid = self.encoder.config.hidden_size
        encoder_layers = nn.TransformerEncoderLayer(self.nhid, 4, dropout = 0.3, dim_feedforward = self.nhid)
        self.intermediate = nn.TransformerEncoder(encoder_layers, 2)
        # Define MLP for learning weights
        self.mlp = nn.Sequential(
            nn.Linear(self.nhid, self.nhid),
            nn.ReLU(),
            nn.Linear(self.nhid, 1),
            nn.Softmax(dim=1)
        )
```

Figure – Layers of the text encoder

Idea : temperature parameter

```
class GraphEncoder(nn.Module):
    def __init__(self, num_node_features, nout, nhid, heads=1, temp=nn.Parameter(torch.Tensor([0.07]))):
```

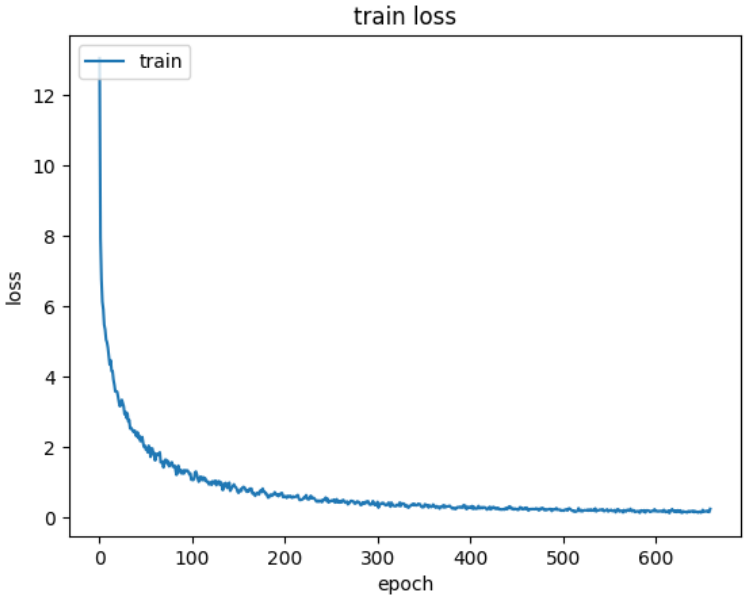
```
self.temp = temp
```

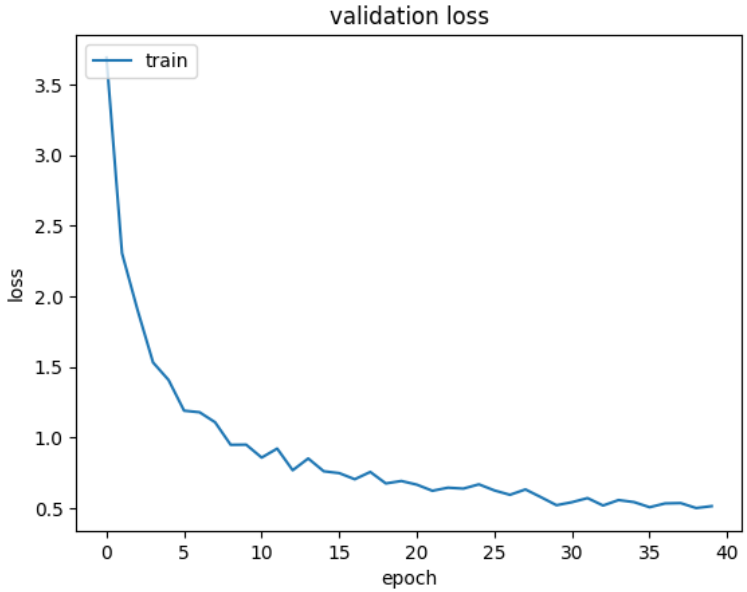
```
x = global_mean_pool(x, batch)
x = self.mol_hidden1(x).relu()
x = self.mol_hidden2(x)
x = self.ln(x)

x = x * torch.exp(self.temp)

return x
```

Figure – Implementation of the temperature parameter

This was inspired by Text2mol, but this was abandoned because it didn't seem to improve visibly the loss or the accuracy

Training and validation loss





We stop at 40 epochs following the hyperparameter of Text2mol, added to the fact that the gains were minimal after. We wanted to keep the solution as fast as possible.

LRAP / Submission accuracy

submission (2).csv Complete · Axel Paul Dumont · 10d ago		0.684	0.6866
submission.csv Complete · Axel Paul Dumont · 10d ago		0.6633	0.6768

Figure – Submissions for GIN and GAT encoders

```
similarity = cosine_similarity(text_embeddings, graph_embeddings)
labels = np.eye(len(similarity))

print(label_ranking_average_precision_score(labels, similarity))
```

Figure – Implementation of LRAP

We get substantially the same LRAP for both models.

Computational Limitations

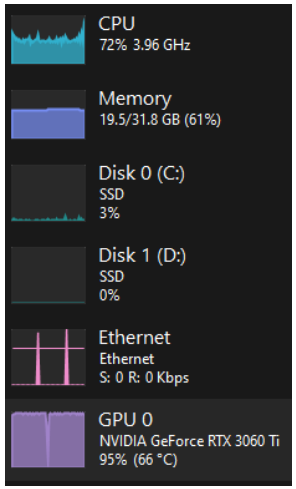


Figure – Local computational cost

Despite having quite good material for the GPU/CPU, the most limiting factor was the computational cost. That was the case even despite the 30 hours of P100 offered by Kaggle, especially for finetuning the hyperparameters via testing.

GraphSAGE

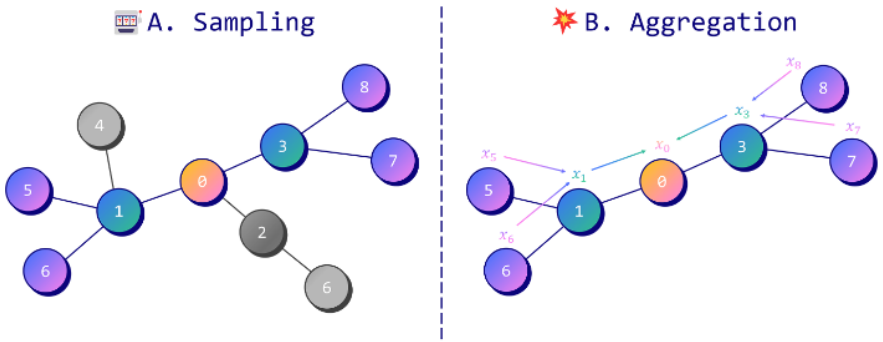


Figure – Theory of GraphSAGE

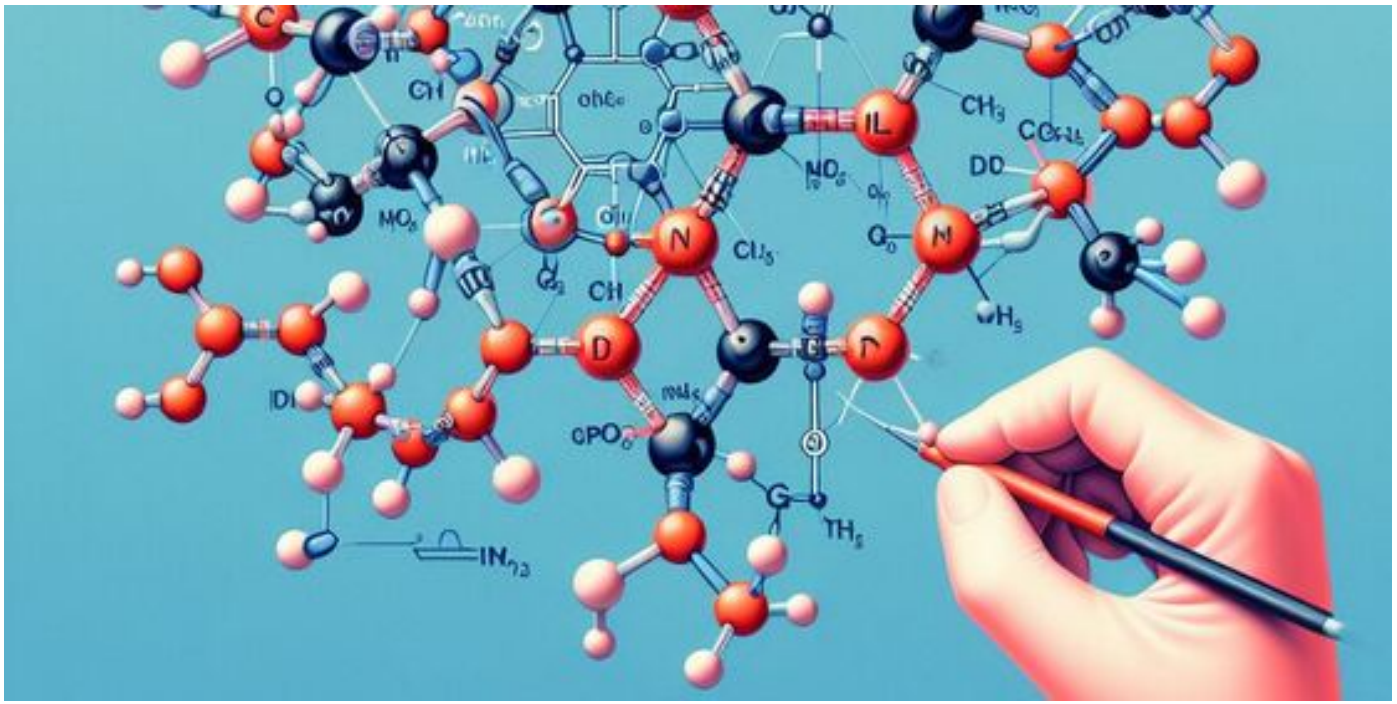
```
self.sage1 = SAGEConv(dim_in, dim_h)
self.sage2 = SAGEConv(dim_h, dim_out)
```

Figure – Implementation

Advantage

The main advantage of this method is that GraphSAGE is reputed to be way faster (up to 88 times) than GCN of GAT

Conclusion



Thank you for your attention !