Introduction

Overview

This notebook is a draft of the in-progress reproduction of the paper "Multi-modal Molecule Structure-text Model for Text-based Retrieval and Editing" originally authored by Liu et al. (2023).

In this section we provide background on the purpose of this research, as well as an overview of the approach concerning structural and textual representation of molecules, as well as the criticality of pretraining for the MoleculeSTM modal.

Please refer to the orginal source code from the paper here, and my own GitHub repo which has been modified for this project here.

Background

Drug discovery is a complex, time-consuming, and costly process that involves identifying molecules with therapeutic potential. To that point, recent progress in artificial intelligence (AI) offers a way to minimize these aspects and transform the process as a whole.

The conventional approach for AI-facilitated drug discovery relies heavily on understanding the chemical structures of molecules, often overlooking the vast, unstructured textual knowledge available, which could offer insights into new drug design objectives, adaptability to text-based instructions, and predictions of complex biological activities. Indeed, Liu et al. (2023) breathe life into this notion with their introduction of MoleculeSTM, a multi-modal molecule structure-text modal that leverages both molecular structural information and textual descriptions through a contrastive learning strategy.

Structural and Textual Representation of Molecules

MoleculeSTM represents a multi-modal foundation model that concurrently learns from molecules' chemical structures and their associated textual descriptions. This dual-branch approach, incorporating a chemical structure branch and a textual description branch, allows for the integration of existing molecular structural models and scientific language models (Liu et al., 2023). The contrastive learning paradigm bridges these two branches, facilitating a comprehensive understanding of molecular data.

Criticality of Pretraining

To enable training for the model given the two distinct branches described, the authors have synthesized this data into a structure-text dataset; in which each chemical structure is associated with a textual description. The open-ended format of the textual descriptions allow the model to achieve better zero-shot performance i.e. generalize to unseen data (Liu et al., 2023). Using the PubChemSTM dataset, the authors pretrain the model via contrastive learning. The general framework is to map representations from the structural and textual branches to a

shared molecular model by reducing the distance between pairs of the same molecule and decreasing that between pairs for different molecules.

Scope of Reproducability

A Note On This Notebook

Given the complexity of this project, it is unfortunately a very prohibitive process to migrate the code, as well as the environmental dependencies, into a Google Colab notebook. To the extent that this draft is intended to reflect progress made on this project, we will save time attempting to make all of this code fit into that format in favor on making real progress towards the end goal of reproducing results.

Hypotheses

Here we list out the hypotheses from the original paper that we will be testing for our reproduction.

Zero-shot structure-text retrieval

MoleculeSTM can accurately link molecular structures to their textual descriptions without being directly trained on these specific pairs, showcasing its ability to generalize from its pretraining.

Zero-shot text-based molecule editing

MoleculeSTM can edit and generate molecular structures to meet new specifications provided via textual descriptions, demonstrating its understanding of complex chemical properties and functionalities from text alone.

Molecular property prediction

MoleculeSTM can predict molecular properties accurately, leveraging its pretraining on structure-text relationships, suggesting that it captures meaningful chemical information that is relevant for property prediction tasks.

Setup

In this section, we do our best to provide updated setup instructions for this project adapted from the Shangchao Liu's recommendations.

OS Requirements

In my own testing, the only operating system that currently works for MoleculeSTM is only workable on a Linux system. Do not even try to do this on Windows. MacOS will only work if you have a plan for using some alternate strategy for pretraining, as most Macs lack sufficient GPUs to perform this.

Dependencies

Next, you'll want to ensure that you can install all of the following dependencies.

```
conda create -n MoleculeSTM python=3.7
conda activate MoleculeSTM

conda install -y -c rdkit rdkit=2020.09.1.0
conda install -y -c conda-forge -c pytorch pytorch=1.9.1
conda install -y -c pyg -c conda-forge pyg==2.0.3

pip install requests
pip install tqdm
pip install matplotlib
pip install spacy
pip install Levenshtein

# for SciBert
conda install -y boto3
pip install transformers

# for MoleculeNet
pip install ogb==1.2.0

# install pysmilesutils
python -m pip install
git+https://github.com/MolecularAI/pysmilesutils.git

pip install deepspeed

# install metagron
# pip install megatron-lm==1.1.5
git clone https://github.com/MolecularAI/MolBART.git --branch
megatron-molbart-with-zinc
cd MolBART/megatron_molbart/Megatron-LM-v1.1.5-3D_parallelism
pip install .
cd ../../..

# install apex
# wget https://github.com/NVIDIA/apex/archive/refs/tags/22.03.zip
# unzip 22.03.zip
git clone https://github.com/chao1224/apex.git
cd apex
pip install -v --disable-pip-version-check --no-cache-dir --global-
option="--cpp_ext" --global-option="--cuda_ext" ./
cd ..
```

Methodology

In this section, we provide an overview of the methodology of this project, including a description of the data with included code for acquisition and preprocessing, as well as the model's architecture and pretraining approach.

Data

Due to the massive size of the data, as well as the need to keep the various datasets aligned with each other for the purposes of pretraining, we are unable to push the datasets with the limitatioms of pushes to a GitHub repository, nor can we simply upload a sample of the data. However, we will provide the link to download the data from HuggingFace here, similarly to how we accessed the data for ourselves. In addition, we provide this script adapted from the original code to download all the required training and downstream datasets that will be used for evaluation:

```
from huggingface_hub import HfApi, snapshot_download
api = HfApi()
snapshot_download(repo_id="chao1224/MoleculeSTM", repo_type="dataset",
local_dir='.')
```

In addition to this, there is a required step to perform preprocessing on the PubChemSTM dataset; this is due to the fact that the PubChemSTM dataset only has the license to share the structural representations of molecules, but not the textual represetntation. So, in order to complete the dataset required for pretraining the model, you must follow the instructions below adapted from the advice of Shangchao Liu:

Step 1: Description Extraction

```
import requests
from tqdm import tqdm
from collections import defaultdict
import json


def clean_up_description(description):
    description = description + " "

    ##### extra adj Pure #####
    if description.startswith("Pure "):
        description = description.replace("Pure ", "")
    ##### fix typo #####
    if description.startswith("Mercurycombines"):
        description = description.replace("Mercurycombines", "Mercury
combines")

    name_special_case_list = [
        '17-Hydroxy-6-methylpregna-3,6-diene-3,20-dione. ',
        '5-Thymidylic acid. ',
```

```
        "5'-S-(3-Amino-3-carboxypropyl)-5'-thioadenosine. ",
        "Guanosine 5'-(trihydrogen diphosphate), monoanhydride with
phosphorothioic acid. ",
        "5'-Uridylic acid. ",
        "5'-Adenylic acid, ",
        "Uridine 5'-(tetrahydrogen triphosphate). ",
        "Inosine 5'-Monophosphate. ",
        "Pivaloyloxymethyl butyrate (AN-9), ",
        "4-Amino-5-cyano-7-(D-ribofuranosyl)-7H- pyrrolo(2,3-
d)pyrimidine. ",
        "Cardamonin (also known as Dihydroxymethoxychalcone), ",
    ]

    ##### a special case #####
    description = description.replace("17-Hydroxy-6-methylpregna-3,6-
diene-3,20-dione. ", "17-Hydroxy-6-methylpregna-3,6-diene-3,20-dione
is ")

    ##### a special case #####
    description = description.replace("5-Thymidylic acid. ", "5-
Thymidylic acid. is ")

    ##### a special case #####
    description = description.replace("5'-S-(3-Amino-3-carboxypropyl)-
5'-thioadenosine. ", "5'-S-(3-Amino-3-carboxypropyl)-5'-thioadenosine.
is ")

    ##### a special case #####
    description = description.replace("Guanosine 5'-(trihydrogen
diphosphate), monoanhydride with phosphorothioic acid. ", "Guanosine
5'-(trihydrogen diphosphate), monoanhydride with phosphorothioic acid
is ")

    ##### a special case #####
    description = description.replace("5'-Uridylic acid. ", "5'-
Uridylic acid is ")

    ##### a special case #####
    description = description.replace("5'-Adenylic acid, ", "5'-
Adenylic acid is ")

    ##### a special case #####
    description = description.replace("Uridine 5'-(tetrahydrogen
triphosphate). ", "Uridine 5'-(tetrahydrogen triphosphate). is ")

    ##### a special case #####
    description = description.replace("Inosine 5'-Monophosphate. ",
"Inosine 5'-Monophosphate. is ")

    ##### a special case #####
```

```
    description = description.replace("Pivaloyloxymethyl butyrate (AN-9), ", "Pivaloyloxymethyl butyrate (AN-9) is ")

    ##### a special case #####
    description = description.replace("4-Amino-5-cyano-7-(D-ribofuranosyl)-7H- pyrrolo(2,3-d)pyrimidine. ", "4-Amino-5-cyano-7-(D-ribofuranosyl)-7H- pyrrolo(2,3-d)pyrimidine is ")

    ##### a special case #####
    description = description.replace("Cardamonin (also known as Dihydroxymethoxychalcone), ", "Cardamonin (also known as Dihydroxymethoxychalcone) is ")

    ##### a special case #####
    description = description.replace("Lithium has been used to treat ", "Lithium is ")

    ##### a special case #####
    description = description.replace("4,4'-Methylenebis ", "4,4'-Methylenebis is ")

    ##### a special case #####
    description = description.replace("2,3,7,8-Tetrachlorodibenzo-p-dioxin", "2,3,7,8-Tetrachlorodibenzo-p-dioxin is ")

    ##### a special case #####
    description = description.replace("Exposure to 2,4,5-trichlorophenol ", "2,4,5-Trichlorophenol exposure ")

    index = 0
    L = len(description)
    if description.startswith('C.I. '):
        start_index = len('C.I. ')
    elif description.startswith('Nectriapyrone. D '):
        start_index = len('Nectriapyrone. D ')
    elif description.startswith('Salmonella enterica sv. Minnesota LPS core oligosaccharide'):
        start_index = len('Salmonella enterica sv. Minnesota LPS core oligosaccharide')
    else:
        start_index = 0
    for index in range(start_index, L - 1):
        if index < L-2:
            if description[index] == '.' and description[index+1] == ' ' and 'A' <= description[index+2] <= 'Z':
                break
        elif index == L - 2:
            break

    first_sentence = description[:index+1]
```

```
    return first_sentence


def extract_name(name_raw, description):
    first_sentence = clean_up_description(description)

    splitter = '  --  --  '
    if ' are ' in first_sentence or ' were ' in first_sentence:
        replaced_words = 'These molecules'
    else:
        replaced_words = 'This molecule'

    first_sentence = first_sentence.replace(' is ', splitter)
    first_sentence = first_sentence.replace(' are ', splitter)
    first_sentence = first_sentence.replace(' was ', splitter)
    first_sentence = first_sentence.replace(' were ', splitter)
    first_sentence = first_sentence.replace(' appears ', splitter)
    first_sentence = first_sentence.replace(' occurs ', splitter)
    first_sentence = first_sentence.replace(' stands for ', splitter)
    first_sentence = first_sentence.replace(' belongs to ', splitter)
    first_sentence = first_sentence.replace(' exists ', splitter) #
only for CID=11443
    first_sentence = first_sentence.replace(' has been used in trials
', splitter)
    first_sentence = first_sentence.replace(' has been investigated ',
splitter)
    first_sentence = first_sentence.replace(' has many uses ',
splitter)

    if splitter in first_sentence:
        extracted_name = first_sentence.split(splitter, 1)[0]
    elif first_sentence.startswith(name_raw):
        extracted_name = name_raw
    elif name_raw in first_sentence:
        extracted_name = name_raw
        extracted_name = None
        print("=====", name_raw)
        print("first sentence: ", first_sentence)
        # print()
    else:
        extracted_name = None

    if extracted_name is not None:
        extracted_description = description.replace(extracted_name,
replaced_words)
    else:
        extracted_description = description

    return extracted_name, extracted_description, first_sentence
```

```
if __name__ == "__main__":
    total_page_num = 290
    # Please put your own dataset path here
    datasets_home_folder = "../../../Datasets"

    PubChemSTM_datasets_description_home_folder =
"{}/step_01_PubChemSTM_description".format(datasets_home_folder)
    valid_CID_list = set()
    CID2name_raw, CID2name_extracted = defaultdict(list),
defaultdict(list)
    CID2text_raw, CID2text_extracted = defaultdict(list),
defaultdict(list)

    for page_index in tqdm(range(total_page_num)):
        page_num = page_index + 1
        compound_description_file_name =
"Compound_description_{}.txt".format(page_num)
        f_out =
open("{}/{}".format(PubChemSTM_datasets_description_home_folder,
compound_description_file_name), "w")

        description_url =
"https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/annotations/heading/
json?
heading_type=Compound&heading=Record+Description&page={}".format(page_
num)
        description_data = requests.get(description_url).json()

        description_data = description_data["Annotations"]
        assert description_data["Page"] == page_num
        assert description_data["TotalPages"] == total_page_num

        record_list = description_data["Annotation"]

        for record in record_list:
            try:
                CID = record["LinkedRecords"]["CID"][0]
                if "Name" in record:
                    name_raw = record["Name"]
                    CID2name_raw[CID].append(name_raw)
                else:
                    name_raw = None

                data_list = record["Data"]
                for data in data_list:
                    description = data["Value"]["StringWithMarkup"][0]
["String"].strip()

                    extracted_name, extracted_description,
```

```
first_sentence = extract_name(name_raw, description)
                    if extracted_name is not None:
                        CID2name_extracted[CID].append(extracted_name)

                    CID_special_case_list = [45266824, 11683, 3759,
9700, 439155, 135398675, 135563708, 6030, 10238, 6133, 135398640,
77918, 60748, 11824, 641785, 11125, 7543, 15625, 7271]

                    ##### only for debugging #####
                    if CID in CID_special_case_list:
                        print("page: {}\tCID: {}".format(page_index,
CID))
                        if "Name" in record:
                            print('yes-name')
                            name = record["Name"]
                            print('name:', name)
                        else:
                            print('no-name')
                        print('extracted name:', extracted_name)
                        print("first_sentence:", first_sentence)
                        print("extracted_description:",
extracted_description)
                        print("description:", description)
                        print()

                    CID2text_raw[CID].append(description)

CID2text_extracted[CID].append(extracted_description)

                    valid_CID_list.add(CID)
                    f_out.write("{}\n".format(CID))
                    f_out.write("{}\n\
n".format(extracted_description))
            except:
                # print("===\n", record)
                # print("missing page: {}\tSourceName: {}\tSourceID:
{}".format(page_index, record['SourceName'], record['SourceID']))
                continue

    valid_CID_list = list(set(valid_CID_list))
    valid_CID_list = sorted(valid_CID_list)
    # print("valid CID list: {}".format(valid_CID_list))
    print("Total CID (with raw name) {}".format(len(CID2name_raw)))
    print("Total CID (with extracted name)
{}".format(len(CID2name_extracted)))
    print("Total CID {}".format(len(valid_CID_list)))

    with
open("{}/PubChemSTM_data/raw/CID2name_raw.json".format(datasets_home_f
older), "w") as f:
```

```
        json.dump(CID2name_raw, f)

    with
open("{}/PubChemSTM_data/raw/CID2name.json".format(datasets_home_folde
r), "w") as f:
        json.dump(CID2name_extracted, f)

    with
open("{}/PubChemSTM_data/raw/CID2text_raw.json".format(datasets_home_f
older), "w") as f:
        json.dump(CID2text_raw, f)

    with
open("{}/PubChemSTM_data/raw/CID2text.json".format(datasets_home_folde
r), "w") as f:
        json.dump(CID2text_extracted, f)
```

Step 2: Download SDF

```
import argparse
from PubChem_utils import download_and_extract_compound_file


parser = argparse.ArgumentParser()
parser.add_argument("--block_id", type=int, default=0)
args = parser.parse_args()


if __name__ == "__main__":
    datasets_home_folder = "../../../Datasets"

    PubChemSTM_datasets_home_folder =
"{}/step_02_PubChemSTM_SDF".format(datasets_home_folder)
    block_id = args.block_id
    block_size = 500000
    start_id = block_id * block_size + 1
    end_id = (block_id + 1) * block_size

    compound_file_name =
"Compound_{:09d}_{:09d}.sdf.gz".format(start_id, end_id)

download_and_extract_compound_file(PubChemSTM_datasets_home_folder,
compound_file_name)
```

Step 3: Filter Out SDF

```
from tqdm import tqdm
import json
import gzip
import numpy as np
```

```
from rdkit import Chem
from rdkit import RDLogger
RDLogger.DisableLog('rdApp.*')
import multiprocessing
from multiprocessing import Pool
import sys


if __name__ == "__main__":
    datasets_home_folder = "../../../Datasets"

    PubChemSTM_datasets_description_home_folder =
"{}/step_01_PubChemSTM_description".format(datasets_home_folder)
    with
open("{}/PubChemSTM_data/raw/CID2text.json".format(datasets_home_folde
r), "r") as f:
        CID2text = json.load(f)
    target_CID_list = set(CID2text.keys())

    PubChemSTM_datasets_input_folder =
"{}/step_02_PubChemSTM_SDF".format(datasets_home_folder)
    PubChemSTM_datasets_output_folder =
"{}/step_03_PubChemSTM_filtered".format(datasets_home_folder)
    block_size = 500000

    def extract_one_SDF_file(block_id):
        valid_mol_count = 0

        writer =
Chem.SDWriter('{}/filtered_{}.sdf'.format(PubChemSTM_datasets_output_f
older, block_id))
        start_id = block_id * block_size + 1
        end_id = (block_id + 1) * block_size

        compound_file_name =
"Compound_{:09d}_{:09d}.sdf.gz".format(start_id, end_id)
        gzip_loader =
gzip.open("{}/{}".format(PubChemSTM_datasets_input_folder,
compound_file_name))
        suppl = Chem.ForwardSDMolSupplier(gzip_loader)

        for mol in tqdm(suppl):
            if mol is None:
                continue
            cid = mol.GetProp("PUBCHEM_COMPOUND_CID")

            if cid not in target_CID_list:
                continue
```

```
            writer.write(mol)
            valid_mol_count += 1

        print("block id: {}\nfound {}\n\n".format(block_id,
valid_mol_count))
        sys.stdout.flush()
        return

    num_process = multiprocessing.cpu_count()
    print("{} CPUs".format(num_process))
    num_process = 8
    p = Pool(num_process)

    total_block_num = 325
    block_id_list = np.arange(total_block_num+1)
    with p:
        p.map(extract_one_SDF_file, block_id_list)
```

Step 4: Merge SDF

```
from tqdm import tqdm
import json

from rdkit import Chem
from rdkit import RDLogger
RDLogger.DisableLog('rdApp.*')


if __name__ == "__main__":
    datasets_home_folder = "../../../Datasets"

    PubChemSTM_datasets_description_home_folder =
"{}/step_01_PubChemSTM_description".format(datasets_home_folder)
    with
open("{}/PubChemSTM_data/raw/CID2text.json".format(datasets_home_folde
r), "r") as f:
        CID2text = json.load(f)
    target_CID_list = set(CID2text.keys())
    print('The length of target_CID_list:
{}'.format(len(target_CID_list)))

    PubChemSTM_datasets_folder =
"{}/step_03_PubChemSTM_filtered".format(datasets_home_folder)
    writer =
Chem.SDWriter('{}/PubChemSTM_data/raw/molecules.sdf'.format(datasets_h
ome_folder))

    total_block_num = 325
    found_CID_set = set()
    for block_id in range(total_block_num+1):
```

```
        compound_file_path =
"{}/filtered_{}.sdf".format(PubChemSTM_datasets_folder, block_id)
        try:
            suppl = Chem.SDMolSupplier(compound_file_path)

            for mol in tqdm(suppl):
                writer.write(mol)
                cid = mol.GetProp("PUBCHEM_COMPOUND_CID")
                found_CID_set.add(cid)
        except:
            print("block id: {} with 0 valid SDF
file".format(block_id))
            continue

    for CID in target_CID_list:
        if CID not in found_CID_set:
            print("CID: {} not found.".format(CID))

    print("In total: {} molecules".format(len(found_CID_set)))
```

Step 5: Sample Extraction

```
from tqdm import tqdm
import json

from rdkit import Chem
from rdkit import RDLogger
RDLogger.DisableLog('rdApp.*')


if __name__ == "__main__":
    datasets_home_folder = "../../../Datasets"

    PubChemSTM_datasets_description_home_folder =
"{}/step_01_PubChemSTM_description".format(datasets_home_folder)
    with
open("{}/PubChemSTM_data/raw/CID2text.json".format(datasets_home_folde
r), "r") as f:
        CID2text = json.load(f)
    target_CID_list = set(CID2text.keys())
    print('The length of target_CID_list:
{}'.format(len(target_CID_list)))

    PubChemSTM_datasets_folder =
"{}/step_03_PubChemSTM_filtered".format(datasets_home_folder)
    writer =
Chem.SDWriter('{}/PubChemSTM_data/raw/molecules.sdf'.format(datasets_h
ome_folder))

    total_block_num = 325
```

```
    found_CID_set = set()
    for block_id in range(total_block_num+1):
        compound_file_path =
"{}/filtered_{}.sdf".format(PubChemSTM_datasets_folder, block_id)
        try:
            suppl = Chem.SDMolSupplier(compound_file_path)

            for mol in tqdm(suppl):
                writer.write(mol)
                cid = mol.GetProp("PUBCHEM_COMPOUND_CID")
                found_CID_set.add(cid)
        except:
            print("block id: {} with 0 valid SDF
file".format(block_id))
            continue

    for CID in target_CID_list:
        if CID not in found_CID_set:
            print("CID: {} not found.".format(CID))

    print("In total: {} molecules".format(len(found_CID_set)))
```

Model

In this subsection, we include the code from the original paper for the architecure and pretraining of the MoleculeSTM model.

Architecture

We begin by including the architecture for MoleculeSTM, which uses a contrastive pretraining approach and a number of linear layers corresponding to the `[input_dim] + hidden_dim`.

```
from torch import nn
from torch.nn import functional as F
from collections.abc import Sequence


class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims, batch_norm=False,
activation="relu", dropout=0):
        super(MLP, self).__init__()

        if not isinstance(hidden_dims, Sequence):
            hidden_dims = [hidden_dims]
        self.dims = [input_dim] + hidden_dims

        if isinstance(activation, str):
            self.activation = getattr(F, activation)
        else:
```

```
            self.activation = activation
        if dropout:
            self.dropout = nn.Dropout(dropout)
        else:
            self.dropout = None

        self.layers = nn.ModuleList()
        for i in range(len(self.dims) - 1):
            self.layers.append(nn.Linear(self.dims[i], self.dims[i +
1]))
        if batch_norm:
            self.batch_norms = nn.ModuleList()
            for i in range(len(self.dims) - 2):
                self.batch_norms.append(nn.BatchNorm1d(self.dims[i +
1]))
        else:
            self.batch_norms = None

    def forward(self, input):
        layer_input = input

        for i, layer in enumerate(self.layers):
            hidden = layer(layer_input)
            if i < len(self.layers) - 1:
                if self.batch_norms:
                    x = hidden.flatten(0, -2)
                    hidden = self.batch_norms[i](x).view_as(hidden)
                hidden = self.activation(hidden)
                if self.dropout:
                    hidden = self.dropout(hidden)
            if hidden.shape == layer_input.shape:
                hidden = hidden + layer_input
            layer_input = hidden

        return hidden
```

Pretraining

Here we provide the script used for pretraining the MoleculeSTM model. This script synthesizes the datasets after preprocessing and utilizes a contrastive learning approach. The general framework is to map representations from the structural and textual branches to a shared molecular model by reducing the distance between pairs of the same molecule and decreasing that between pairs for different molecules.

```
import os
import time
import numpy as np
from tqdm import tqdm
import argparse
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader as torch_DataLoader

from torch_geometric.loader import DataLoader as pyg_DataLoader
from transformers import AutoModel, AutoTokenizer

from MoleculeSTM.datasets import (
    PubChemSTM_Datasets_SMILES, PubChemSTM_SubDatasets_SMILES,
    PubChemSTM_Datasets_Graph, PubChemSTM_SubDatasets_Graph,
    PubChemSTM_Datasets_Raw_SMILES, PubChemSTM_SubDatasets_Raw_SMILES,
    PubChemSTM_Datasets_Raw_Graph, PubChemSTM_SubDatasets_Raw_Graph
)
from MoleculeSTM.models import GNN, GNN_graphpred
from MoleculeSTM.utils import prepare_text_tokens,
get_molecule_repr_MoleculeSTM, freeze_network
from MoleculeSTM.models.mega_molbart.mega_mol_bart import MegaMolBART


def cycle_index(num, shift):
    arr = torch.arange(num) + shift
    arr[-shift:] = torch.arange(shift)
    return arr


def do_CL(X, Y, args):
    if args.normalize:
        X = F.normalize(X, dim=-1)
        Y = F.normalize(Y, dim=-1)

    if args.SSL_loss == 'EBM_NCE':
        criterion = nn.BCEWithLogitsLoss()
        neg_Y = torch.cat([Y[cycle_index(len(Y), i + 1)] for i in
range(args.CL_neg_samples)], dim=0)
        neg_X = X.repeat((args.CL_neg_samples, 1))

        pred_pos = torch.sum(X * Y, dim=1) / args.T
        pred_neg = torch.sum(neg_X * neg_Y, dim=1) / args.T

        loss_pos = criterion(pred_pos,
torch.ones(len(pred_pos)).to(pred_pos.device))
        loss_neg = criterion(pred_neg,
torch.zeros(len(pred_neg)).to(pred_neg.device))
        CL_loss = (loss_pos + args.CL_neg_samples * loss_neg) / (1 +
args.CL_neg_samples)

        CL_acc = (torch.sum(pred_pos > 0).float() + torch.sum(pred_neg
```

```
< 0).float()) / \
                  (len(pred_pos) + len(pred_neg))
        CL_acc = CL_acc.detach().cpu().item()

    elif args.SSL_loss == 'InfoNCE':
        criterion = nn.CrossEntropyLoss()
        B = X.size()[0]
        logits = torch.mm(X, Y.transpose(1, 0))  # B*B
        logits = torch.div(logits, args.T)
        labels = torch.arange(B).long().to(logits.device)  # B*1

        CL_loss = criterion(logits, labels)
        pred = logits.argmax(dim=1, keepdim=False)
        CL_acc = pred.eq(labels).sum().detach().cpu().item() * 1. / B

    else:
        raise Exception

    return CL_loss, CL_acc


def save_model(save_best, epoch=None):
    if args.output_model_dir is not None:
        if save_best:
            global optimal_loss
            print("save model with loss: {:.5f}".format(optimal_loss))
            model_file = "model.pth"

        elif epoch is None:
            model_file = "model_final.pth"

        else:
            model_file = "model_{}.pth".format(epoch)

        saved_file_path = os.path.join(args.output_model_dir,
"text_{}".format(model_file))
        torch.save(text_model.state_dict(), saved_file_path)

        saved_file_path = os.path.join(args.output_model_dir,
"molecule_{}".format(model_file))
        torch.save(molecule_model.state_dict(), saved_file_path)

        saved_file_path = os.path.join(args.output_model_dir,
"text2latent_{}".format(model_file))
        torch.save(text2latent.state_dict(), saved_file_path)

        saved_file_path = os.path.join(args.output_model_dir,
"mol2latent_{}".format(model_file))
        torch.save(mol2latent.state_dict(), saved_file_path)
    return
```

```
def train(
    epoch,
    dataloader,
    text_model, text_tokenizer,
    molecule_model, MegaMolBART_wrapper=None):

    if args.representation_frozen:
        text_model.eval()
        molecule_model.eval()
    else:
        text_model.train()
        molecule_model.train()
    text2latent.train()
    mol2latent.train()

    if args.verbose:
        L = tqdm(dataloader)
    else:
        L = dataloader

    start_time = time.time()
    accum_loss, accum_acc = 0, 0
    for step, batch in enumerate(L):
        description = batch[0]
        molecule_data = batch[1]

        description_tokens_ids, description_masks =
prepare_text_tokens(
            device=device, description=description,
tokenizer=text_tokenizer, max_seq_len=args.max_seq_len)
        description_output =
text_model(input_ids=description_tokens_ids,
attention_mask=description_masks)
        description_repr = description_output["pooler_output"]
        description_repr = text2latent(description_repr)

        if molecule_type == "SMILES":
            molecule_data = list(molecule_data) # for SMILES_list
            molecule_repr = get_molecule_repr_MoleculeSTM(
                molecule_data, mol2latent=mol2latent,
                molecule_type=molecule_type,
MegaMolBART_wrapper=MegaMolBART_wrapper)
        else:
            molecule_data = molecule_data.to(device)
            molecule_repr = get_molecule_repr_MoleculeSTM(
                molecule_data, mol2latent=mol2latent,
                molecule_type=molecule_type,
molecule_model=molecule_model)
```

```
        loss_01, acc_01 = do_CL(description_repr, molecule_repr, args)
        loss_02, acc_02 = do_CL(molecule_repr, description_repr, args)
        loss = (loss_01 + loss_02) / 2
        acc = (acc_01 + acc_02) / 2
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        accum_loss += loss.item()
        accum_acc += acc

    accum_loss /= len(L)
    accum_acc /= len(L)

    global optimal_loss
    temp_loss = accum_loss
    if temp_loss < optimal_loss:
        optimal_loss = temp_loss
        save_model(save_best=True, epoch=epoch)
    print("CL Loss: {:.5f}\tCL Acc: {:.5f}\tTime:
{:.5f}".format(accum_loss, accum_acc, time.time() - start_time))
    return


if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument("--seed", type=int, default=42)
    parser.add_argument("--device", type=int, default=0)

    parser.add_argument("--dataspace_path", type=str,
default="../data")
    parser.add_argument("--dataset", type=str, default="PubChemSTM")
    parser.add_argument("--text_type", type=str, default="SciBERT",
choices=["SciBERT"])
    parser.add_argument("--molecule_type", type=str, default="SMILES",
choices=["SMILES", "Graph"])
    parser.add_argument("--representation_frozen",
dest='representation_frozen', action='store_true')
    parser.add_argument('--no_representation_frozen',
dest='representation_frozen', action='store_false')
    parser.set_defaults(representation_frozen=False)

    parser.add_argument("--batch_size", type=int, default=32)
    parser.add_argument("--text_lr", type=float, default=1e-4)
    parser.add_argument("--mol_lr", type=float, default=1e-5)
    parser.add_argument("--text_lr_scale", type=float, default=1)
    parser.add_argument("--mol_lr_scale", type=float, default=1)
    parser.add_argument("--num_workers", type=int, default=8)
```

```
    parser.add_argument("--epochs", type=int, default=100)
    parser.add_argument("--decay", type=float, default=0)
    parser.add_argument('--verbose', dest='verbose',
action='store_true')
    parser.set_defaults(verbose=False)
    parser.add_argument("--output_model_dir", type=str, default=None)

    ########## for SciBERT ##########
    parser.add_argument("--max_seq_len", type=int, default=512)

    ########## for MegaMolBART ##########
    parser.add_argument("--megamolbart_input_dir", type=str,
default="../data/pretrained_MegaMolBART/checkpoints")
    parser.add_argument("--vocab_path", type=str,
default="../MoleculeSTM/bart_vocab.txt")

    ########## for 2D GNN ##########
    parser.add_argument("--pretrain_gnn_mode", type=str,
default="GraphMVP_G", choices=["GraphMVP_G"])
    parser.add_argument("--gnn_emb_dim", type=int, default=300)
    parser.add_argument("--num_layer", type=int, default=5)
    parser.add_argument('--JK', type=str, default='last')
    parser.add_argument("--dropout_ratio", type=float, default=0.5)
    parser.add_argument("--gnn_type", type=str, default="gin")
    parser.add_argument('--graph_pooling', type=str, default='mean')

    ########## for contrastive SSL ##########
    parser.add_argument("--SSL_loss", type=str, default="EBM_NCE",
choices=["EBM_NCE", "InfoNCE"])
    parser.add_argument("--SSL_emb_dim", type=int, default=256)
    parser.add_argument("--CL_neg_samples", type=int, default=1)
    parser.add_argument("--T", type=float, default=0.1)
    parser.add_argument('--normalize', dest='normalize',
action='store_true')
    parser.add_argument('--no_normalize', dest='normalize',
action='store_false')
    parser.set_defaults(normalize=True)

    args = parser.parse_args()
    print("arguments\t", args)

    torch.manual_seed(args.seed)
    np.random.seed(args.seed)
    device = torch.device("cuda:" + str(args.device)) \
        if torch.cuda.is_available() else torch.device("cpu")
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(args.seed)

    if "PubChemSTM" in args.dataset:
        dataset_root = os.path.join(args.dataspace_path,
```

```
"PubChemSTM_data")
    else:
        raise Exception

    kwargs = {}

    # ##### prepare text model #####
    if args.text_type == "SciBERT":
        pretrained_SciBERT_folder = os.path.join(args.dataspace_path,
'pretrained_SciBERT')
        text_tokenizer =
AutoTokenizer.from_pretrained('allenai/scibert_scivocab_uncased',
cache_dir=pretrained_SciBERT_folder)
        text_model =
AutoModel.from_pretrained('allenai/scibert_scivocab_uncased',
cache_dir=pretrained_SciBERT_folder).to(device)
        kwargs["text_tokenizer"] = text_tokenizer
        kwargs["text_model"] = text_model
        text_dim = 768
    else:
        raise Exception

    ##### prepare molecule model #####
    molecule_type = args.molecule_type
    if molecule_type == "SMILES":
        if args.dataset == "PubChemSTM":
            dataset = PubChemSTM_Datasets_SMILES(dataset_root)
        elif args.dataset == "PubChemSTM1K":
            # only for testing
            dataset = PubChemSTM_SubDatasets_SMILES(dataset_root,
size=1000)
        elif args.dataset == "PubChemSTM_Raw":
            dataset = PubChemSTM_Datasets_Raw_SMILES(dataset_root)
        elif args.dataset == "PubChemSTM10K_Raw":
            # only for testing
            dataset = PubChemSTM_SubDatasets_Raw_SMILES(dataset_root,
size=10000)
        else:
            raise Exception
        dataloader_class = torch_DataLoader

        if args.output_model_dir is not None:
            MegaMolBART_dir = os.path.join(args.output_model_dir,
"MegaMolBART")
        else:
            MegaMolBART_dir = None
        MegaMolBART_wrapper = MegaMolBART(
            vocab_path=args.vocab_path,
            input_dir=args.megamolbart_input_dir,
            output_dir=MegaMolBART_dir)
```

```
        molecule_model = MegaMolBART_wrapper.model
        kwargs["MegaMolBART_wrapper"] = MegaMolBART_wrapper
        kwargs["molecule_model"] = molecule_model
        molecule_dim = 256

    elif molecule_type == "Graph":
        if args.dataset == "PubChemSTM":
            dataset = PubChemSTM_Datasets_Graph(dataset_root)
        elif args.dataset == "PubChemSTM1K":
            dataset = PubChemSTM_SubDatasets_Graph(dataset_root,
size=1000)
        elif args.dataset == "PubChemSTM10K":
            dataset = PubChemSTM_SubDatasets_Graph(dataset_root,
size=10000)
        elif args.dataset == "PubChemSTM_Raw":
            dataset = PubChemSTM_Datasets_Raw_Graph(dataset_root)
        elif args.dataset == "PubChemSTM1K_Raw":
            dataset = PubChemSTM_SubDatasets_Raw_Graph(dataset_root,
size=1000)
        elif args.dataset == "PubChemSTM10K_Raw":
            dataset = PubChemSTM_SubDatasets_Raw_Graph(dataset_root,
size=10000)
        dataloader_class = pyg_DataLoader
        molecule_node_model = GNN(
            num_layer=args.num_layer, emb_dim=args.gnn_emb_dim,
            JK=args.JK, drop_ratio=args.dropout_ratio,
            gnn_type=args.gnn_type)
        molecule_model = GNN_graphpred(
            num_layer=args.num_layer, emb_dim=args.gnn_emb_dim,
JK=args.JK, graph_pooling=args.graph_pooling,
            num_tasks=1, molecule_node_model=molecule_node_model)
        pretrained_model_path = os.path.join(args.dataspace_path,
"pretrained_GraphMVP", args.pretrain_gnn_mode, "model.pth")
        molecule_model.from_pretrained(pretrained_model_path)

        molecule_model = molecule_model.to(device)

        kwargs["molecule_model"] = molecule_model
        molecule_dim = args.gnn_emb_dim

    else:
        raise Exception
    dataloader = dataloader_class(dataset, batch_size=args.batch_size,
shuffle=True, num_workers=args.num_workers)

    text2latent = nn.Linear(text_dim, args.SSL_emb_dim).to(device)
    mol2latent = nn.Linear(molecule_dim, args.SSL_emb_dim).to(device)

    if args.representation_frozen:
        print("Representation is fronzen during pretraining.")
```

```
        freeze_network(text_model)
        freeze_network(molecule_model)
        model_param_group = [
            {"params": text2latent.parameters(), "lr": args.text_lr *
args.text_lr_scale},
            {"params": mol2latent.parameters(), "lr": args.mol_lr *
args.mol_lr_scale},
        ]
    else:
        model_param_group = [
            {"params": text_model.parameters(), "lr": args.text_lr},
            {"params": molecule_model.parameters(), "lr":
args.mol_lr},
            {"params": text2latent.parameters(), "lr": args.text_lr *
args.text_lr_scale},
            {"params": mol2latent.parameters(), "lr": args.mol_lr *
args.mol_lr_scale},
        ]
    optimizer = optim.Adam(model_param_group, weight_decay=args.decay)
    optimal_loss = 1e10

    for e in range(1, args.epochs+1):
        print("Epoch {}".format(e))
        train(e, dataloader, **kwargs)

    save_model(save_best=False)
```

Results

Our current progress in reproducing this paper has been limited given the extensive issues regarding the initial setup and preprocessing of the data. Configuring the proper environment setup and the preprocessing of the data in sum took ~2 weeks to get through. I'm currently at the point where MoleculeSTM is being pretrained for 100 epochs, as described in the original paper. Unfortunately, although the script has been running for several days, it's still only managed to get through 8/100 epochs in total. I've reached out to Shangchao Lui again for guidance and attempting to understand how they trained the model from a computation perspective, but they have yet to respond to this inquiry.

To that end, I unfortunately do not have much to show in terms of results at this stage. I'm dubious that I will find a way to increase the speed of the pretraining, and at this rate there simply isn't enough time alloted for the remainder of the semester for me to complete the training and have time to submit more comprehensive results. To that point, the next section will detail all of the progress made thus far on this project, including setbacks and future plans to ensure that I have a feasible workload for the final project submission.

Discussion

In this section, in lieu of tangible results given the current state of the pretraining process, we include this discussion section to cover the feasibility of this project given issues we've faced and our future plans to adapt to these challenges.

Reproducability

Given the state of my initial results and the unexpected computational complexity of training, I do not believe that this paper will be reproducible within the timeline of this project. I even reached out to Shangchao Liu, the first author of the original paper, for guidance and while he indicated that he was "pretty sure" the results could be reproduced, he acknowledged that the setup, processing, and training would be a painful process. To that point, even after having successfully completed the initial setup, preprocessing, and beginning the pretraining process with an NVIDA RTX 4090 GPU running for the past 4 days, I'm still only on epoch 8/100; therefore, given these complexities, I do not believe that this project will be reproducible within the scope of this project.

Setbacks and Diffuclties

The most major setback I've faced thus far in reproducing the results of this paper have been the significant amount of time sunk into the intial setup and preprocessing of the data. From a setup standpoint, I realized early on in my testing that many of the packages required for running this model are either deprecated or suffer from version incompatibilities. In total, I made 23 setup attempts with only the last succeeding. A significant amount of time has been invested in refactoring the source code, testing different version combinations for packages, using 3 different operating systems, and reaching out to the author for support. To finally get through the setup process, I had to take everything I had learned from my testing and apply it to a brand new computer I purchased exclusively to run Ubuntu, per the suggestion of the author. The several open issues on the GitHub repo containing the original source code, and the analysis of my peer groups trying to replicate this paper have confirmed my own experience that the setup for this project needs to be improved substantially to enable future researchers to replicate this work.

Another major setback I've experienced has been with regard to computational complexity. I assumed, naively, that my RTX 4090 card that I've used for training other models without issue would be suitable for this project. In reality, even with the top-of-the-line consumer GPU at my disposal, the pretraining process is taking an exhaustive and prohibitive amount of time. Though I've left the pretraining script running for several days at this point, it has only completed 8 of the 100 epochs necessary to complete. The authors make note of checkpoints with a pretrained model that can be used, and I considered doing this to cut down on time, but if I'm only using their pretrained model that seems to defeat the point of this project. Moreover, even if I continue to allow the pretraining script to run, I've estimated that it will take ~30 days to complete at its current pace, meaning even by the end of the semester, I would have close to nothing to show for all my efforts in attempting to reproduce this paper. To that end, as I will expound on the next section, I believe it would be warranted for me to switch my project topic and idenfity one of my backup papers that will be more feasible to reproduce for he purposes of this project.

Future Plans

Given everything discussed in the previous sections regarding the difficulty of setup and the computational limitations that we're currently faced with, the best option seems to be requesting the ability to change our project to reproduce one of our backup papers instead. This is disappointing because on one hand, the research done by Liu et al. (2023) is truly groundbreaking and interesting from a deep learning perspective, but on the other hand, the lack of documentation for updated frameworks and the extensive need for computing has thrown a wrench into every expectation for this project. It would be in our best interest to switch papers as soon as possible and start again from scratch, as the current pace of the pretraining process does not bode well for our final project submission timeline.

References

Liu, S., Nie, W., Wang, C., Lu, J., Qiao, Z., Liu, L., ... & Anandkumar, A. (2023). Multi-modal molecule structure–text model for text-based retrieval and editing. Nature Machine Intelligence, 5(12), 1447-1457.