In [ ]:
```
from IPython.display import Image
from PIL import Image  as pimg
```

ChemicalX

By Edwin Tembo - Harvard Extension School 2024, Modified in Jan - 2025

Code Url : https://github.com/AstraZeneca/chemicalx

Paper Url: https://arxiv.org/pdf/2202.05240

Documentation: https://chemicalx.readthedocs.io/en/latest/

Important Note: This Notebook ran on an single A100 GPU in a Google Colab Environment.

This notebook is a demonstration of a series of experiments for Drug-Drug Interaction, Polypharmacy and Drug Synergy capabilities of the chemicalX library. Developed by AstraZeneca in 2022, this set of deep leaning models was created as an additional tool for Drug Safety Researchers, Computational Chemists and Oncologists to help solve drug combination synergy problems, polypharmacy side effects and drug-drug interactions with less reliance on physical lab tests.

The models in the library work by creating latent respresentations of drug molecule pairs, with preprocessed features and labels as input. In the case of drug combination synegy, a third biological context input is used. Once each input of the Drug-Drug pair or Drug-Drug-Context triple is encoded into it's latent representation, it is passed into a **Head Layer** that outputs a scaler probability of a positive answer to the following questions, depending on the type of research:

• **Polypharmacy side effects.** Is it possible that drugs X and Y together cause polypharmacy side effect Z?

• **Drug-drug interactions.** Can drugs X and Y have interaction Z when administered together?

• **Pair synergy identification.** Are drugs X and Y synergistic at treating disease Z when applied in combination.

A diagram of the encoder /head layer architecture is shown below:

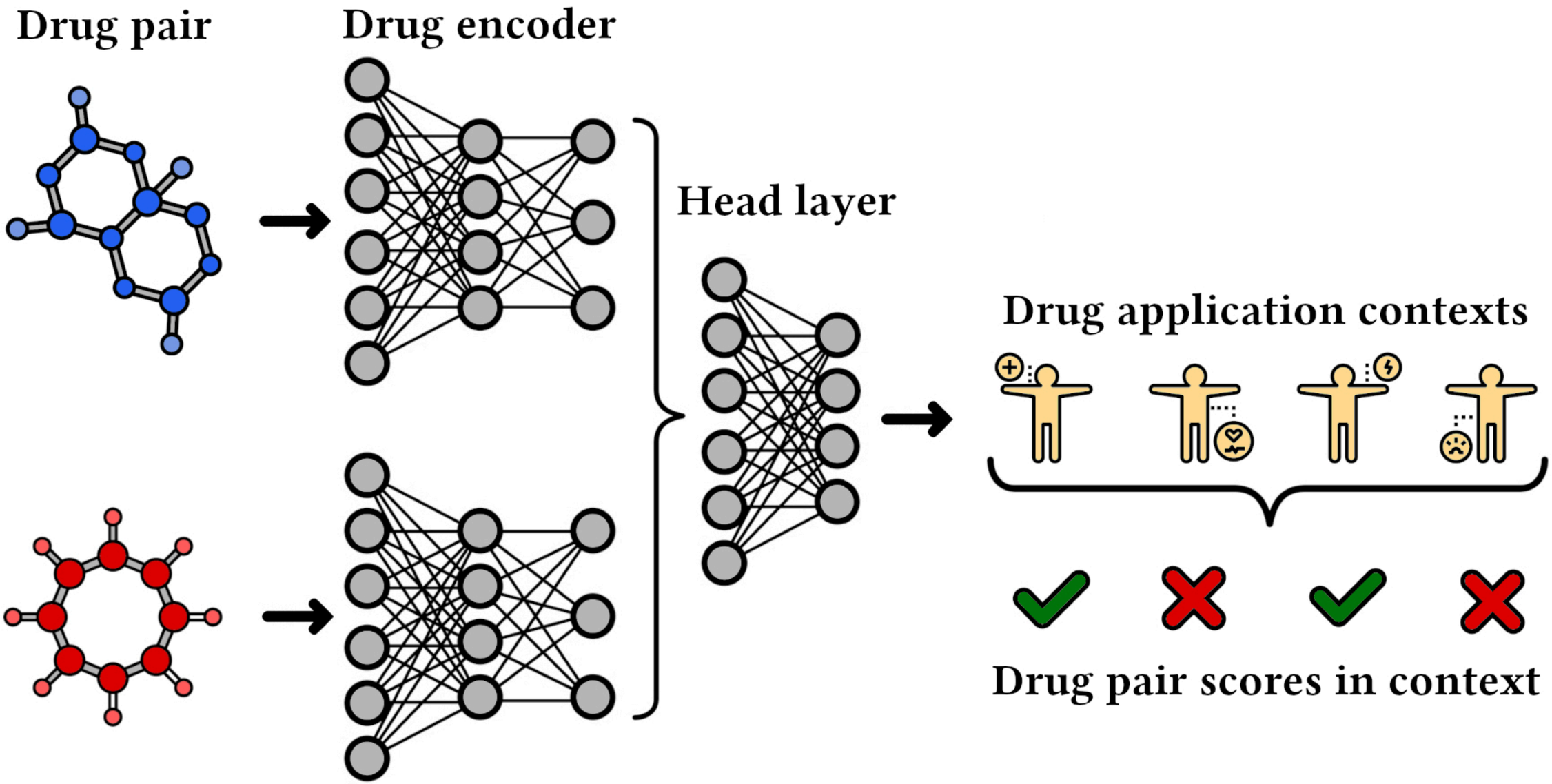


Image From ChemicalX Github Repository

Task Definitions:

Polypharmacy - According to the National Library of Medicine, Polypharmacy occurs when 5 or more medicines are administered, increasing the risk of falls, fraility, disability and mortality. Other definitions consider just 2 drugs administered at the same time as polypharmacy.

Drug-Drug Interactions: "A change in a drug's effect on the body when the drug is taken together with a second drug. A drug-drug interaction can delay, decrease, or enhance absorption of either drug. This can decrease or increase the action of either or both drugs

or cause adverse effects." *(National Institute of Health)*

Synergy - "In medicine, describes the interaction of two or more drugs when their combined effect is greater than the sum of the effects seen when each drug is given alone." *(National Cancer Institute)*

Installation

ChemicalX is based on torchdrug and Pytorch Goemetric. The accompanying libraries, including Torch Scatter will be needed.

Additional libraries will include:

molSimplify - A 3D rendering toolkit for molecules using SMILES as input.

py3Dmol - Required for molSimplify.

rdkit - A toolkit for rendering molecules in 2 and 3 dimensional space.

Installation may take up to 30 minutes.

In [ ]:
```
##Finding Pytorch Version
import torch
import numpy as np
import pandas as pd
import os
print(torch.__version__)
torch_version = f"{torch.__version__}"
```

```
2.5.1+cu121
```

In [ ]:
```
## Install ChemicalX and rdkit to visualize molecules

! pip install torch-scatter -f https://pytorch-geometric.com/whl/torch-{torch_version}.html
! pip install torchdrug
! pip install chemicalx
! rdkit==2022.9.5
!pip install molSimplify
!pip install py3Dmol
!pip install torchmetrics
```

To train using TorchDrug molecules, replace the node_feature attribute calls in chemicalX model files with calls for atom_feature. The node_feature attribute is deprecated in TorchDrug and has been replaced with atom_feature.

In this case SSIDDI and DEEPDRUG are used to predict the relation based on the drug/protein pair

In [ ]:
```
ssiddi_path = "/usr/local/lib/python3.10/dist-packages/chemicalx/models/ssiddi.py"
deepdrug_path = "/usr/local/lib/python3.10/dist-packages/chemicalx/models/deepdrug.py"

file_paths = [ssiddi_path, deepdrug_path]
```

In [ ]:
```
for path in file_paths:
  os.environ['FILE_PATH'] = path
  !sed -i 's/node_feature/atom_feature/g' $FILE_PATH
```

Restart Runtime Here for changes to take effect if running this for the same time in the current session on Colab.

In [ ]:
```
import os
import rdkit
from rdkit import Chem
```

In [ ]:
```
import datetime
import torch
from torch import nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

import numpy as np
import time

from pathlib import Path
from google.colab import userdata

RANDOM_SEED = 1024
torch.manual_seed(RANDOM_SEED)
torch.cuda.manual_seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
```

Loading Data

ChemicalX provides various preprocessed datasets from publicly available data sources. Below is a list of the data sources with respective ChemicalX tasks.

Dataset	ChemicalX Task	Url
TWOSIDES	**Polypharmacy**	https://tatonettilab.org/resources/tatonetti-stm.html
Drugbank	**Drug-Drug Interaction**	https://go.drugbank.com/
DrugComb	**Synergy**	https://drugcomb.fimm.fi/
DrugCombDB	**Synergy**	http://drugcombdb.denglab.org/main
OncoPolyPharm	**Synergy**	

Potential ChemicalX compatibility issue!

Depending on the version of TorchDrug, some users may run into the following error:

AttributeError: type object 'Molecule' has no attribute 'dummy_atom'. To resolve this issue, run the code below:

In [ ]:
```
#Determine your Python location
!which python
!python --version
```

```
/usr/local/bin/python
Python 3.10.12
```

In [ ]:
```
##Create a variable, making sure to point to the constants.py file in the correct dist-packages folder as shown below :
chemicalX_constants_path = "/usr/local/lib/python3.10/dist-packages/chemicalx/constants.py"
os.environ['CX_PATH'] = chemicalX_constants_path
```

In [ ]:
```
## Run this line to change the  TORCH_NODE_FEATURES variable in the the constants.py file
## This code is replacing the variable value for the TORCHDRUG_NODE_FEATURES from len(atom_default(Molecule.dummy_atom)) to 128.
## 128 was chosen based on the features used in the examples, but can be any appropriate integer.

!sed -i 's/TORCHDRUG_NODE_FEATURES = len(atom_default(Molecule.dummy_atom))/TORCHDRUG_NODE_FEATURES = 128/g' $CX_PATH

## Note: If this does not work, open the constants.py file directly and update the TORCHDRUG_NODE_FEATURES variable accordingly.
```

Data Exploration:

Although the data is already preprocessed, users may want to further preprocess or validate it. There are several ways to access the data from the sources using the provided datasetloader module. The data provided here may not be up to date with the newest drugs, but users may add new data as needed.

In [ ]:
```
from torchdrug.data import Graph, PackedGraph, Molecule
from chemicalx.data import  BatchGenerator, DrugFeatureSet, DrugPairBatch
from typing import Dict, Iterable, Mapping, Union
from collections import defaultdict
```

In [ ]:
```
class CustomDrugFeatureSet(DrugFeatureSet):
    """Drug feature set for compounds."""


    @classmethod
    def from_custom_dict(cls, data: Dict[str, Dict], atom_feature: list= ['default']) -> "DrugFeatureSet":
        """Generate a drug feature set from a data dictionary."""
        return cls(
            {
                key: {
                    "features": torch.FloatTensor(features["features"]).view(1, -1),
                    "molecule": Molecule.from_smiles(features["smiles"], atom_feature),
                }
                for key, features in data.items()
            }
        )
```

In [ ]:
```

```

See linst of properties for atom_feature : https://torchdrug.ai/docs/_modules/torchdrug/data/feature.html

"atom_default", "atom_center_identification", "atom_synthon_completion","atom_symbol", "atom_explicit_property_prediction", "atom_property_prediction","atom_position", "atom_pretrain", "atom_residue_symbol"

In [ ]:
```
## We'll use some of these. I'm not a chemist, so some of these may be irrelevant/redundant.

atom_feature_list =  ["default", "center_identification", "synthon_completion", "explicit_property_prediction","position"]
```

In [ ]:
```
## To get the features of any of the mentioned datasets:
from chemicalx.data.datasetloader import RemoteDatasetLoader
from rdkit.Chem import rdFingerprintGenerator , DataStructs , rdMolDescriptors, Draw, MolFromSmiles
from rdkit.Chem.rdMolDescriptors import GetHashedMorganFingerprint
from torchdrug.data import Molecule
from molSimplify.Informatics.jupyter_vis import view_structures
from molSimplify.Classes.mol3D import mol3D
from functools import lru_cache
## current list of data sources:
data_sources = ["drugcombdb", "drugcomb", "twosides", "drugbankddi"]

##  Expose some of the functions in the RemoteDatasetLoader to get more access to the dataset.
class CustomRtDatasetLoader(RemoteDatasetLoader):
    def __init__(self, data_source: str, atom_feature_list: list[str] = atom_feature_list):
      super().__init__(data_source)
      self.raw_data_url          = f"https://raw.githubusercontent.com/AstraZeneca/chemicalx/main/dataset/{data_source}/drug_set.json"
      self.raw_data_json         = self.load_raw_json_data(path = self.raw_data_url)
      self.rdkit_drug_mols  = []
      self.atom_feature_list = atom_feature_list
    ## return rdkit 2D visualization of one molecule using the id
    def get_molecule_by_id (self, identifier: str):
      smiles = self.raw_data_json[identifier]['smiles']
      _molecule= MolFromSmiles(smiles)
      return {identifier : { "mol":_molecule, "smiles":smiles }}
```

```
@lru_cache(maxsize=1)
def get_custom_drug_features(self) -> DrugFeatureSet:
    """Get the drug feature set."""
    path = self.generate_path("drug_set.json")
    raw_data = self.load_raw_json_data(path)
    raw_data = {
        key: {"smiles": value["smiles"], "features": np.array(value["features"]).reshape(1, -1)}
        for key, value in raw_data.items()
    }
    return CustomDrugFeatureSet.from_custom_dict(raw_data, self.atom_feature_list)

## return rdkit 2D visualization of one molecule using the smiles
def get_molecule_by_smiles (self, smiles: str):
  _molecule= MolFromSmiles(smiles)
  return {smiles : _molecule }

def get_molecules(self, data, drugs: Iterable[str]) -> PackedGraph:
    """Get the molecular structures.
    :param drugs: A list of drug identifiers.
    :returns: The molecules batched together for message passing.
    """

    graphs = Graph.pack([data[drug]["molecule"] for drug in drugs])

    return graphs
```

```
  ## create rdkit 2D visualizations of all the returned molecules and save as attribute
  def set_rdkit_drug_mols_by_id(self):
    print('..Generating Drug Molecule Visualizations')
    self.rdkit_drug_mols = [self.get_molecule_by_id(id) for id in list(self.raw_data_json.keys())]

  ## Morgan Fingerprints based on description in paper i.e.
  ## 256Bit Hashed Morgan Fingerprints with a Radius of 2
  def get_morgan_fingerprints(self, smiles, radius=2, nBits=256):
    m1  = MolFromSmiles(smiles)
    fp  = GetHashedMorganFingerprint(mol=m1, radius=radius, nBits=nBits)
    arr = np.zeros((0, ), dtype=np.int8)
    DataStructs.ConvertToNumpyArray(fp, arr)
    if isinstance(arr, np.ndarray):
      return torch.tensor(arr)
    return None

  def draw_rdkit_im(self, mols, legends , molsPerRow=2,  subImgSize= (500, 500) ):
    img=Draw.MolsToGridImage(mols,molsPerRow=molsPerRow,subImgSize=(500,500) , legends = legends)
    return img

  def view_in_3D(self, smiles, limit = 4):
      "View smiles in 3D with a limit"
      _mol0 = mol3D()
      _mol1 = mol3D()
      _mol2 = mol3D()
      _mol3 = mol3D()

      _mols = []
      for  idx, smile in enumerate(smiles[0:limit]):

          if idx == 0 :
            _mol0.read_smiles(smile)
            _mols.append(_mol0)

          elif idx == 1:
            _mol1.read_smiles(smile)
            _mols.append(_mol1)

          elif idx == 2:
            _mol2.read_smiles(smile)
            _mols.append(_mol2)

          elif idx == 3:
            _mol3.read_smiles(smile)
            _mols.append(_mol3)

      return view_structures(_mols)
```

Features and Visualizations:

This section will show how to examine and visualize the data. Although the visualization libraries used here are not part of ChemicalX, they can assist in validating the data.

In [ ]:
```
#View the preprocessed features of one compound as follows:
# This process will also retrieve every compound in the dataset and place it in a local json file.
drugbankddi_rds = CustomRtDatasetLoader("drugbankddi")
remote_drug_features = drugbankddi_rds.get_custom_drug_features()
list(remote_drug_features.keys())[0:1]
```

Out[ ]:
```
['DB04571']
```

In [ ]:
```
remote_drug_features['DB04571']
```

Out[ ]:
```
{'features': tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
          0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 3., 0., 1.,
          0., 0., 0., 2., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 2., 0., 0., 0.,
          0., 0., 0., 0., 0., 1., 0., 0., 0., 3., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 9., 0., 0., 0., 0., 0., 0., 3.,
          0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1.,
          0., 0., 0., 0., 0., 0., 0., 2., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
          2., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 3., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0.,
          1., 0., 0., 0.]]),
 'molecule': Molecule(num_atom=17, num_bond=38)}
```

In [ ]:
```
molecules2 = drugbankddi_rds.get_molecules(remote_drug_features, ['DB04571'])
```

In [ ]:
```
molecules2.atom_feature
```

Out[ ]:
```
tensor([[ 0.0000,  0.0000,  1.0000,  ..., -5.4960, -0.1202,  0.0000],
        [ 0.0000,  0.0000,  1.0000,  ..., -3.9961, -0.1115,  0.0000],
        [ 0.0000,  0.0000,  1.0000,  ..., -3.1073, -1.3199,  0.0000],
        ...,
        [ 0.0000,  0.0000,  1.0000,  ..., -0.4063,  2.9094,  0.0000],
        [ 0.0000,  0.0000,  1.0000,  ..., -1.6922,  0.6519,  0.0000],
        [ 0.0000,  0.0000,  0.0000,  ..., -3.1215,  1.1071,  0.0000]])
```

In [ ]:
```
## To get ALL the smiles and RdKit molecules for the downloaded Dataset, do the following using the same instance of CustomRtDatasetLoader:
drugbankddi_rds.set_rdkit_drug_mols_by_id()
```

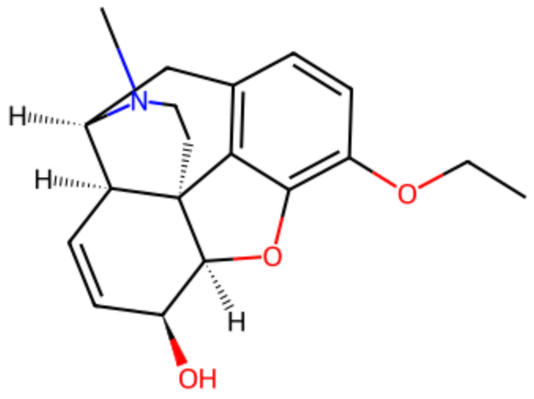
```
..Generating Drug Molecule Visualizations
```

In [ ]:
```
## visualise a few raw examples of RdKit and SMILES data using the rdkit_drug_mols attribute from the CustomDatasetLoader Class
drugbankddi_rds.rdkit_drug_mols[10:12]
```

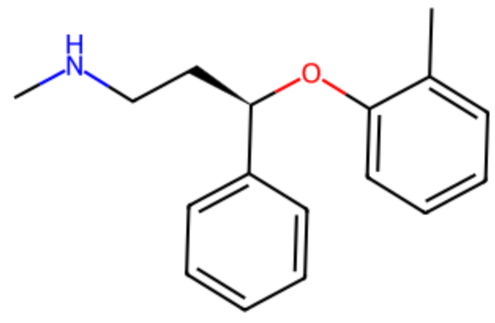
Out[ ]:
```
[{'DB00140': {'mol': <rdkit.Chem.rdchem.Mol at 0x78859701b060>,
   'smiles': 'CC1=C(C)C=C2N(C[C@H](O)[C@H](O)[C@H](O)CO)C3=NC(=O)NC(=O)C3=NC2=C1'}},
 {'DB00821': {'mol': <rdkit.Chem.rdchem.Mol at 0x78859701aff0>,
   'smiles': 'CC(C(O)=O)C1=CC2=C(C=C1)C1=C(N2)C=CC(Cl)=C1'}}]
```

In [ ]:
```
## Visualize 2D Representations of 4 Randomly selected modelcules
idxs = np.random.randint(0, len(drugbankddi_rds.rdkit_drug_mols)-1, 4)
mols    = [list(mol.values())[0]['mol'] for idx,mol in enumerate(drugbankddi_rds.rdkit_drug_mols)  if idx in idxs]
legends = [ list(mol.keys())[0] + ': ' + list(mol.values())[0]['smiles'] for idx,mol in enumerate(drugbankddi_rds.rdkit_drug_mols)  if idx in idxs]
smiles  = [ list(mol.values())[0]['smiles'] for idx,mol in enumerate(drugbankddi_rds.rdkit_drug_mols)  if idx in idxs]
drugbankddi_rds.draw_rdkit_im(mols, legends , molsPerRow=2,  subImgSize= (500,500) )
```

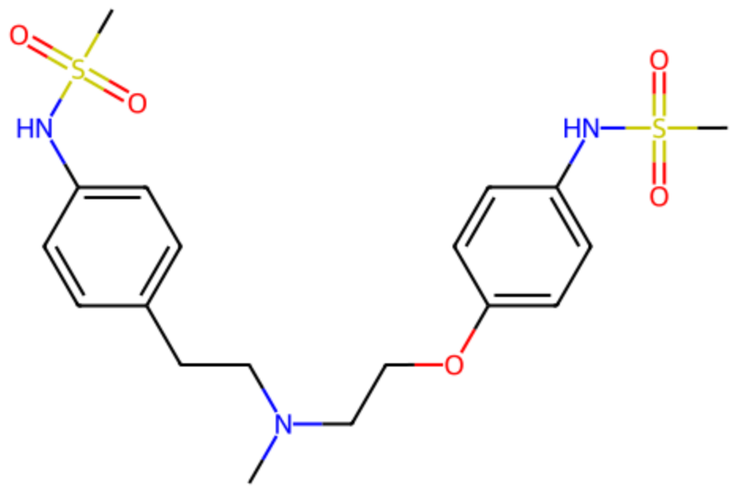
Out[ ]:



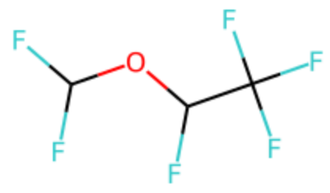
DB01466: [H][C@@]12OC3=C(OCC)C=CC4=C3[C@@]11CCN(C)[C@]([H])(C4)[C@]1([H])C=C[C@@H]2O



DB00289: CNCC[C@@H](OC1=CC=CC=C1C)C1=CC=CC=C1

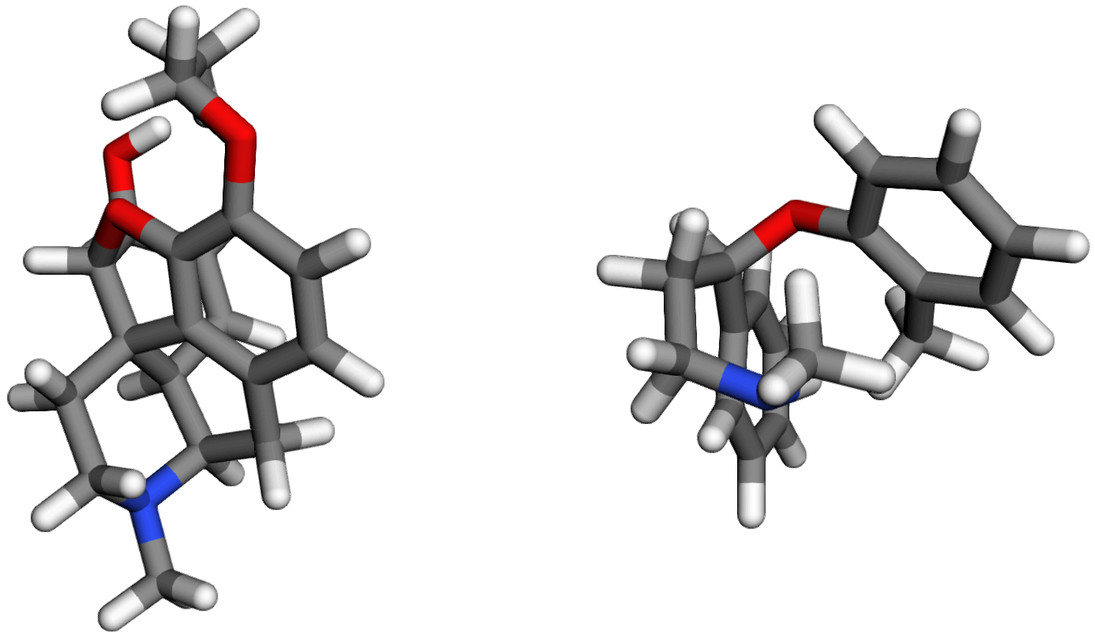


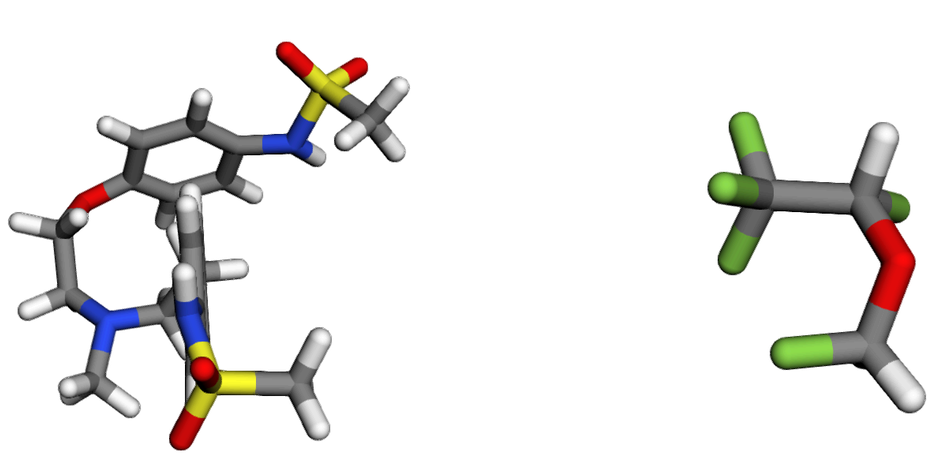
DB00204: CN(CCOC1=CC=C(NS(C)(=O)=O)C=C1)CCC1=CC=C(NS(C)(=O)=O)C=C1



DB01189: FC(F)OC(F)C(F)(F)F

In [ ]:
```
## For Rotatable 3D Structures of the same molecules (This only works in NoteBook Environment)
drugbankddi_rds.view_in_3D(smiles)
```





In [ ]:

Context:

These are one hot encoded vectors of contexts provided from the Data source. They are essentially a label for the context.

Examining Context and Features for DrugBankDDI

See the data source at https://bitbucket.org/kaistsystemsbiology/deepddi/src/master/data/DrugBank_known_ddi.txt

Context Information: https://bitbucket.org/kaistsystemsbiology/deepddi/src/master/data/Interaction_information.csv

DrugBankDDI Data Source: https://www.pnas.org/doi/10.1073/pnas.1803294115

In [ ]:
```
drugbankddi_ctx = drugbankddi_rds.get_context_features()
drugbankddi_ctx ['context_01']
```

Out[ ]:
```
tensor([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

In [ ]:
```
drugbankddi_ctx ['context_02']
```

Out[ ]:
```
tensor([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

To get an idea of the Contexts used for DrugBankDDI, load the Interaction_Contexts.csv file. A copy is also available here https://bitbucket.org/kaistsystemsbiology/deepddi/src/master/data/Interaction_information.csv. Simply copy or download it into a file and rename it.

In [ ]:
```
contexts_df = pd.read_csv("/content/Interaction_information.csv")
```

In [ ]:
```
contexts_df.head()
```

Out[ ]:

	Interaction type	Description	Subject	DDI type
0	67	#Drug1 can cause a decrease in the absorption ...	2	DDI type 1
1	18	#Drug1 can cause an increase in the absorption...	1	DDI type 2
2	13	The absorption of #Drug2 can be decreased when...	1	DDI type 3
3	3	The bioavailability of #Drug2 can be decreased...	1	DDI type 4
4	62	The bioavailability of #Drug2 can be increased...	1	DDI type 5

The context here represents 1 of 86 interaction types. It is a one hot encoded vector for each context. For example context_1 has value 1.0 at position [0], otherwise 0.0, context_2 has 1.0 and position [1] otherwise 0 , and so on. The source data can be founs at can be found at https://bitbucket.org/kaistsystemsbiology/deepddi/src/master/data/Interaction_information.csv .

Examining Context and Features for DrugCombDB

The identifiers/keys for the context in this dataset can be seen below. See more details at http://drugcombdb.denglab.org

In [ ]:
```
drugcombdb_rds = CustomRtDatasetLoader("drugcombdb")
drubcombdb_drug_features =drugcombdb_rds.get_drug_features()
```

In [ ]:
```
drugcombdb_ctx = drugcombdb_rds.get_context_features()
```

In [ ]:
```
## This is list of contexts from Drugcombdb
print(list(drugcombdb_ctx.keys()))
```

```
['RXF 393', 'HS 578T', 'SK-MEL-5', 'UO-31', 'UWB1289', 'T-47D', 'MDAMB436', 'U-HO1', 'UACC-257', 'SK-OV-3', 'HCT-15', 'JHH-520', 'ES2', 'EW-8', 'MALME-3M', 'LOVO', 'NCIH1650', 'MDA-MB-435', 'SK-MEL-28', 'PC-3',
'UWB1289+BRCA1', 'JHH-136', 'NCI-H226', 'COLO 205', 'VCAP', 'PA1', 'MDA-MB-468', 'TMD8', 'HOP-62', 'A375', 'RD', 'DU-145', '786-0', 'OVCAR-8', 'OVCAR-5', 'SF-539', 'SW-620', 'K-562', 'KM12', 'IGROV1', 'ZR751',
'RPMI-8226', 'SK-MEL-2', 'KBM-7', 'L-1236', 'OVCAR3', 'NCI-H322M', 'NCI-H522', 'U251', 'KPL1', 'HOP-92', 'NCI-ADR-RES', 'MDA-MB-231', 'SF-268', 'TK-10', 'A2058', 'MOLT-4', 'CAKI-1', 'CAOV3', 'OV90', '3D7', 'MCF
7', 'SR', 'HCC-2998', 'T98G', 'DD2', 'HDLM-2', 'Rh36', 'OVCAR-4', 'RPMI7951', 'HCT116', 'NCIH520', 'LOX IMVI', 'CCRF-CEM', 'CTR', 'SF-295', 'EFM192B', 'SKMEL30', 'HT29', 'A549', 'EKVX', 'COLO320DM', 'DIPG25', 'A
-673', 'HT144', 'M14', 'TC-32', 'SW837', 'MSTO', 'BT-549', 'SU-DIPG-XIII', 'RKO', 'SKMES1', 'A2780', 'NCI-H460', 'NCIH23', 'NCIH2122', 'TC-71', 'SNB-19', 'A427', 'ACHN', 'L-428', 'DLD1', 'SNB-75', 'SMS-CTR', 'SN
12C', 'LNCAP', 'HB3', 'OCUBM', 'HL-60(TB)', 'UACC62', 'A498']
```

Labeled Triples:

These include the Identifiers of the two drugs to be compared, the id for the context, and the label. For DrugBankDDI a label on 1.0 indicates a known interaction, ***while 0.0 may indicate no interaction or unkown***. The data is held in a pandas DataFrame. These are essentially the input labels.

This demonstration will focus on DrugBankDDI for most of this data loading demonstration, then switches to DeepCombDB for training a model.

In [ ]:
```
labeled_triples_drugbank_ddi = drugbankddi_rds.get_labeled_triples()
##View the labelled Triples
labeled_triples_drugbank_ddi.data
```

Out[ ]:

	drug_1	drug_2	context	label
0	DB04571	DB00460	context_01	1.0
1	DB00855	DB00460	context_01	1.0
2	DB09536	DB00460	context_01	1.0
3	DB01600	DB00460	context_01	1.0
4	DB09000	DB00460	context_01	1.0
...	...	...	...	...
383611	DB06335	DB06706	context_02	0.0
383612	DB00946	DB00575	context_60	0.0
383613	DB01081	DB04576	context_25	0.0
383614	DB01463	DB08899	context_29	0.0
383615	DB00213	DB01489	context_17	0.0

383616 rows × 4 columns

Examining some values with known interactions of a given context.

In [ ]:
```
context_id = "context_30"
context_id_int = int(context_id.replace("context_", ""))
for descr in contexts_df["Description"][contexts_df["Interaction type"]== context_id_int].values:
  print(context_id , " - ", descr)
```

```
context_30  -  #Drug1 may increase the orthostatic hypotensive activities of #Drug2.
```

View some known interaction labelled triples for the context

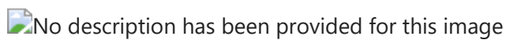
In [ ]:
```
filter = (labeled_triples_drugbank_ddi.data["context"] == context_id) & (labeled_triples_drugbank_ddi.data["label"]==1.0)
labeled_triples_drugbank_ddi.data[filter]
```

Out[ ]:

	drug_1	drug_2	context	label
28072	DB00960	DB00457	context_30	1.0
28073	DB01297	DB00457	context_30	1.0
28074	DB01351	DB00999	context_30	1.0
28075	DB01154	DB00436	context_30	1.0
28076	DB00571	DB00598	context_30	1.0
...	...	...	...	...
28683	DB04846	DB00598	context_30	1.0
28684	DB00489	DB00476	context_30	1.0
28685	DB01136	DB00457	context_30	1.0
28686	DB01580	DB01162	context_30	1.0
28687	DB04846	DB00590	context_30	1.0

616 rows × 4 columns

To validate the Interaction Data, the Interaction Checker at https://dev.drugbank.com/demo/ddi_checker can be used. As an example, Pindolol - DB00960 and Prazosin - DB00457 from the labelled triples were used.


No description has been provided for this image

As the image shows, the interaction context matches the description of context_30 above.

Other Datasets , Brief Example

To get an idea of how to access data for the other datasets, here is a brief example of the DrugCombDB dataset Labelled Triples. The structure is very similar to DrugBankDDI.

In [ ]:
```
labeled_triples_drugcombdb = drugcombdb_rds.get_labeled_triples()
labeled_triples_drugcombdb.data
```

Out[ ]:

	drug_1	drug_2	context	label
0	3385	11960529	A2058	1.0
1	3385	24856436	A2058	1.0
2	3385	11977753	A2058	1.0
3	3385	387447	A2058	0.0
4	3385	3062316	A2058	1.0
...	...	...	...	...
191386	46918825	11598628	A-673	1.0
191387	68210102	11598628	A-673	0.0
191388	5281607	11598628	A-673	0.0
191389	515328	16654980	U-HO1	1.0
191390	9806229	148198	U-HO1	1.0

191391 rows × 4 columns

Loading Custom Data

In [ ]:
```
os.chdir("/usr/local/lib/python3.10/dist-packages/chemicalx/data")
```

Example: Adding new drugs from DrugBank:

The Drugs Nirogacestat, with brand name Ogsiveo was approved by the Food and Drug Administration (FDA) in November 2023. It is not in the list we downloaded from the ChemicalX repository because it is a static file that has not been updated since 2022. We will attempt to preprocess and load it locally.

This example uses a context and interaction data found on DrugBank.com. The DrugBank Accession Number of Nirogacestat/Ogsiveo is DB12005.

DrugBank.com shows that when Aluminum hydroxide - DB06723 interacts with Nirogacestat is DB12005, ***the serum concentration of Nirogacestat can be decreased when it is combined with Aluminum hydroxide.*** This corresponds with context_73 in the contexts dataset; Aluminum hydroxide - DB06723 already exists in the dataset.

Important: *This test is not to be taken as fact. It is merely an example of data preparation and explorartion for ChemicalX Drug-Drug Interaction deep learning. Please consult a medical or pharmaceutical professional for drug-drug interaction related questions*

In [ ]:
```
##Viweing the context chosen
contexts_df[contexts_df["Interaction type"] == 73]

for x in contexts_df["Description"][contexts_df["Interaction type"] == 73].values:
  print(x)
```

```
The serum concentration of #Drug2 can be increased when it is combined with #Drug1.
```

In [ ]:
```
## Features for Aluminum hydroxide - DB06723
remote_drug_features["DB06723"]
```

```
Out[ ]: {'features': tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 3., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0.]]),
 'molecule': Molecule(num_atom=4, num_bond=0)}
```

```
In [ ]: ##
        ## These are hardcoded smiles but can be accessed through an API.
        ## The API requires manual Approval from DrugBank Online and is also a commercial, paid product.
        ## However, for this quick example, the smiles and Drug Accession Number were copied and pasted

        ## Adding Nirogacestat/Ogsiveo to featureset
        DrugBank_Accession_Number = "DB12005"
        smiles = "CCC[C@H](N[C@H]1CCC2=CC(F)=CC(F)=C2C1)C(=O)NC1=CN(C=N1)C(C)(C)CNCC(C)(C)C"
        try:
          ##Checking if record exists in the dataset
          x = remote_drug_features[DrugBank_Accession_Number]

        except KeyError:
          print(f'{DrugBank_Accession_Number} not found, preparing data...')
          features = drugbankddi_rds.get_morgan_fingerprints(smiles, radius=2, nBits=256)
          ##Torchdrug's Molecule Class is used to genetate a molecular graph
          remote_drug_features[DrugBank_Accession_Number] = {"features" : features , "molecule": Molecule.from_smiles(smiles)}

DB12005 not found, preparing data...
```

```
In [ ]: ##The newly created Feature
        remote_drug_features["DB12005"]

Out[ ]: {'features': tensor([0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         1, 0, 1, 1, 0, 0, 0, 0, 1, 7, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1,
         1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 1, 0, 0, 1, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
         0, 0, 1, 0, 5, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
         0, 0, 1, 0, 1, 1, 0, 0, 3, 0, 0, 0, 0, 0, 1, 0, 7, 0, 2, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 0, 0, 1, 0, 0, 2,
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
         0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], dtype=torch.int8),
 'molecule': Molecule(num_atom=35, num_bond=74)}
```

Adding New data to Labelled Triples

- This can be accomplished by appending the new row(s) to the labelled triples dataset.

In [ ]:
```
##get the columns from the existing dataset
columns = labeled_triples_drugbank_ddi.data.columns
columns
```

Out[ ]:
```
Index(['drug_1', 'drug_2', 'context', 'label'], dtype='object')
```

In [ ]:
```
## this adds the new_data using the known context and interaction label into the dataset
new_context = "context_73"
new_ddi_df  = pd.DataFrame(data= [["DB06723","DB12005", new_context, 1.0]], columns = labeled_triples_drugbank_ddi.data.columns,  )
```

In [ ]:
```
new_ddi_df
```

Out[ ]:

	drug_1	drug_2	context	label
0	DB06723	DB12005	context_73	1.0

In [ ]:
```
new_ddi_df
```

Out[ ]:

	drug_1	drug_2	context	label
0	DB06723	DB12005	context_73	1.0

In [ ]:
```
#append to the end of the existing data
pd.concat([labeled_triples_drugbank_ddi.data, new_ddi_df], axis = 0, ignore_index=True)
```

Out[ ]:

	drug_1	drug_2	context	label
0	DB04571	DB00460	context_01	1.0
1	DB00855	DB00460	context_01	1.0
2	DB09536	DB00460	context_01	1.0
3	DB01600	DB00460	context_01	1.0
4	DB09000	DB00460	context_01	1.0
...	...	...	...	...
383612	DB00946	DB00575	context_60	0.0
383613	DB01081	DB04576	context_25	0.0
383614	DB01463	DB08899	context_29	0.0
383615	DB00213	DB01489	context_17	0.0
383616	DB06723	DB12005	context_73	1.0

383617 rows × 4 columns

In [ ]:
```
##original
labeled_triples_drugbank_ddi.data.tail()
```

Out[ ]:

	drug_1	drug_2	context	label
383611	DB06335	DB06706	context_02	0.0
383612	DB00946	DB00575	context_60	0.0
383613	DB01081	DB04576	context_25	0.0
383614	DB01463	DB08899	context_29	0.0
383615	DB00213	DB01489	context_17	0.0

In [ ]:
```
## After confirming indexing  recreate the data
labeled_triples_drugbank_ddi.data = pd.concat([labeled_triples_drugbank_ddi.data, new_ddi_df], axis = 0, ignore_index=True)
```

In [ ]:
```
## this confirms that the new data has been added.
labeled_triples_drugbank_ddi.data.tail()
```

Out[ ]:

	drug_1	drug_2	context	label
383612	DB00946	DB00575	context_60	0.0
383613	DB01081	DB04576	context_25	0.0
383614	DB01463	DB08899	context_29	0.0
383615	DB00213	DB01489	context_17	0.0
383616	DB06723	DB12005	context_73	1.0

Loading Custom Data

In this example, we will use the custom dataset with the new features and labels added.

In [ ]:
```
from chemicalx import pipeline
from chemicalx.models import DeepSynergy
from chemicalx.data import DrugbankDDI

## To load data using the provided utilities,use the datasets's Class
loader = DrugbankDDI()
default_context_set = loader.get_context_features()
default_drug_set = loader.get_drug_features()
default_triples = loader.get_labeled_triples()
```

In [ ]:
```
## confirming that the new drug set has the new data
print( "original data count: ", len(list(default_drug_set.keys())))
print( "new data count :" , len(list(remote_drug_features.keys())))
conf = "exists" if not default_drug_set.get(DrugBank_Accession_Number) is None else "does not exist"
print( f"The new drug {DrugBank_Accession_Number} {conf} in the default_drug_set.")
conf = "exists" if not remote_drug_features.get(DrugBank_Accession_Number) is None else "does not exist"
print( f"The new drug {DrugBank_Accession_Number} {conf} in the new data set.")
```

```
original data count:  1706
new data count : 1707
The new drug DB12005 does not exist in the default_drug_set.
The new drug DB12005 exists in the new data set.
```

In [ ]:
```python
##examining whether  the default dataset does not contain the new data
default_drug_set.get("DB12005") is None
```

Out[ ]:
```
True
```

Training a Model:

ChemicalX uses Geometric Deep Learning, Deep Chemistry layers and many other utiliies available in Pytorch.The models provided can perform drug-pair scoring tasks on datasets with hundreds of thoudands of compounds, all on widely available hardware.

The Deep Synergy Model can be used for Drug Pair Synergy and has been evaluated in the ChemicalX paper with a AUROC score of 0.744 on **DrugComb**. This demonstration will train a model using the **DrugCombDB data**, while using the data loaded with the custom data loader class. This allows some flexibility for users to add new data in the same way the data was added into the DrugBankDDI dataset.

In [ ]:
```python
##create instance of Custom Dataset Loader
drugcombdb_rds = CustomRtDatasetLoader("drugcombdb")
```

In [ ]:
```python
#this function can be used to add new data
def add_new_features(drug_features , new_id, smiles,  loader : CustomRtDatasetLoader):
    print(f'{new_id} not found, preparing data...')
    features = loader.get_morgan_fingerprints(smiles, radius=2, nBits=256)
    ##Torchdrug's Molecule Class is used to genetate a molecular graph
    drug_features[new_id] = {"features" : features , "molecule": Molecule.from_smiles(smiles)}
    return drug_features
```

In [ ]:
```python
## For this hypothetical example , a feature for SUCROSE
## will be created for the DrugCombDB dataset

smiles = "C(C1C(C(C(C(O1)OC2(C(C(C(O2)CO)O)O)CO)O)O)O)O"
test_features =drugcombdb_rds.get_drug_features()
new_test_features = add_new_features(drug_features =test_features ,
                                     new_id  = '99999999',
                                     smiles  = smiles,
                                     loader  = drugcombdb_rds)
```

```
99999999 not found, preparing data...
```

In [ ]:
```python
## checking the new data
new_test_features['99999999']
```

Out[ ]:
```
{'features': tensor([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 8, 0, 0, 0, 1, 0, 0, 1, 0,
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0, 1, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 4, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 6, 0, 0, 2, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 8, 0, 0, 0, 0], dtype=torch.int8),
 'molecule': Molecule(num_atom=23, num_bond=48)}
```

In [ ]:
```
##Once the features are added, the additional feature must be added to the labelled Triples as shown in
## the DrugBankDDI Example.
##Clean up
del new_test_features
del test_features
```

Loading Data and Model Training:

In [ ]:
```
drubcombdb_drug_features          =drugcombdb_rds.get_drug_features()
drugcombdb_ctx                    =drugcombdb_rds.get_context_features()
drugcombdb_labeled_triples        =drugcombdb_rds.get_labeled_triples()
```

In [ ]:
```
## examine labels

drugcombdb_labeled_triples.data.head()
```

Out[ ]:

	drug_1	drug_2	context	label
0	3385	11960529	A2058	1.0
1	3385	24856436	A2058	1.0
2	3385	11977753	A2058	1.0
3	3385	387447	A2058	0.0
4	3385	3062316	A2058	1.0

Checking for Class Imbalance

Since this is Binary Classification, checking and rectifying the data for imbalance can improve the model's performance.

In [ ]:
```
## check for imbalance in the dataset labels
negatives = drugcombdb_labeled_triples.get_negative_rate()
negatives
```

Out[ ]:
```
0.7067416963180086
```

In [ ]:
```
1
```

Out[ ]:
```
1
```

The dataset is Skewed towards the negative class, a common issue in many binary classification problems. There are many methods to rectify this including over and undersampling. However, in this demonstration, we will run this as it is to showcase the library.

In [ ]:
```
from chemicalx.data import  BatchGenerator, DrugFeatureSet, DrugPairBatch


drugcombdb_rds = CustomRtDatasetLoader("drugcombdb", atom_feature_list)
drug_set       =drugcombdb_rds.get_drug_features()
context_set    =drugcombdb_rds.get_context_features()
triples        =drugcombdb_rds.get_labeled_triples()
```

```
train_data, test_data = triples.train_test_split(train_size=0.6,random_state= 0)


## Elect to include molecules in the training. This will result in more training time.
molecules  = False


def data_generator(use_drug_molecules ,
                   use_context,
                   use_drug_features,
                   train_data,
                   test_data,
                   batch_size  =1024,
                   context_set = context_set,
                   drug_set    = drug_set,
                   ):
    train_generator = BatchGenerator(batch_size=batch_size,
                                     context_features= use_context,
                                     drug_features=use_drug_features,
                                     drug_molecules=use_drug_molecules,
                                     context_feature_set=context_set,
                                     drug_feature_set=drug_set,
                                     labeled_triples=train_data)


    valid_generator = BatchGenerator(batch_size=batch_size,
                                     context_features=use_context,
                                     drug_features=use_drug_features,
                                     drug_molecules=use_drug_molecules,
                                     context_feature_set=context_set,
                                     drug_feature_set=drug_set,
                                     labeled_triples=test_data)


    return train_generator, valid_generator


train_generator, valid_generator = data_generator(False,True, True, train_data, test_data, )
```

In [ ]:

In [ ]:
```
drug_set.get_molecules(list(drug_set.keys()))[0].atom_feature
```

Out[ ]:
```
tensor([[0, 0, 1,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 1, 1],
        [0, 0, 1,  ..., 0, 1, 1],
        ...,
        [0, 0, 1,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0]])
```

Training a DeepSynergy Model on DrugCombDB data.

This method does not use molecululular structure data. It takes the prepared drug context features.

In [ ]:

```
from chemicalx.models import DeepSynergy, DeepDrug, SSIDDI


model = DeepSynergy(context_channels=112,
                    drug_channels=256)

torch.cuda.manual_seed(23)
torch.manual_seed(23)
epochs = 20
optimizer = torch.optim.Adam(model.parameters())
loss = torch.nn.BCELoss()


def train(model, train_gen, valid_gen, loss_fn, optim, epochs):

    training_losses = []
    valid_losses    = []
    training_acc    = []
    valid_acc       = []
    for epoch in range(epochs):
        train_batch_counter = 0
        train_loss_sum      = 0

        valid_batch_counter  = 0
        valid_loss_sum       = 0
        train_epoch_acc      = []
        valid_epoch_acc      = []

        model.train()
    for batch in train_gen:
        optim.zero_grad()
        prediction = model(batch.context_features,
                           batch.drug_features_left,
                           batch.drug_features_right)
        loss_value = loss_fn(prediction, batch.labels)
        ## Accuracy added December 2024

        acc_value  = torch.sum(torch.where(prediction >0.5, 1.0, 0.0 )==batch.labels)/len(batch.labels)
        loss_value.backward()
        optim.step()
        train_loss_sum+= loss_value.item()
        train_epoch_acc.append(acc_value.item())
        train_batch_counter += 1

    model.eval()
    for vbatch in valid_gen :
        prediction = model(vbatch.context_features,
                           vbatch.drug_features_left,
                           vbatch.drug_features_right)
        valid_loss_value = loss_fn(prediction, vbatch.labels)
        ## Accuracy added December 2024
        val_acc_value  = torch.sum(torch.where(prediction >0.5, 1.0, 0.0 ) ==vbatch.labels)/len(vbatch.labels)
        valid_loss_sum+= valid_loss_value.item()
        valid_epoch_acc.append(val_acc_value.item())
        valid_batch_counter += 1

    train_epoch_loss = train_loss_sum/train_batch_counter
    valid_epoch_loss = valid_loss_sum/valid_batch_counter
```

```
        train_acc   = sum(train_epoch_acc)/len(train_epoch_acc)
        val_acc     = sum(valid_epoch_acc)/len(valid_epoch_acc)

        print("Epoch Loss: {a:4d} | Train Loss: {b:4.8f},  Valid Loss: {c:4.8f}, Train Acc: {d:4.8f},  Valid Acc: {e:4.8f}".
         format(a = epoch + 1, b = train_epoch_loss, c=valid_epoch_loss, d=train_acc, e=val_acc))
        training_losses.append(train_epoch_loss)
        valid_losses.append(valid_epoch_loss)
        training_acc.append(train_acc)
        valid_acc.append(val_acc)

    return training_losses, valid_losses, training_acc, valid_acc
```

In [ ]:
```
train_losses, valid_losses , training_acc, valid_acc= train(model, train_generator, valid_generator, loss, optimizer, epochs)
```

```
Epoch Loss:    1 | Train Loss: 0.57218684,  Valid Loss: 0.52839263, Train Acc: 0.71815760,  Valid Acc: 0.74830664
Epoch Loss:    2 | Train Loss: 0.52111395,  Valid Loss: 0.50541970, Train Acc: 0.75455121,  Valid Acc: 0.76539749
Epoch Loss:    3 | Train Loss: 0.50096828,  Valid Loss: 0.48720048, Train Acc: 0.76968231,  Valid Acc: 0.77674212
Epoch Loss:    4 | Train Loss: 0.48662430,  Valid Loss: 0.47830729, Train Acc: 0.77913729,  Valid Acc: 0.78198142
Epoch Loss:    5 | Train Loss: 0.47631492,  Valid Loss: 0.47316041, Train Acc: 0.78655178,  Valid Acc: 0.78554711
Epoch Loss:    6 | Train Loss: 0.46712319,  Valid Loss: 0.46524912, Train Acc: 0.79107174,  Valid Acc: 0.79016227
Epoch Loss:    7 | Train Loss: 0.45931563,  Valid Loss: 0.47072417, Train Acc: 0.79579023,  Valid Acc: 0.78390995
Epoch Loss:    8 | Train Loss: 0.45471168,  Valid Loss: 0.46601315, Train Acc: 0.79746716,  Valid Acc: 0.78740185
Epoch Loss:    9 | Train Loss: 0.44870501,  Valid Loss: 0.46156796, Train Acc: 0.80043166,  Valid Acc: 0.79158356
Epoch Loss:   10 | Train Loss: 0.44464859,  Valid Loss: 0.45548678, Train Acc: 0.80268785,  Valid Acc: 0.79539086
Epoch Loss:   11 | Train Loss: 0.43989401,  Valid Loss: 0.45444750, Train Acc: 0.80549631,  Valid Acc: 0.79468282
Epoch Loss:   12 | Train Loss: 0.43530977,  Valid Loss: 0.45139972, Train Acc: 0.80678375,  Valid Acc: 0.79695914
Epoch Loss:   13 | Train Loss: 0.43169743,  Valid Loss: 0.45094467, Train Acc: 0.80868514,  Valid Acc: 0.79646290
Epoch Loss:   14 | Train Loss: 0.42835499,  Valid Loss: 0.45060199, Train Acc: 0.81038717,  Valid Acc: 0.79623923
Epoch Loss:   15 | Train Loss: 0.42345455,  Valid Loss: 0.45119561, Train Acc: 0.81262691,  Valid Acc: 0.79669959
Epoch Loss:   16 | Train Loss: 0.42148075,  Valid Loss: 0.44897700, Train Acc: 0.81293661,  Valid Acc: 0.79823605
Epoch Loss:   17 | Train Loss: 0.41692122,  Valid Loss: 0.45439167, Train Acc: 0.81644650,  Valid Acc: 0.79561568
Epoch Loss:   18 | Train Loss: 0.41459090,  Valid Loss: 0.46671413, Train Acc: 0.81609170,  Valid Acc: 0.78721869
Epoch Loss:   19 | Train Loss: 0.41458767,  Valid Loss: 0.45078208, Train Acc: 0.81747527,  Valid Acc: 0.79740302
Epoch Loss:   20 | Train Loss: 0.40955742,  Valid Loss: 0.45233799, Train Acc: 0.81935026,  Valid Acc: 0.79864173
```

Next - Training using molecular structures and features.

In [ ]:
```
ds_name = "drugcombdb"

drug_rds = CustomRtDatasetLoader(ds_name)
drug_features         =drug_rds.get_custom_drug_features()
drug_ctx              =drug_rds.get_context_features()
drug_labeled_triples  =drug_rds.get_labeled_triples()
##Loading the data
context_set = drug_ctx
drug_set    = drug_features
triples     = drug_labeled_triples


train_data, test_data = triples.train_test_split(train_size=0.6,random_state=1024)
```

In [ ]:
```
import gc


batch_size = 1024
train_generator, valid_generator = data_generator(True,
```

```
                                            False,
                                            True,
                                            train_data,
                                            test_data,
                                            batch_size,
                                            context_set = context_set,
                                            drug_set    = drug_set,)

t        = iter(train_generator)
batch1   = next(t)
channels = batch1.drug_molecules_left.data_dict['atom_feature'].shape[-1]


##Model Setup
molecule_model = SSIDDI(  molecule_channels= channels, hidden_channels=(500, 500), head_number=(4, 4))
model_type     = 'ssiddi'
torch.cuda.manual_seed(23)
torch.manual_seed(23)
epochs    = 20
optimizer = torch.optim.Adam(molecule_model.parameters())
loss      = torch.nn.BCELoss()

molecule_model.to(0)
def train_w_molecules(model, train_gen, valid_gen, loss_fn, optim, epochs):

    training_losses = []
    valid_losses    = []
    training_acc    = []
    valid_acc       = []
    for epoch in range(epochs):
        train_batch_counter = 0
        train_loss_sum      = 0

        valid_batch_counter  = 0
        valid_loss_sum       = 0
        train_epoch_acc      = []
        valid_epoch_acc      = []

        model.train()
        for batch in train_gen:
            optim.zero_grad()

            mols_left  = batch.drug_molecules_left.to(0)
            mols_right = batch.drug_molecules_right.to(0)
            labels     = batch.labels.to(0)

            prediction = model(
                               mols_left,
                               mols_right)
            loss_value = loss_fn(prediction,labels)
            ## Accuracy added December 2024

            acc_value  = torch.sum(torch.where(prediction >0.5, 1.0, 0.0 )==labels)/len(labels)
            loss_value.backward()
            optim.step()
            train_loss_sum+= loss_value.item()
            train_epoch_acc.append(acc_value.item())
            train_batch_counter += 1
```

```
    model.eval()
    for vbatch in valid_gen :

        vmols_left = vbatch.drug_molecules_left.to(0)
        vmols_right= vbatch.drug_molecules_right.to(0)
        vlabels    = vbatch.labels.to(0)

        prediction = model(
                         vmols_left,
                         vmols_right)
        valid_loss_value = loss_fn(prediction, vlabels)
        ## Accuracy added December 2024
        val_acc_value  = torch.sum(torch.where(prediction >0.5, 1.0, 0.0 ) ==vlabels)/len(vlabels)
        valid_loss_sum+= valid_loss_value.item()
        valid_epoch_acc.append(val_acc_value.item())
        valid_batch_counter += 1

    train_epoch_loss = train_loss_sum/train_batch_counter
    valid_epoch_loss = valid_loss_sum/valid_batch_counter


    train_acc   = sum(train_epoch_acc)/len(train_epoch_acc)
    val_acc     = sum(valid_epoch_acc)/len(valid_epoch_acc)

    print("Epoch Loss: {a:4d} | Train Loss: {b:4.8f},  Valid Loss: {c:4.8f}, Train Acc: {d:4.8f},  Valid Acc: {e:4.8f}".
     format(a = epoch + 1, b = train_epoch_loss, c=valid_epoch_loss, d=train_acc, e=val_acc))
    training_losses.append(train_epoch_loss)
    valid_losses.append(valid_epoch_loss)
    training_acc.append(train_acc)
    valid_acc.append(val_acc)
    gc.collect()
    return training_losses, valid_losses, training_acc, valid_acc
```

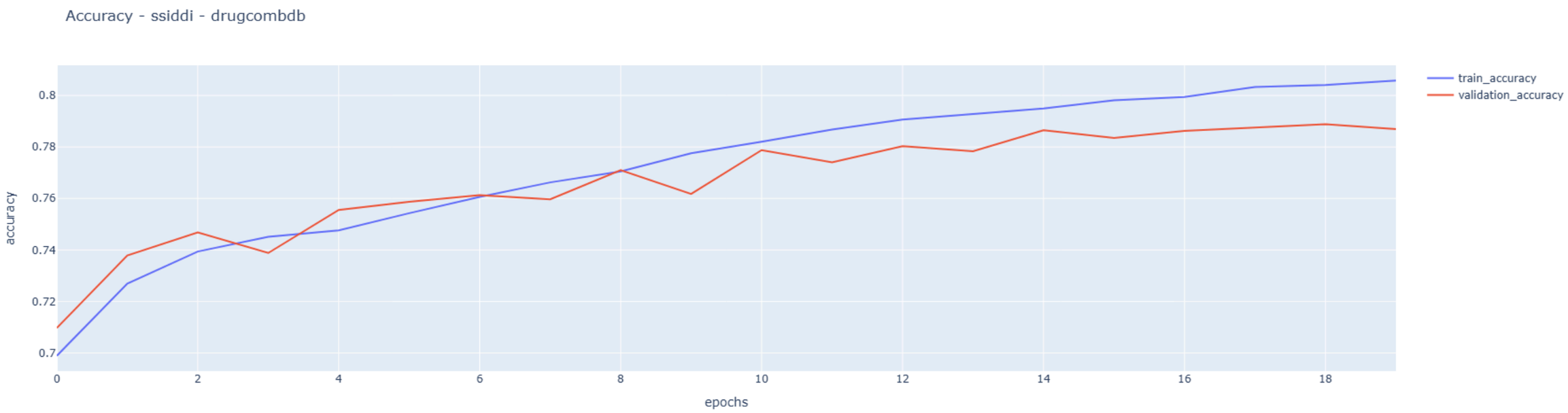
In [ ]:
```
train_losses, valid_losses , training_acc, valid_acc= train_w_molecules(molecule_model, train_generator, valid_generator, loss, optimizer, epochs)
```

```
Epoch Loss:    1 | Train Loss: 0.60926508,  Valid Loss: 0.57342262, Train Acc: 0.69892953,  Valid Acc: 0.70974065
Epoch Loss:    2 | Train Loss: 0.55882924,  Valid Loss: 0.54513713, Train Acc: 0.72692073,  Valid Acc: 0.73785843
Epoch Loss:    3 | Train Loss: 0.54117257,  Valid Loss: 0.53042213, Train Acc: 0.73940852,  Valid Acc: 0.74680288
Epoch Loss:    4 | Train Loss: 0.53128095,  Valid Loss: 0.53985101, Train Acc: 0.74513873,  Valid Acc: 0.73885582
Epoch Loss:    5 | Train Loss: 0.52476417,  Valid Loss: 0.51951362, Train Acc: 0.74759287,  Valid Acc: 0.75555028
Epoch Loss:    6 | Train Loss: 0.51383577,  Valid Loss: 0.50897280, Train Acc: 0.75428212,  Valid Acc: 0.75869785
Epoch Loss:    7 | Train Loss: 0.50211492,  Valid Loss: 0.50625861, Train Acc: 0.76059136,  Valid Acc: 0.76132054
Epoch Loss:    8 | Train Loss: 0.49148005,  Valid Loss: 0.50498110, Train Acc: 0.76625197,  Valid Acc: 0.75967556
Epoch Loss:    9 | Train Loss: 0.48387195,  Valid Loss: 0.48890051, Train Acc: 0.77051267,  Valid Acc: 0.77103842
Epoch Loss:   10 | Train Loss: 0.47393667,  Valid Loss: 0.49795789, Train Acc: 0.77759165,  Valid Acc: 0.76176353
Epoch Loss:   11 | Train Loss: 0.46744712,  Valid Loss: 0.47563878, Train Acc: 0.78206733,  Valid Acc: 0.77876556
Epoch Loss:   12 | Train Loss: 0.45931191,  Valid Loss: 0.48628364, Train Acc: 0.78679459,  Valid Acc: 0.77405578
Epoch Loss:   13 | Train Loss: 0.45250645,  Valid Loss: 0.47383396, Train Acc: 0.79066532,  Valid Acc: 0.78034745
Epoch Loss:   14 | Train Loss: 0.44790523,  Valid Loss: 0.47383639, Train Acc: 0.79274926,  Valid Acc: 0.77832692
Epoch Loss:   15 | Train Loss: 0.44257255,  Valid Loss: 0.46643042, Train Acc: 0.79493608,  Valid Acc: 0.78648316
Epoch Loss:   16 | Train Loss: 0.43766886,  Valid Loss: 0.46672165, Train Acc: 0.79810692,  Valid Acc: 0.78353900
Epoch Loss:   17 | Train Loss: 0.43441488,  Valid Loss: 0.46764218, Train Acc: 0.79943757,  Valid Acc: 0.78627974
Epoch Loss:   18 | Train Loss: 0.42784810,  Valid Loss: 0.46995010, Train Acc: 0.80329280,  Valid Acc: 0.78759311
Epoch Loss:   19 | Train Loss: 0.42466066,  Valid Loss: 0.46488524, Train Acc: 0.80407805,  Valid Acc: 0.78886452
Epoch Loss:   20 | Train Loss: 0.42195874,  Valid Loss: 0.46722015, Train Acc: 0.80578174,  Valid Acc: 0.78691950
```

Visualizations

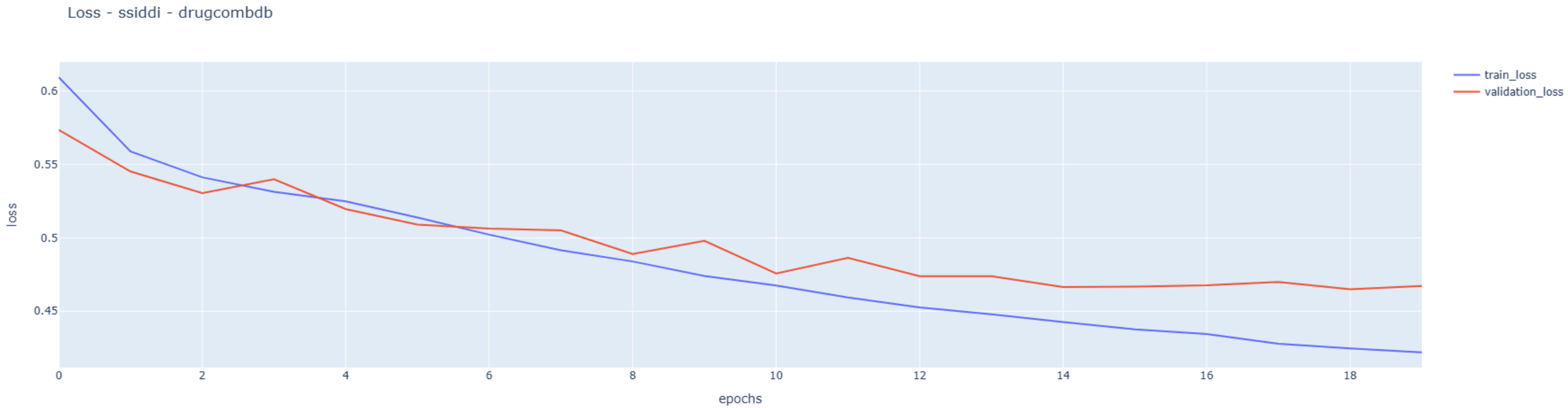
In [ ]:
```
import plotly.graph_objects as go
import plotly.figure_factory as ff
import plotly.express as px
import plotly.io as pio

np.random.seed(23)


def plot_multiple_lines(x      : list[list],
                        y_list : list[list],
                        series : list[str],
                        title  : str,
                        axis_titles : dict[str], as_image = False)->None:
    """
     Multiseries line chart
    """
    fig = go.Figure()

    for i,y  in enumerate(y_list):
        fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name=series[i]))

    # Update layout (optional)
    fig.update_layout(title= title,
                      xaxis_title=axis_titles['x'],
                      yaxis_title= axis_titles['y'])

    if as_image :
        # Convert figure to an image
        img_bytes = pio.to_image(fig, format="png")
        Image(img_bytes)


    else:
        # Show the plot
        fig.show()
```

In [ ]:
```
x      = [n for n in range(0, epochs)]
y_list = training_acc, valid_acc
series = ['train_accuracy', 'validation_accuracy']
title  = f"Accuracy - {model_type} - {ds_name}"
axis_titles = {'x' : 'epochs', 'y' : 'accuracy'}


plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```


Accuracy - ssiddi - drugcombdb
train_accuracy
validation_accuracy
accuracy
epochs
0.8
0.78
0.76
0.74
0.72
0.7
0
2
4
6
8
10
12
14
16
18

DrugcombDB is an unbalanced dataset. In a later section, the Binary Cross Entropy Loss is replaced by the Binary Cross Entropy Loss With Logits to use a weight in the loss function, and the final Sigmoid Layer is removed Skipped to accomodate this.

In [ ]:
```
x      = [n for n in range(0, epochs)]
y_list = train_losses, valid_losses
series = ['train_loss', 'validation_loss']
title  = f"Loss - {model_type} - {ds_name}"
axis_titles = {'x' : 'epochs', 'y' : 'loss'}

plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```



In [ ]:
```
from datetime import datetime , timezone

SAVE_MODEL  = True
models_path = '/content/drive/MyDrive/chemicalx_models'

def save_model(model , base_path ):
    now = datetime.now().replace(tzinfo=timezone.utc)

    now = now.strftime('%Y%m%d%H%M%S')

    save_path = f'{base_path}_{now}.pth'

    print(save_path)

    torch.save(model.state_dict(), save_path)

    return save_path
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive',  force_remount=True)
```

```
Mounted at /content/drive
```

In [ ]:
```
if SAVE_MODEL:
  save_path = save_model(model = molecule_model , base_path = f'{models_path}/{model_type}_{ds_name}' )
```

Release GPU Cache to train other models if needed.

In [ ]:
```
def release_gpu_mem():
  """
```

```
  - Release GPU Memory
  """
  gc.collect()
  torch.cuda.empty_cache()

release_gpu_mem()
```

Next - Creating a CustomSSDI Model

We'll modify the SSIDDI Class to allow custom adjustments, like adding batch normalization to the GraphAttentionConv Layers and Subsampling Nodes/Edges to help reduce overfitting. Since this is a Google Colab environment, I'll just copy and paste the code from the ssiddi.py file and modify it in the cell below.

In [ ]:
```
"""An implementation of the SSI-DDI model."""

import torch
import torch.nn.functional
from torch import nn
from torch.nn import LayerNorm
from torch.nn.modules.container import ModuleList
from torchdrug.data import PackedGraph
from torchdrug.layers import GraphAttentionConv, AttentionReadout, MaxReadout,MeanReadout, NodeSampler, EdgeSampler

from chemicalx.constants import TORCHDRUG_NODE_FEATURES
from chemicalx.data import DrugPairBatch
from chemicalx.models import Model

__all__ = [
    "SSIDDI",
]



class EmbeddingLayer(torch.nn.Module):
    """Attention layer."""

    def __init__(self, feature_number: int):
        """Initialize the relational embedding layer.

        :param feature_number: Number of features.
        """
        super().__init__()
        self.weights = torch.nn.Parameter(torch.zeros(feature_number, feature_number))
        torch.nn.init.xavier_uniform_(self.weights)

    def forward(
        self,
        left_representations: torch.FloatTensor,
        right_representations: torch.FloatTensor,
        alpha_scores: torch.FloatTensor,
    ):
        """
        Make a forward pass with the drug representations.

        :param left_representations: Left side drug representations.
        :param right_representations: Right side drug representations.
        :param alpha_scores: Attention scores.
```

```
        :returns: Positive label scores vector.
        """
        attention = torch.nn.functional.normalize(self.weights, dim=-1)
        left_representations = torch.nn.functional.normalize(left_representations, dim=-1)
        right_representations = torch.nn.functional.normalize(right_representations, dim=-1)
        attention = attention.view(-1, self.weights.shape[0], self.weights.shape[1])
        scores = alpha_scores * (left_representations @ attention @ right_representations.transpose(-2, -1))
        scores = scores.sum(dim=(-2, -1)).view(-1, 1)
        return scores


class DrugDrugAttentionLayer(torch.nn.Module):
    """Co-attention layer for drug pairs."""

    def __init__(self, feature_number: int):
        """Initialize the co-attention layer.

        :param feature_number: Number of input features.
        """
        super().__init__()
        self.weight_query = torch.nn.Parameter(torch.zeros(feature_number, feature_number // 2))
        self.weight_key = torch.nn.Parameter(torch.zeros(feature_number, feature_number // 2))
        self.bias = torch.nn.Parameter(torch.zeros(feature_number // 2))
        self.attention = torch.nn.Parameter(torch.zeros(feature_number // 2))
        self.tanh = torch.nn.Tanh()

        torch.nn.init.xavier_uniform_(self.weight_query)
        torch.nn.init.xavier_uniform_(self.weight_key)
        torch.nn.init.xavier_uniform_(self.bias.view(*self.bias.shape, -1))
        torch.nn.init.xavier_uniform_(self.attention.view(*self.attention.shape, -1))

    def forward(self, left_representations: torch.Tensor, right_representations: torch.Tensor):
        """Make a forward pass with the co-attention calculation.

        :param left_representations: Matrix of left hand side representations.
        :param right_representations: Matrix of right hand side representations.
        :returns: Attention scores.
        """
        keys = left_representations @ self.weight_key
        queries = right_representations @ self.weight_query
        e_activations = queries.unsqueeze(-3) + keys.unsqueeze(-2) + self.bias
        attentions = self.tanh(e_activations) @ self.attention
        return attentions


class CustomSSIDDIBlock(torch.nn.Module):
    """SSIDDI Block with convolution and pooling."""

    def __init__(self,
                 head_number: int,
                 in_channels: int,
                 out_channels: int,
                 sampling: bool = False,
                 GATC_batchnorm : bool=False ,
                 readout=MeanReadout(type='node')):
        """Initialize an SSI-DDI Block.

        :param head_number: Number of attention heads.
        :param in_channels: Number of input channels.
```

```
        :param out_channels: Number of convolutional filters.
        """
        super().__init__()
        self.conv = GraphAttentionConv(input_dim=in_channels,
                                       output_dim=out_channels,
                                       num_head=head_number,
                                       batch_norm=GATC_batchnorm,
                                       activation = 'leaky_relu')

        self.readout = readout

        ## Used to prevent sampling of atoms/bonds  in validation data when training data using sampling
        self.sampling = sampling

    def forward(self, molecules: PackedGraph):
        """Make a forward pass.

        :param molecules: A batch of graphs.
        :returns: The molecules with updated atom states and the pooled representations.
        """


        molecules.atom_feature = self.conv(molecules, molecules.atom_feature)
        h_graphs = self.readout(molecules, molecules.atom_feature)
        return molecules, h_graphs


class CustomSSIDDI(Model):
    """An implementation of the SSI-DDI model from [nyamabo2021]_.
       Edited by Edwin Tembo 01/07/2024
    .. seealso:: This model was suggested in https://github.com/AstraZeneca/chemicalx/issues/11

    .. [nyamabo2021] Nyamabo, A. K., *et al.* (2021). `SSI-DDI: substructure-substructure interactions
       for drug-drug interaction prediction <https://doi.org/10.1093/bib/bbab133>`_.
       *Briefings in Bioinformatics*, 22(6).
    """

    def __init__(
        self,
        *,
        molecule_channels: int = TORCHDRUG_NODE_FEATURES,
        hidden_channels     =(32, 32),
        head_number         =(2, 2),
        use_logits          = False,
        GATC_batchnorm      = False,
        SUBSAMPLE_NODES     = False,
        NODE_SUBSAMPLE_RATIO= 1.0,
        SUBSAMPLE_EDGES     = False,
        EDGE_SUBSAMPLE_RATIO= 1.0,
        CONTEXT_COUNT       = None ,
        READOUT             = MeanReadout(type='node')

    ):
        """Instantiate the SSI-DDI model.

        :param molecule_channels: int   - The number of molecular features.
        :param hidden_channels  : tuple -  The list of neurons for each hidden layer block.
        :param head_number      : tuple -  The number of attention heads in each block.
        :param use_logits       : bool  -  Whether to return Logits or Probabilities
```

```
        :param GATC_batchnorm   : bool  - Whether to use batch norm in GATCONV layers
        :param SUBSAMPLE_NODES  : bool  - Whether drop some ATOMS to reduce Overfitting during training
        :param NODE_SUBSAMPLE_RATIO: float - Ratio of atoms to keep. Only applies if SUBSAMPLE_NODES=True.Must be between 0 and 1
        :param SUBSAMPLE_EDGES     : bool  - Whether to exclude a subset of edges/bonds during  Training.
        :param EDGE_SUBSAMPLE_RATIO: float - Ratio of edges/bonds to keep. Only applies if SUBSAMPLE_EDGES=True.Must be between 0 and 1
        :param READOUT  :  - The Pooling Readout Layer
        """
        super().__init__()
        self.initial_norm          = LayerNorm(molecule_channels)
        self.blocks                = ModuleList()
        self.net_norms             = ModuleList()
        self.use_logits            = use_logits

        self.GATC_batchnorm        = GATC_batchnorm
        self.SUBSAMPLE_NODES       = SUBSAMPLE_NODES
        self.NODE_SUBSAMPLE_RATIO  = NODE_SUBSAMPLE_RATIO
        self.SUBSAMPLE_EDGES       = SUBSAMPLE_EDGES
        self.EDGE_SUBSAMPLE_RATIO  = EDGE_SUBSAMPLE_RATIO
        self.CONTEXT_COUNT         = CONTEXT_COUNT


        self.sampling              =  SUBSAMPLE_NODES | SUBSAMPLE_EDGES
        self.readout               = READOUT

        channels                   = molecule_channels
```

```
    for hidden_channel, head_number in zip(hidden_channels, head_number):

        self.blocks.append(CustomSSIDDIBlock(head_number, channels, hidden_channel, self.sampling, self.GATC_batchnorm, readout=self.readout))
        self.net_norms.append(LayerNorm(hidden_channel))
        channels = hidden_channel

    self.co_attention = DrugDrugAttentionLayer(channels)
    self.relational_embedding = EmbeddingLayer(channels)
    if not self.use_logits:
      self.final = nn.Sigmoid()


def unpack(self, batch: DrugPairBatch):
    """Return the left molecular graph and right molecular graph."""
    return (
        batch.drug_molecules_left,
        batch.drug_molecules_right,
    )

def _forward_molecules(self, molecules: PackedGraph):

    if self.sampling:

        if self.SUBSAMPLE_NODES:
          node_sampler = NodeSampler( ratio=self.NODE_SUBSAMPLE_RATIO)
          molecules    = node_sampler(molecules)

        if self.SUBSAMPLE_EDGES:
```

```
            edge_sampler = EdgeSampler( ratio=self.EDGE_SUBSAMPLE_RATIO)
            molecules    = edge_sampler(molecules)

        molecules.atom_feature = self.initial_norm(molecules.atom_feature)
        representation = []

        for block, net_norm in zip(self.blocks, self.net_norms):
            molecules, pooled_hidden_left = block(molecules)
            representation.append(pooled_hidden_left)
            molecules.atom_feature = torch.nn.functional.elu(net_norm(molecules.atom_feature))
        return torch.stack(representation, dim=-2)

    def _combine_sides(self, features_left, features_right) -> torch.FloatTensor:
        attentions = self.co_attention(features_left, features_right)
        return self.relational_embedding(features_left, features_right, attentions)

    def forward(self, molecules_left: PackedGraph, molecules_right: PackedGraph, sampling: bool = False, context_value : float = None, ) -> torch.FloatTensor:
        """Run a forward pass of the SSI-DDI model.

        :param molecules_left: Batched molecules for the left side drugs.
        :param molecules_right: Batched molecules for the right side drugs.
        :param sampling: whether the use subsample of nodes and/or edges, must always be False for Validation
        :returns: A column vector of predicted synergy scores or relational embeddings.
        """

        self.sampling = sampling


        features_left  = self._forward_molecules(molecules_left)
        features_right = self._forward_molecules(molecules_right)
        hidden = self._combine_sides(features_left, features_right)

        if context_value is not None:
            ## to aviod overlap when  embeddings are negative, subtract when negative and add when positive

            hidden = torch.where(hidden > 0 , hidden + context_value , hidden - context_value)
            ## scale for efficiency
            hidden = torch.div(hidden, self.CONTEXT_COUNT)
        return hidden if self.use_logits else  self.final(hidden)
```

A train with molecules function with parameters for sampling

In [ ]:

```
import gc


def train_w_molecules_subs(model, train_gen, valid_gen, loss_fn, optim, epochs, subsampling =False, use_ctx=False):

    training_losses = []
    valid_losses    = []
    training_acc    = []
    valid_acc       = []
    for epoch in range(epochs):
        train_batch_counter = 0
        train_loss_sum      = 0

        valid_batch_counter  = 0
        valid_loss_sum       = 0
        train_epoch_acc      = []
        valid_epoch_acc      = []
```

```
    model.train()
    for batch in train_gen:
        optim.zero_grad()

        mols_left  = batch.drug_molecules_left.to(0)
        mols_right = batch.drug_molecules_right.to(0)
        labels     = batch.labels.to(0)

        ctx = None

        if use_ctx :
            # Using  a numeric eqivalent of the Context Value categorical variable as a . This can be a more complex data types like  protein chain embeddings
            # The application of this in the model must also be changed accordingly!
            # In this implementation, this is simply added to the final output embeddings and scaled the known number of categories in the dataset
            # to help differentiate the use of the same drug pair by context
            ctx=torch.argwhere(batch.context_features !=0. )[:, 1]

            # reshape to (batch_size, 1) and add one so very small numbers do not become 0 when divided
            ctx = ctx.type(torch.float32).reshape(batch.drug_features_left.data.shape[0], 1) + 1.0
            ctx =ctx.to(0)

        prediction = model(
                          mols_left,
                          mols_right,
                          subsampling,
                          context_value = ctx )

        loss_value = loss_fn(prediction,labels)
        ## Accuracy added December 2024

        acc_value  = torch.sum(torch.where(prediction >0.5, 1.0, 0.0 )==labels)/len(labels)
        loss_value.backward()
        optim.step()
        train_loss_sum+= loss_value.item()
        train_epoch_acc.append(acc_value.item())
        train_batch_counter += 1

    model.eval()
    for vbatch in valid_gen :

        vmols_left = vbatch.drug_molecules_left.to(0)
        vmols_right= vbatch.drug_molecules_right.to(0)
        vlabels    = vbatch.labels.to(0)
        vctx      = None
        if use_ctx :
            vctx=torch.argwhere(vbatch.context_features !=0. )[:, 1]
            ## reshape to (batch_size, 1) and add one so very small numbers do not become 0 when divided
            vctx = vctx.type(torch.float32).reshape(vbatch.drug_features_left.data.shape[0], 1) + 1.0
            vctx =vctx.to(0)

        prediction = model(
                          vmols_left,
                          vmols_right,
                          ###must always be False for validation or eval
                          sampling = False,
                          context_value = vctx )
        valid_loss_value = loss_fn(prediction, vlabels)
        ## Accuracy added December 2024
```

```
            val_acc_value  = torch.sum(torch.where(prediction >0.5, 1.0, 0.0 ) ==vlabels)/len(vlabels)
            valid_loss_sum+= valid_loss_value.item()
            valid_epoch_acc.append(val_acc_value.item())
            valid_batch_counter += 1

        train_epoch_loss = train_loss_sum/train_batch_counter
        valid_epoch_loss = valid_loss_sum/valid_batch_counter


        train_acc   = sum(train_epoch_acc)/len(train_epoch_acc)
        val_acc     = sum(valid_epoch_acc)/len(valid_epoch_acc)

        print("Epoch Loss: {a:4d} | Train Loss: {b:4.8f},  Valid Loss: {c:4.8f}, Train Acc: {d:4.8f},  Valid Acc: {e:4.8f}".
         format(a = epoch + 1, b = train_epoch_loss, c=valid_epoch_loss, d=train_acc, e=val_acc))
        training_losses.append(train_epoch_loss)
        valid_losses.append(valid_epoch_loss)
        training_acc.append(train_acc)
        valid_acc.append(val_acc)

    ##collect garbage
    gc.collect()
    return training_losses, valid_losses, training_acc, valid_acc
```

Handing the imbalance in DrugCombDB Dataset.

In this section, we'll use the BCEWithLogitsLoss function, which allows us to easily weight the positive training examples.

In [ ]:
```
## Data Loader class modified to generate and hold all required attributes for custom training.
class TrainingData(CustomRtDatasetLoader):
    def __init__(self,
                 ds_name,
                 batch_size         = 1024,
                 train_size         = 0.6,
                 split_random_state = 1024,
                 use_context_features   = False,
                 use_drug_features      = True,
                 use_drug_molecules     = True,
                 preprocessed_triples_data   = None

                 ):

        super().__init__(ds_name)
        self.ds_name               = ds_name
        self.batch_size            = batch_size

        ##self.drug_rds              = CustomRtDatasetLoader(ds_name)
        self.drug_features         = self.get_custom_drug_features()
        self.drug_ctx              = self.get_context_features()


        self.drug_labeled_triples  = self.get_labeled_triples()


        if preprocessed_triples_data is not None :
          self.drug_labeled_triples.data = preprocessed_triples_data

        self.pos_weight = torch.tensor(self.drug_labeled_triples.get_negative_count()/self.drug_labeled_triples.get_positive_count())
```

```
        self.train_size         = train_size
        self.split_random_state = split_random_state

        self.context_count  = len(triples.data['context'].value_counts())

        if train_size == 1.0:
          self.train_data = self.drug_labeled_triples
          self.test_data  = None
        else:
          self.train_data, self.test_data = self.drug_labeled_triples.train_test_split(train_size=self.train_size,
                                                                                       random_state=self.split_random_state
                                                                                       )


        self.use_context_features   = use_context_features,
        self.use_drug_features      = use_drug_features ,
        self.use_drug_molecules     = use_drug_molecules,

        self.train_generator = BatchGenerator(batch_size        = self.batch_size,
                                              context_features   = self.use_context_features,
                                              drug_features      = self.use_drug_features,
                                              drug_molecules     = self.use_drug_molecules,
                                              context_feature_set =self.drug_ctx,
                                              drug_feature_set    =self.drug_features,
                                              labeled_triples     =self.train_data)

        if self.train_size < 1.0 :

            self.valid_generator = BatchGenerator(batch_size         = self.batch_size,
                                                  context_features   = self.use_context_features,
                                                  drug_features      = self.use_drug_features,
                                                  drug_molecules     = self.use_drug_molecules,
                                                  context_feature_set =self.drug_ctx,
                                                  drug_feature_set    =self.drug_features,
                                                  labeled_triples     =self.test_data)

        else :

            self.valid_generator = None


        ##get the shape of channels
        t        = iter(self.train_generator)
        batch1   = next(t)
        self.channels = batch1.drug_molecules_left.data_dict['atom_feature'].shape[-1]
```

In [ ]:
```
release_gpu_mem()
```

In [ ]:
```
torch.cuda.manual_seed(23)
torch.manual_seed(23)
```

Out[ ]:
```
<torch._C.Generator at 0x78869c711250>
```

Important Note: The Batch Size used below will require up to 30+ GB of Peak GPU RAM on a single NVIDIA A100 GPU. Reduce the batch size accordingly to avoid an out of memory error in accordance to the available GPU RAM.

In [ ]:
```
training_data = TrainingData(  ds_name           ='drugcombdb',
                               batch_size        =3072,
```

```
                                train_size        =0.6,
                                split_random_state=1024,
                              )

print('pos_weight : ', training_data.pos_weight)

if training_data.pos_weight != 1.0 :
  loss                  = torch.nn.BCEWithLogitsLoss(pos_weight =training_data.pos_weight)
  use_logits            = True

else :
   loss                 = torch.nn.BCELoss()
   use_logits           = False
```

```
pos_weight :  tensor(2.4100)
```

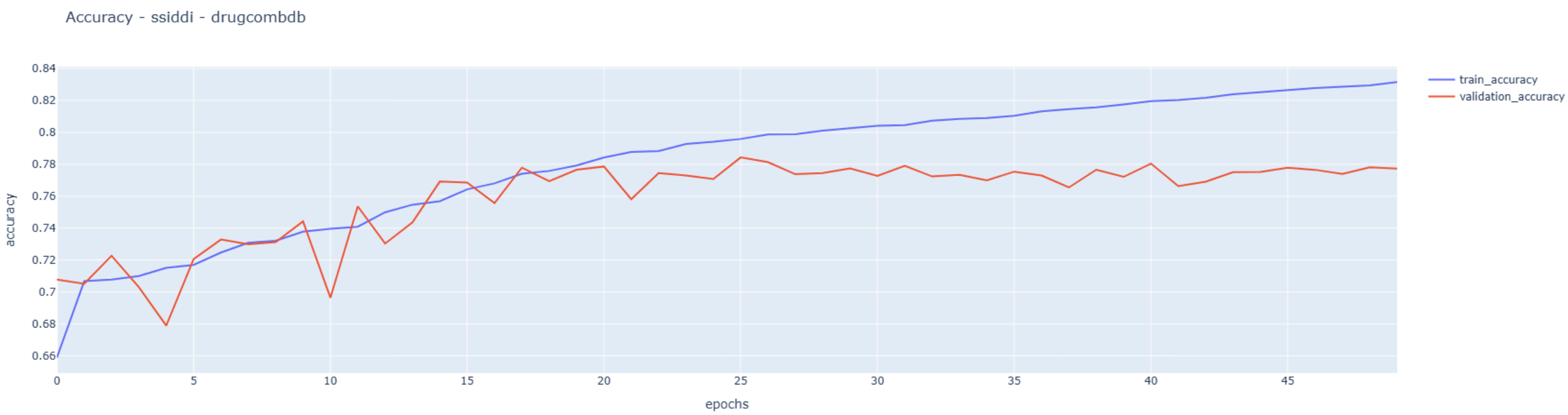
In [ ]:
```
molecule_model = CustomSSIDDI(  molecule_channels= training_data.channels,
                                hidden_channels  =(500, 500),
                                head_number      =(4, 4),
                                use_logits       =  use_logits
                              )

model_type = 'ssiddi'
epochs     = 50
optimizer  = torch.optim.Adam(molecule_model.parameters())

## to GPU 0
molecule_model.to(0)
```

Out[ ]:
```
CustomSSIDDI(
  (initial_norm): LayerNorm((199,), eps=1e-05, elementwise_affine=True)
  (blocks): ModuleList(
    (0): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=199, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
    (1): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=500, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
  )
  (net_norms): ModuleList(
    (0-1): 2 x LayerNorm((500,), eps=1e-05, elementwise_affine=True)
  )
  (readout): MeanReadout()
  (co_attention): DrugDrugAttentionLayer(
    (tanh): Tanh()
  )
  (relational_embedding): EmbeddingLayer()
)
```

In [ ]:
```
train_losses, valid_losses , training_acc, valid_acc= train_w_molecules_subs(molecule_model,
                                                                             training_data.train_generator,
                                                                             training_data.valid_generator,
                                                                             loss,
                                                                             optimizer,
```

```
                                                                epochs)

x      = [n for n in range(0, epochs)]
y_list = training_acc, valid_acc
series = ['train_accuracy', 'validation_accuracy']
title  = f"Accuracy - {model_type} - {training_data.ds_name}"
axis_titles = {'x' : 'epochs', 'y' : 'accuracy'}


plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```

```
Epoch Loss:    1 | Train Loss: 1.12042731,  Valid Loss: 0.96202660, Train Acc: 0.65876536,  Valid Acc: 0.70780126
Epoch Loss:    2 | Train Loss: 0.94769208,  Valid Loss: 0.93056080, Train Acc: 0.70672732,  Valid Acc: 0.70515244
Epoch Loss:    3 | Train Loss: 0.93008221,  Valid Loss: 0.91862594, Train Acc: 0.70770182,  Valid Acc: 0.72267042
Epoch Loss:    4 | Train Loss: 0.91784017,  Valid Loss: 0.90419794, Train Acc: 0.70997409,  Valid Acc: 0.70291949
Epoch Loss:    5 | Train Loss: 0.90313734,  Valid Loss: 0.90765352, Train Acc: 0.71509251,  Valid Acc: 0.67885011
Epoch Loss:    6 | Train Loss: 0.89688852,  Valid Loss: 0.87932213, Train Acc: 0.71694493,  Valid Acc: 0.72056696
Epoch Loss:    7 | Train Loss: 0.88277577,  Valid Loss: 0.87058378, Train Acc: 0.72471483,  Valid Acc: 0.73284241
Epoch Loss:    8 | Train Loss: 0.86977510,  Valid Loss: 0.86032867, Train Acc: 0.73076152,  Valid Acc: 0.72981886
Epoch Loss:    9 | Train Loss: 0.86633054,  Valid Loss: 0.85772916, Train Acc: 0.73205618,  Valid Acc: 0.73120442
Epoch Loss:   10 | Train Loss: 0.85432721,  Valid Loss: 0.84632723, Train Acc: 0.73775832,  Valid Acc: 0.74426656
Epoch Loss:   11 | Train Loss: 0.85121184,  Valid Loss: 0.88143007, Train Acc: 0.73962671,  Valid Acc: 0.69636410
Epoch Loss:   12 | Train Loss: 0.84169026,  Valid Loss: 0.84044162, Train Acc: 0.74085266,  Valid Acc: 0.75364795
Epoch Loss:   13 | Train Loss: 0.82618288,  Valid Loss: 0.83547733, Train Acc: 0.74991045,  Valid Acc: 0.73028689
Epoch Loss:   14 | Train Loss: 0.81307006,  Valid Loss: 0.81451752, Train Acc: 0.75465751,  Valid Acc: 0.74356264
Epoch Loss:   15 | Train Loss: 0.80381842,  Valid Loss: 0.80336992, Train Acc: 0.75686763,  Valid Acc: 0.76916959
Epoch Loss:   16 | Train Loss: 0.78361421,  Valid Loss: 0.78672454, Train Acc: 0.76427316,  Valid Acc: 0.76851112
Epoch Loss:   17 | Train Loss: 0.77367471,  Valid Loss: 0.78428735, Train Acc: 0.76801669,  Valid Acc: 0.75562975
Epoch Loss:   18 | Train Loss: 0.75606644,  Valid Loss: 0.76980190, Train Acc: 0.77406128,  Valid Acc: 0.77787502
Epoch Loss:   19 | Train Loss: 0.74556226,  Valid Loss: 0.76495410, Train Acc: 0.77581101,  Valid Acc: 0.76946100
Epoch Loss:   20 | Train Loss: 0.73478258,  Valid Loss: 0.75898892, Train Acc: 0.77923012,  Valid Acc: 0.77659003
Epoch Loss:   21 | Train Loss: 0.72429640,  Valid Loss: 0.75476602, Train Acc: 0.78422970,  Valid Acc: 0.77857295
Epoch Loss:   22 | Train Loss: 0.71195698,  Valid Loss: 0.76279271, Train Acc: 0.78772701,  Valid Acc: 0.75812193
Epoch Loss:   23 | Train Loss: 0.70647259,  Valid Loss: 0.75056422, Train Acc: 0.78821946,  Valid Acc: 0.77450445
Epoch Loss:   24 | Train Loss: 0.69802407,  Valid Loss: 0.75125237, Train Acc: 0.79269438,  Valid Acc: 0.77302232
Epoch Loss:   25 | Train Loss: 0.69129956,  Valid Loss: 0.75161469, Train Acc: 0.79407457,  Valid Acc: 0.77074767
Epoch Loss:   26 | Train Loss: 0.68369382,  Valid Loss: 0.76141719, Train Acc: 0.79584579,  Valid Acc: 0.78431330
Epoch Loss:   27 | Train Loss: 0.67631001,  Valid Loss: 0.74584181, Train Acc: 0.79871235,  Valid Acc: 0.78128384
Epoch Loss:   28 | Train Loss: 0.67239423,  Valid Loss: 0.75394609, Train Acc: 0.79878927,  Valid Acc: 0.77375667
Epoch Loss:   29 | Train Loss: 0.66797476,  Valid Loss: 0.74822812, Train Acc: 0.80103694,  Valid Acc: 0.77446355
Epoch Loss:   30 | Train Loss: 0.66120748,  Valid Loss: 0.75389619, Train Acc: 0.80274700,  Valid Acc: 0.77745947
Epoch Loss:   31 | Train Loss: 0.65673957,  Valid Loss: 0.75019333, Train Acc: 0.80410057,  Valid Acc: 0.77268122
Epoch Loss:   32 | Train Loss: 0.65290975,  Valid Loss: 0.75195918, Train Acc: 0.80451606,  Valid Acc: 0.77903499
Epoch Loss:   33 | Train Loss: 0.64473536,  Valid Loss: 0.76080823, Train Acc: 0.80728505,  Valid Acc: 0.77235091
Epoch Loss:   34 | Train Loss: 0.64276719,  Valid Loss: 0.76282764, Train Acc: 0.80842863,  Valid Acc: 0.77340552
Epoch Loss:   35 | Train Loss: 0.63767357,  Valid Loss: 0.76166387, Train Acc: 0.80893616,  Valid Acc: 0.76988094
Epoch Loss:   36 | Train Loss: 0.63196836,  Valid Loss: 0.77148091, Train Acc: 0.81038611,  Valid Acc: 0.77537564
Epoch Loss:   37 | Train Loss: 0.62838079,  Valid Loss: 0.75833407, Train Acc: 0.81320330,  Valid Acc: 0.77297359
Epoch Loss:   38 | Train Loss: 0.62416454,  Valid Loss: 0.77262477, Train Acc: 0.81456965,  Valid Acc: 0.76545713
Epoch Loss:   39 | Train Loss: 0.61810231,  Valid Loss: 0.77322999, Train Acc: 0.81569294,  Valid Acc: 0.77656398
Epoch Loss:   40 | Train Loss: 0.61377908,  Valid Loss: 0.77922781, Train Acc: 0.81746292,  Valid Acc: 0.77211309
Epoch Loss:   41 | Train Loss: 0.60870717,  Valid Loss: 0.78635917, Train Acc: 0.81961320,  Valid Acc: 0.78042406
Epoch Loss:   42 | Train Loss: 0.60612424,  Valid Loss: 0.78213279, Train Acc: 0.82026310,  Valid Acc: 0.76633064
Epoch Loss:   43 | Train Loss: 0.60269385,  Valid Loss: 0.78641949, Train Acc: 0.82159721,  Valid Acc: 0.76909154
Epoch Loss:   44 | Train Loss: 0.59599950,  Valid Loss: 0.79722607, Train Acc: 0.82389962,  Valid Acc: 0.77507258
Epoch Loss:   45 | Train Loss: 0.59134626,  Valid Loss: 0.80207665, Train Acc: 0.82534632,  Valid Acc: 0.77517187
Epoch Loss:   46 | Train Loss: 0.58716373,  Valid Loss: 0.80449889, Train Acc: 0.82647382,  Valid Acc: 0.77779465
Epoch Loss:   47 | Train Loss: 0.58333088,  Valid Loss: 0.80744041, Train Acc: 0.82783486,  Valid Acc: 0.77650711
Epoch Loss:   48 | Train Loss: 0.57994576,  Valid Loss: 0.80661489, Train Acc: 0.82867331,  Valid Acc: 0.77397803
Epoch Loss:   49 | Train Loss: 0.57874433,  Valid Loss: 0.82345260, Train Acc: 0.82942711,  Valid Acc: 0.77806777
Epoch Loss:   50 | Train Loss: 0.57273253,  Valid Loss: 0.83027282, Train Acc: 0.83152489,  Valid Acc: 0.77730953
```


Accuracy - ssiddi - drugcombdb
0.84
0.82
0.8
0.78
0.76
0.74
0.72
0.7
0.68
0.66
accuracy
0
5
10
15
20
25
30
35
40
45
epochs
train_accuracy
validation_accuracy

The chart above shows that validation accuracy plateau's at about 78%. This starts at about epoch 20, so 20 epochs will be the maximum number of epochs used in subsequent experiments.

In [ ]:
```
release_gpu_mem()
```

CustomSSIDDI Training - Edge Sampling = n% , 20 epochs

The next step will reduce the nodes ratio to n to try and reduce overfitting.

In [ ]:
```
print('Dataset : ' , training_data.ds_name, '\n')

molecule_model = CustomSSIDDI(  molecule_channels  = training_data.channels,
                                hidden_channels    =(500, 500),
                                head_number        =(4, 4),
                                use_logits         = use_logits,
                                GATC_batchnorm     = False,
                                SUBSAMPLE_EDGES    = True,
                                EDGE_SUBSAMPLE_RATIO= 0.6
                             )

model_type = 'ssiddi'
epochs     = 20
optimizer  = torch.optim.Adam(molecule_model.parameters())

## to GPU 0
molecule_model.to(0)
```

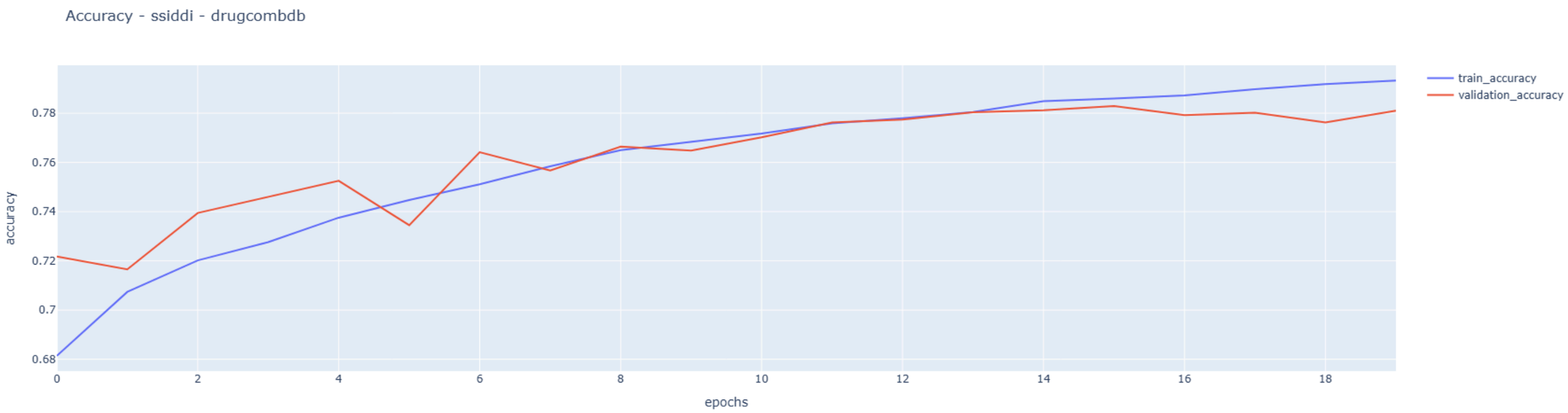
```
Dataset :  drugcombdb
```

Out[ ]:
```
CustomSSIDDI(
  (initial_norm): LayerNorm((199,), eps=1e-05, elementwise_affine=True)
  (blocks): ModuleList(
    (0): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=199, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
    (1): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=500, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
  )
  (net_norms): ModuleList(
    (0-1): 2 x LayerNorm((500,), eps=1e-05, elementwise_affine=True)
  )
  (readout): MeanReadout()
  (co_attention): DrugDrugAttentionLayer(
    (tanh): Tanh()
  )
  (relational_embedding): EmbeddingLayer()
)
```

In [ ]:
```
train_losses, valid_losses , training_acc, valid_acc= train_w_molecules_subs(molecule_model,
                                                                             training_data.train_generator,
                                                                             training_data.valid_generator,
```

```
                                                                    loss,
                                                                    optimizer,
                                                                    epochs,
                                                                    subsampling=True)



x      = [n for n in range(0, epochs)]
y_list = training_acc, valid_acc
series = ['train_accuracy', 'validation_accuracy']
title  = f"Accuracy - {model_type} - {training_data.ds_name}"
axis_titles = {'x' : 'epochs', 'y' : 'accuracy'}

plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```

```
Epoch Loss:    1 | Train Loss: 0.99489955,  Valid Loss: 0.92279305, Train Acc: 0.68141924,  Valid Acc: 0.72179129
Epoch Loss:    2 | Train Loss: 0.91905557,  Valid Loss: 0.89156194, Train Acc: 0.70737966,  Valid Acc: 0.71654043
Epoch Loss:    3 | Train Loss: 0.89337954,  Valid Loss: 0.88045507, Train Acc: 0.72017998,  Valid Acc: 0.73945854
Epoch Loss:    4 | Train Loss: 0.87933882,  Valid Loss: 0.86241369, Train Acc: 0.72762087,  Valid Acc: 0.74602249
Epoch Loss:    5 | Train Loss: 0.85921038,  Valid Loss: 0.85672853, Train Acc: 0.73755069,  Valid Acc: 0.75251584
Epoch Loss:    6 | Train Loss: 0.84366238,  Valid Loss: 0.83927401, Train Acc: 0.74471503,  Valid Acc: 0.73451353
Epoch Loss:    7 | Train Loss: 0.82421895,  Valid Loss: 0.81335353, Train Acc: 0.75111140,  Valid Acc: 0.76415358
Epoch Loss:    8 | Train Loss: 0.80660433,  Valid Loss: 0.80226759, Train Acc: 0.75839626,  Valid Acc: 0.75672910
Epoch Loss:    9 | Train Loss: 0.79038034,  Valid Loss: 0.78506574, Train Acc: 0.76498122,  Valid Acc: 0.76642904
Epoch Loss:   10 | Train Loss: 0.77946142,  Valid Loss: 0.78185214, Train Acc: 0.76827364,  Valid Acc: 0.76481647
Epoch Loss:   11 | Train Loss: 0.76761612,  Valid Loss: 0.76772360, Train Acc: 0.77178235,  Valid Acc: 0.77018049
Epoch Loss:   12 | Train Loss: 0.75378275,  Valid Loss: 0.76677019, Train Acc: 0.77585375,  Valid Acc: 0.77624213
Epoch Loss:   13 | Train Loss: 0.74765772,  Valid Loss: 0.75708311, Train Acc: 0.77795355,  Valid Acc: 0.77739837
Epoch Loss:   14 | Train Loss: 0.73860954,  Valid Loss: 0.75547181, Train Acc: 0.78051092,  Valid Acc: 0.78041805
Epoch Loss:   15 | Train Loss: 0.72874582,  Valid Loss: 0.75881681, Train Acc: 0.78487662,  Valid Acc: 0.78120017
Epoch Loss:   16 | Train Loss: 0.72043117,  Valid Loss: 0.75216115, Train Acc: 0.78594860,  Valid Acc: 0.78293772
Epoch Loss:   17 | Train Loss: 0.71465589,  Valid Loss: 0.74911947, Train Acc: 0.78725215,  Valid Acc: 0.77920798
Epoch Loss:   18 | Train Loss: 0.70732632,  Valid Loss: 0.75658917, Train Acc: 0.78972522,  Valid Acc: 0.78020393
Epoch Loss:   19 | Train Loss: 0.70041994,  Valid Loss: 0.74973206, Train Acc: 0.79183343,  Valid Acc: 0.77627396
Epoch Loss:   20 | Train Loss: 0.69728935,  Valid Loss: 0.75194210, Train Acc: 0.79326696,  Valid Acc: 0.78098315
```


Accuracy - ssiddi - drugcombdb
train_accuracy
validation_accuracy
accuracy
epochs
0.68
0.7
0.72
0.74
0.76
0.78
0
2
4
6
8
10
12
14
16
18

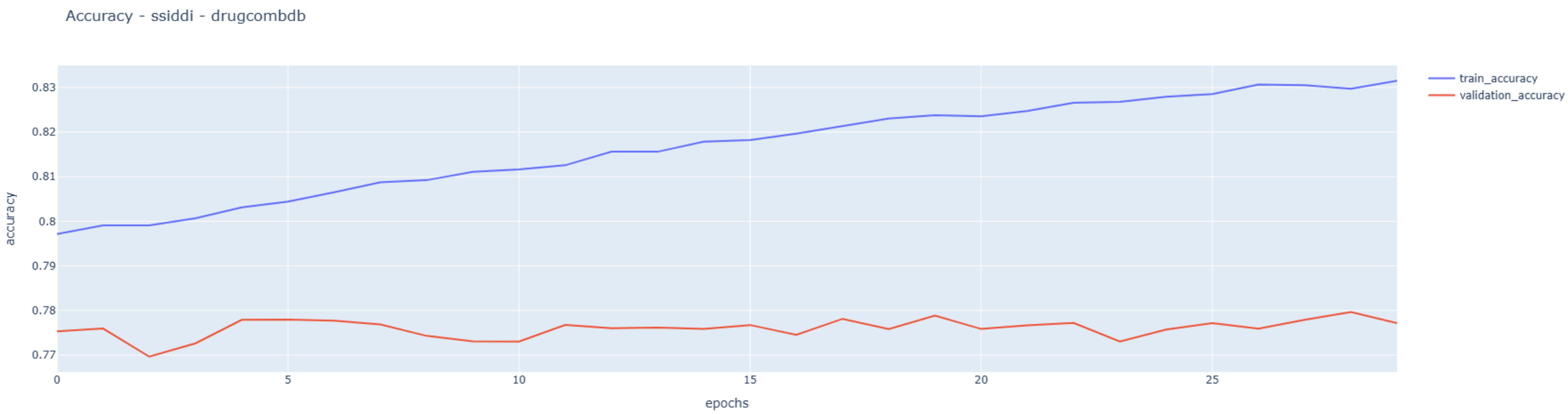
In [ ]:
```
release_gpu_mem()
```

Dropping 40% of the edges / bonds in each batch resulted in a much smoother validation curve, that more closely aligns with the training curve. However, validation accuracy remained around a maximum of 78%. Next, we'll examine the effect of increasing the number of epochs to 50 i.e. training for 30 more epochs.

CustomSSIDDI Training - Edge Sampling = n% , 30 more epochs

In [ ]:
```
epochs=30

train_losses, valid_losses , training_acc, valid_acc= train_w_molecules_subs(molecule_model,
                                                                             train_generator,
                                                                             valid_generator,
                                                                             loss,
                                                                             optimizer,
                                                                             epochs,
                                                                             subsampling=True)



x      = [n for n in range(0, epochs)]
y_list = training_acc, valid_acc
series = ['train_accuracy', 'validation_accuracy']
title  = f"Accuracy - {model_type} - {training_data.ds_name}"
axis_titles = {'x' : 'epochs', 'y' : 'accuracy'}

plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```

```
Epoch Loss:    1 | Train Loss: 0.68100299,  Valid Loss: 0.75706056, Train Acc: 0.79711284,  Valid Acc: 0.77530173
Epoch Loss:    2 | Train Loss: 0.67577710,  Valid Loss: 0.75524796, Train Acc: 0.79907531,  Valid Acc: 0.77594322
Epoch Loss:    3 | Train Loss: 0.67165016,  Valid Loss: 0.76364317, Train Acc: 0.79906608,  Valid Acc: 0.76963505
Epoch Loss:    4 | Train Loss: 0.66653681,  Valid Loss: 0.76374008, Train Acc: 0.80068228,  Valid Acc: 0.77263333
Epoch Loss:    5 | Train Loss: 0.65989362,  Valid Loss: 0.76595937, Train Acc: 0.80312031,  Valid Acc: 0.77789664
Epoch Loss:    6 | Train Loss: 0.65577183,  Valid Loss: 0.77649901, Train Acc: 0.80440740,  Valid Acc: 0.77793571
Epoch Loss:    7 | Train Loss: 0.65047278,  Valid Loss: 0.77395865, Train Acc: 0.80652520,  Valid Acc: 0.77768686
Epoch Loss:    8 | Train Loss: 0.64584453,  Valid Loss: 0.78503074, Train Acc: 0.80876399,  Valid Acc: 0.77686423
Epoch Loss:    9 | Train Loss: 0.64138406,  Valid Loss: 0.78855790, Train Acc: 0.80922143,  Valid Acc: 0.77429941
Epoch Loss:   10 | Train Loss: 0.63946361,  Valid Loss: 0.78850264, Train Acc: 0.81108908,  Valid Acc: 0.77307517
Epoch Loss:   11 | Train Loss: 0.63435147,  Valid Loss: 0.79704006, Train Acc: 0.81162324,  Valid Acc: 0.77301093
Epoch Loss:   12 | Train Loss: 0.63039803,  Valid Loss: 0.79202351, Train Acc: 0.81256570,  Valid Acc: 0.77674704
Epoch Loss:   13 | Train Loss: 0.62346097,  Valid Loss: 0.81465733, Train Acc: 0.81560845,  Valid Acc: 0.77600919
Epoch Loss:   14 | Train Loss: 0.62240684,  Valid Loss: 0.81477532, Train Acc: 0.81559933,  Valid Acc: 0.77617846
Epoch Loss:   15 | Train Loss: 0.61695283,  Valid Loss: 0.82337356, Train Acc: 0.81784617,  Valid Acc: 0.77586336
Epoch Loss:   16 | Train Loss: 0.61072816,  Valid Loss: 0.82390636, Train Acc: 0.81820974,  Valid Acc: 0.77673112
Epoch Loss:   17 | Train Loss: 0.60886136,  Valid Loss: 0.83770060, Train Acc: 0.81966955,  Valid Acc: 0.77451383
Epoch Loss:   18 | Train Loss: 0.60331334,  Valid Loss: 0.85136937, Train Acc: 0.82135583,  Valid Acc: 0.77811828
Epoch Loss:   19 | Train Loss: 0.60041857,  Valid Loss: 0.85363517, Train Acc: 0.82303264,  Valid Acc: 0.77582661
Epoch Loss:   20 | Train Loss: 0.59571019,  Valid Loss: 0.85802825, Train Acc: 0.82379197,  Valid Acc: 0.77885122
Epoch Loss:   21 | Train Loss: 0.59523672,  Valid Loss: 0.86519831, Train Acc: 0.82355780,  Valid Acc: 0.77584570
Epoch Loss:   22 | Train Loss: 0.59164837,  Valid Loss: 0.87122365, Train Acc: 0.82473432,  Valid Acc: 0.77664461
Epoch Loss:   23 | Train Loss: 0.58987520,  Valid Loss: 0.88113764, Train Acc: 0.82656563,  Valid Acc: 0.77721116
Epoch Loss:   24 | Train Loss: 0.58600268,  Valid Loss: 0.87115602, Train Acc: 0.82680027,  Valid Acc: 0.77304449
Epoch Loss:   25 | Train Loss: 0.58273576,  Valid Loss: 0.88052801, Train Acc: 0.82790563,  Valid Acc: 0.77569496
Epoch Loss:   26 | Train Loss: 0.57933924,  Valid Loss: 0.91747697, Train Acc: 0.82851106,  Valid Acc: 0.77714635
Epoch Loss:   27 | Train Loss: 0.57559852,  Valid Loss: 0.90386546, Train Acc: 0.83064578,  Valid Acc: 0.77593425
Epoch Loss:   28 | Train Loss: 0.57443211,  Valid Loss: 0.91634199, Train Acc: 0.83050715,  Valid Acc: 0.77789751
Epoch Loss:   29 | Train Loss: 0.57514748,  Valid Loss: 0.91574678, Train Acc: 0.82973860,  Valid Acc: 0.77966256
Epoch Loss:   30 | Train Loss: 0.57065254,  Valid Loss: 0.91217258, Train Acc: 0.83151758,  Valid Acc: 0.77715647
```


Accuracy - ssiddi - drugcombdb
accuracy
epochs
train_accuracy
validation_accuracy
0.83
0.82
0.81
0.8
0.79
0.78
0.77
0
5
10
15
20
25

Adding 30 more epochs proved inconsequential. The validation accuracy still plateaued just below 78%.

NEXT - Outlier Detection

This process will try to improve the models by finding outliers in the modelcular structures. By removing outliers, the goal is for the model to learn the most infulential patterns , thereby increasing validation accuracy.

First, we will create embeddings using the combined molecular structres of each drug pair, with an optional context feature included in computation of these embeddings.

In [ ]:
```
release_gpu_mem()
```

To remove outliers we will make the train size 100% (the , then run the model though one epoch that returns the output of the relational embedding layer. These embeddings are then used to find outliers.

In [ ]:
```
training_data = TrainingData(  ds_name           ='drugcombdb',
                               batch_size        =1024,
                               train_size        =1.0,
                               split_random_state=1024,
                               use_context_features   = True,
                               use_drug_features      = True,
                               use_drug_molecules     = True,

                            )
print('Dataset : ' , training_data.ds_name, '\n')
print('pos_weight : ', training_data.pos_weight)

if training_data.pos_weight != 1.0 :
  loss                 = torch.nn.BCEWithLogitsLoss(pos_weight =training_data.pos_weight)
  use_logits           = True

else :
   loss                = torch.nn.BCELoss()
   use_logits          = False


molecule_model = CustomSSIDDI(  molecule_channels= training_data.channels,
                                hidden_channels  =(500, 500),
                                head_number      =(4, 4),
                                use_logits       = True,
                                GATC_batchnorm   = False,
                                SUBSAMPLE_EDGES    = True,
                                EDGE_SUBSAMPLE_RATIO= 0.6,
                                CONTEXT_COUNT    = training_data.context_count,
                                READOUT          = MeanReadout(type='node')
                             )



model_type = 'ssiddi'
epochs     = 20
optimizer  = torch.optim.Adam(molecule_model.parameters())

## to GPU 0
device = 0
molecule_model.to(device)
```

```
Dataset :  drugcombdb

pos_weight :  tensor(2.4100)
```

Out[ ]:
```
CustomSSIDDI(
  (initial_norm): LayerNorm((199,), eps=1e-05, elementwise_affine=True)
  (blocks): ModuleList(
    (0): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=199, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
    (1): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=500, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
  )
  (net_norms): ModuleList(
    (0-1): 2 x LayerNorm((500,), eps=1e-05, elementwise_affine=True)
  )
  (readout): MeanReadout()
  (co_attention): DrugDrugAttentionLayer(
    (tanh): Tanh()
  )
  (relational_embedding): EmbeddingLayer()
 )
```

In [ ]:
```python
def embed_data_df(model, datagen, device =0, use_ctx=False):
"""
Function to compute relational embeddings based on the molecular structure input of two drugs
and an optional context computation/feature

Returns: Pandas DataFrame
"""
df_embeddings = pd.DataFrame(columns = ['drug_1', 'drug_2','context', 'embeddings' ])

for batch in datagen:

  batch_df = batch.identifiers[['drug_1', 'drug_2', 'context']]
  batch_df['embeddings'] = np.nan

  df_embeddings= pd.concat((df_embeddings,batch_df) , axis= 0)
  mols_left  = batch.drug_molecules_left.to(device)
  mols_right = batch.drug_molecules_right.to(device)
  labels     = batch.labels.to(device)
  embeddings = None
  ctx = None
  if use_ctx :
            # Using  The categorical Context Value. This can be a more complex data type like a protein chain embeddings
            # The application of this in the mode must also be changed accordingly!
            # In this implementation, this is simply added to the final output embeddings and scaled to min/max using the known number of categories in the dataset
            # to help differentiate the use of the same drug pair by context
            ctx=torch.argwhere(batch.context_features !=0. )[:, 1]

            # reshape to (batch_size, 1) and add one so very small numbers do not become 0 when divided
            ctx = ctx.type(torch.float32).reshape(batch.drug_features_left.data.shape[0], 1) + 1.0
            ctx = ctx.to(device)
```

```
    with torch.no_grad():
        embeddings = model(
                           mols_left,
                           mols_right,
                           False,
                           context_value =ctx
                          )
        embeddings = embeddings.cpu()
        df_embeddings.loc[df_embeddings['embeddings'].isnull(), ['embeddings']] = embeddings.flatten()

    batch_df   = None
    embeddings = None

  return df_embeddings
```

In [ ]:
```
device = torch.device("cuda")
device
```

Out[ ]:
```
device(type='cuda')
```

In [ ]:
```
embeddings =[]
molecule_model.eval()  # eval mode disables training-time operators (like batch normalization)
# Generate feature embeddings
train_feature_embeddings = embed_data_df(molecule_model, training_data.train_generator, use_ctx=True)
print(f'Train embeddings  shape: {train_feature_embeddings.shape}')

gc.collect()
```

```
train_feature_embeddings
```

```
Train embeddings  shape: (191391, 4)
```

Out[ ]:

	drug_1	drug_2	context	embeddings
77771	2141	545	PC-3	0.200132
72576	163659	387447	IGROV1	-0.400658
174188	60198	54611422	NCI-ADR-RES	-0.520985
47057	6013	10090750	KBM-7	-0.441275
292	24958200	46926350	A2058	-0.560601
...	...	...	...	...
123090	3081361	123608	SF-268	-0.540188
30487	5920	19493	KBM-7	-0.440458
40821	9444	115355	KBM-7	-0.440372
95866	3657	23615975	HCT116	-0.710146
145953	24776445	163659	SK-OV-3	-0.101121

191391 rows × 4 columns

Cleanlab will be used for the initial outlier detection, using the OutOfDistribution class.

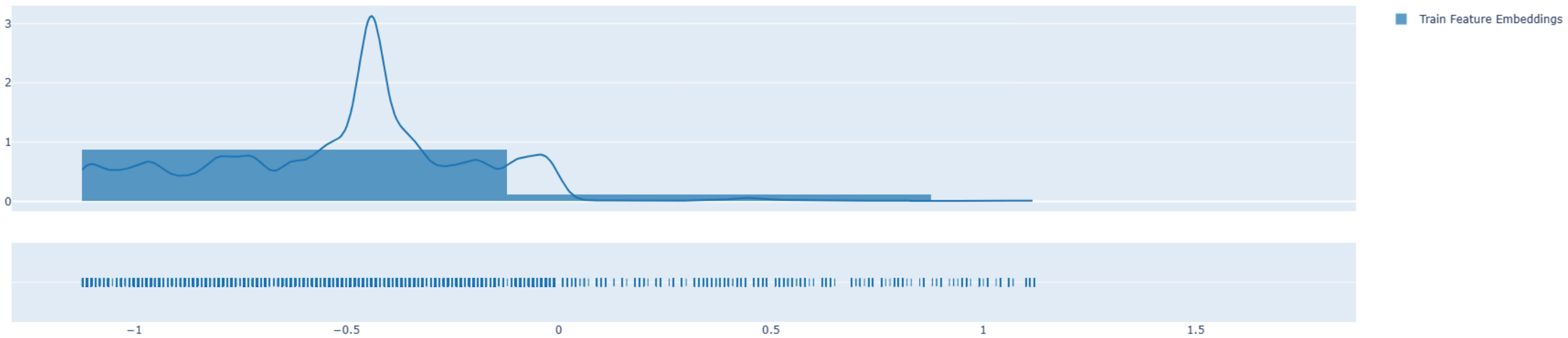
In [ ]:
```
!pip install  cleanlab
```

Distribution of Embeddings

Since this embeds the data to 1 dimension, it is easy to visualize the distribution.

In [ ]:
```
hist_data    = [list(train_feature_embeddings['embeddings'])]
group_labels = ['Train Feature Embeddings']

# Create distplot with custom bin_size
fig = ff.create_distplot(hist_data, group_labels)
fig.show()
```

Output hidden; open in https://colab.research.google.com to view.



In [ ]:
```
from cleanlab.outlier import OutOfDistribution
from cleanlab.rank import find_top_issues
import numpy as np
import pandas as pd
```

Well start by excluding the Top n% Out Of Distribution Values.

In [ ]:
```
embs       = train_feature_embeddings['embeddings'].to_numpy().reshape((len(train_feature_embeddings), 1))
embs
```

Out[ ]:
```
array([[0.20013155043125153],
       [-0.4006577134132385],
       [-0.5209851861000061],
       ...,
       [-0.44037187099456787],
       [-0.7101461887359619],
       [-0.1011207103729248]], dtype=object)
```

In [ ]:
```
ood = OutOfDistribution()
ood_train_feature_scores = ood.fit_score(features=embs)
ood_train_feature_scores
```

```
Fitting OOD estimator based on provided features ...
```

Out[ ]:
```
array([3.40129505e-07, 3.71078373e-01, 9.90938090e-02, ...,
       1.00000000e+00, 4.31785285e-01, 2.00351094e-01])
```

In [ ]:
```
top_idxs = []

train_ood_features_scores = ood.fit_score(features=embs)
top_n = int(0.025 * embs.shape[0] )
top_train_ood_features_idxs = find_top_issues(quality_scores=train_ood_features_scores, top=top_n)
top_idxs.append(top_train_ood_features_idxs)

top_idxs[0].shape
```

Out[ ]:
```
(4784,)
```

In [ ]:
```
assert training_data.train_generator.sample_count == len(train_feature_embeddings['embeddings']) , 'Embedding Samples Length Must Equal Generator Samples Length'
```

Validating that the Out-of-Distribution Index Matches the location in the original training set.

In [ ]:
```
release_gpu_mem()
```

In [ ]:
```
## A random index value in the top_idex

random_idx = np.random.randint(0, len(top_idxs[0]) -1,1)
#print('Random Idx :', random_idx[0])
print(train_feature_embeddings.iloc[top_idxs[0][random_idx]])
print(training_data.train_generator.labeled_triples.data.iloc[top_idxs[0][random_idx]])
```

```
       drug_1 drug_2   context embeddings
70410  444795    237  HCC-2998  -0.641567
       drug_1 drug_2   context  label
70410  444795    237  HCC-2998    0.0
```

Removing outliers

Running the code above a few times has confirmed that the order is the train_generator has been preserved. The next step will remove outliers.

In [ ]:
```
td = training_data.train_generator.labeled_triples.data
filter = td.iloc[top_idxs[0]]
training_data.train_generator.labeled_triples.data = td[~td.index.isin(filter.index)]
```

In [ ]:
```
training_data.train_generator.labeled_triples.data.head()
```

Out[ ]:

	drug_1	drug_2	context	label
72576	163659	387447	IGROV1	1.0
174188	60198	54611422	NCI-ADR-RES	0.0
47057	6013	10090750	KBM-7	1.0
292	24958200	46926350	A2058	1.0
33866	4197	656583	KBM-7	0.0

In [ ]:
```
len(training_data.train_generator.labeled_triples.data)
```

Out[ ]: 186607

Spliting the data back to a train/test split of 0.6/0.4. Since we already have a reliable way to split and generate data, we will just change the presplit triples to the cleaned training data labled triples then regenerate the train/valid generators.

In [ ]:
```
training_data = TrainingData(  ds_name           ='drugcombdb',
                               batch_size        =1024,
                               train_size        =0.6,
                               split_random_state=1024,
                               use_context_features     = True,
                               use_drug_features        = True,
                               use_drug_molecules       = True,
                               preprocessed_triples_data= training_data.train_generator.labeled_triples.data

                            )
```

In [ ]:
```
## confirm the training data

train_len = len(training_data.train_generator.labeled_triples.data)

valid_len = len(training_data.valid_generator.labeled_triples.data)

train_len + valid_len
```

Out[ ]: 186607

Training After Removing Outliers

In [ ]:
```
release_gpu_mem()
```

In [ ]:
```
if training_data.pos_weight != 1.0 :
  loss                  = torch.nn.BCEWithLogitsLoss(pos_weight =training_data.pos_weight)
  use_logits            = True

else :
   loss                 = torch.nn.BCELoss()
   use_logits           = False

molecule_model = CustomSSIDDI(  molecule_channels= training_data.channels,
                                hidden_channels  =(500, 500),
                                head_number      =(4, 4),
                                use_logits       = True,
```

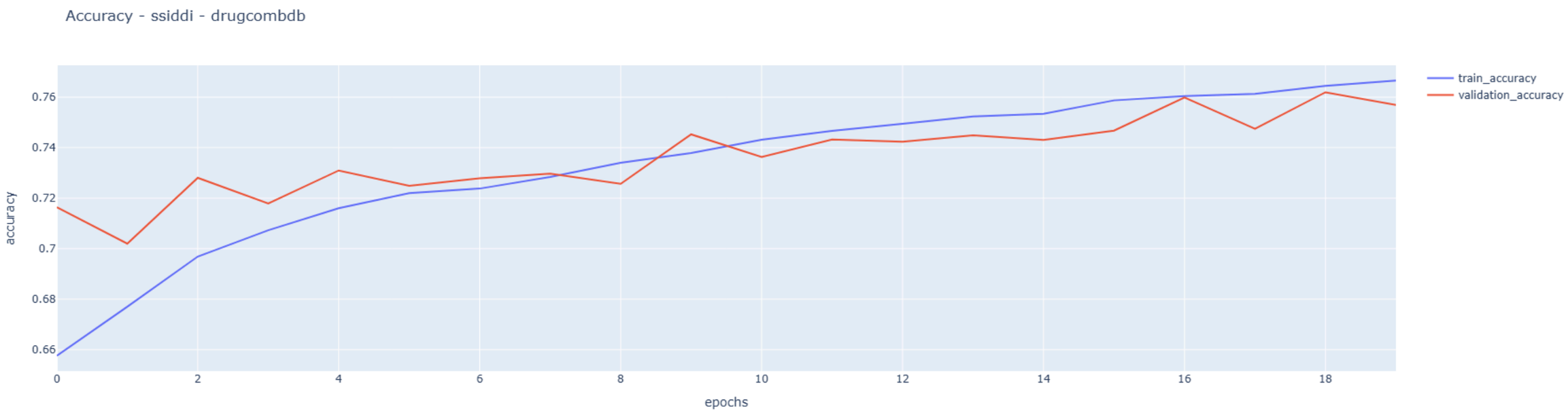
```
                                GATC_batchnorm   = False,
                                SUBSAMPLE_EDGES    = True,
                                EDGE_SUBSAMPLE_RATIO= 0.6,
                                CONTEXT_COUNT    = training_data.context_count,
                                READOUT          = MeanReadout(type='node')
                              )



model_type = 'ssiddi'
epochs     = 20
optimizer  = torch.optim.Adam(molecule_model.parameters())

## to GPU 0
device = 0
molecule_model.to(device)
```

Out[ ]:
```
CustomSSIDDI(
  (initial_norm): LayerNorm((199,), eps=1e-05, elementwise_affine=True)
  (blocks): ModuleList(
    (0): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=199, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
    (1): CustomSSIDDIBlock(
      (conv): GraphAttentionConv(
        (linear): Linear(in_features=500, out_features=500, bias=True)
      )
      (readout): MeanReadout()
    )
  )
  (net_norms): ModuleList(
    (0-1): 2 x LayerNorm((500,), eps=1e-05, elementwise_affine=True)
  )
  (readout): MeanReadout()
  (co_attention): DrugDrugAttentionLayer(
    (tanh): Tanh()
  )
  (relational_embedding): EmbeddingLayer()
)
```

In [ ]:
```
epochs = 20

train_losses, valid_losses , training_acc, valid_acc= train_w_molecules_subs(molecule_model,
                                                                             training_data.train_generator,
                                                                             training_data.valid_generator,
                                                                             loss,
                                                                             optimizer,
                                                                             epochs,
                                                                             subsampling=True,
                                                                             use_ctx = True)



x      = [n for n in range(0, epochs)]
y_list = training_acc, valid_acc
series = ['train_accuracy', 'validation_accuracy']
title  = f"Accuracy - {model_type} - {training_data.ds_name}"
```

```
axis_titles = {'x' : 'epochs', 'y' : 'accuracy'}

plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```

```
Epoch Loss:    1 | Train Loss: 0.98864261,  Valid Loss: 0.97212414, Train Acc: 0.65757535,  Valid Acc: 0.71639210
Epoch Loss:    2 | Train Loss: 0.93498214,  Valid Loss: 0.91028191, Train Acc: 0.67703486,  Valid Acc: 0.70199876
Epoch Loss:    3 | Train Loss: 0.90446544,  Valid Loss: 0.89237209, Train Acc: 0.69688316,  Valid Acc: 0.72805512
Epoch Loss:    4 | Train Loss: 0.88893651,  Valid Loss: 0.87451624, Train Acc: 0.70727605,  Valid Acc: 0.71785948
Epoch Loss:    5 | Train Loss: 0.86914509,  Valid Loss: 0.86241503, Train Acc: 0.71598593,  Valid Acc: 0.73095486
Epoch Loss:    6 | Train Loss: 0.85339351,  Valid Loss: 0.85161322, Train Acc: 0.72199245,  Valid Acc: 0.72492414
Epoch Loss:    7 | Train Loss: 0.84397433,  Valid Loss: 0.83598150, Train Acc: 0.72378782,  Valid Acc: 0.72787865
Epoch Loss:    8 | Train Loss: 0.83093182,  Valid Loss: 0.82893552, Train Acc: 0.72838756,  Valid Acc: 0.72968143
Epoch Loss:    9 | Train Loss: 0.81763691,  Valid Loss: 0.82395663, Train Acc: 0.73401938,  Valid Acc: 0.72566565
Epoch Loss:   10 | Train Loss: 0.80822703,  Valid Loss: 0.81207373, Train Acc: 0.73787797,  Valid Acc: 0.74525838
Epoch Loss:   11 | Train Loss: 0.79614220,  Valid Loss: 0.80683096, Train Acc: 0.74311233,  Valid Acc: 0.73629603
Epoch Loss:   12 | Train Loss: 0.78520108,  Valid Loss: 0.79779727, Train Acc: 0.74660560,  Valid Acc: 0.74324377
Epoch Loss:   13 | Train Loss: 0.77913149,  Valid Loss: 0.79084880, Train Acc: 0.74948866,  Valid Acc: 0.74236211
Epoch Loss:   14 | Train Loss: 0.77050730,  Valid Loss: 0.78742006, Train Acc: 0.75232018,  Valid Acc: 0.74485831
Epoch Loss:   15 | Train Loss: 0.76531940,  Valid Loss: 0.78603697, Train Acc: 0.75338195,  Valid Acc: 0.74304345
Epoch Loss:   16 | Train Loss: 0.75645582,  Valid Loss: 0.78670622, Train Acc: 0.75871243,  Valid Acc: 0.74674039
Epoch Loss:   17 | Train Loss: 0.75109970,  Valid Loss: 0.78322227, Train Acc: 0.76045555,  Valid Acc: 0.75987528
Epoch Loss:   18 | Train Loss: 0.74574793,  Valid Loss: 0.78107252, Train Acc: 0.76132915,  Valid Acc: 0.74749402
Epoch Loss:   19 | Train Loss: 0.74126522,  Valid Loss: 0.77901817, Train Acc: 0.76442016,  Valid Acc: 0.76193605
Epoch Loss:   20 | Train Loss: 0.73645284,  Valid Loss: 0.77899456, Train Acc: 0.76656421,  Valid Acc: 0.75687073
```


Accuracy - ssiddi - drugcombdb
train_accuracy
validation_accuracy
accuracy
epochs
0.66
0.68
0.7
0.72
0.74
0.76
0
2
4
6
8
10
12
14
16
18

Training for an additional 30 epochs

In [ ]:
```
epochs = 30

train_losses, valid_losses , training_acc, valid_acc= train_w_molecules_subs(molecule_model,
                                                                             training_data.train_generator,
                                                                             training_data.valid_generator,
                                                                             loss,
                                                                             optimizer,
                                                                             epochs,
                                                                             subsampling=True,
                                                                             use_ctx = True)



x      = [n for n in range(0, epochs)]
y_list = training_acc, valid_acc
series = ['train_accuracy', 'validation_accuracy']
title  = f"Accuracy - {model_type} - {training_data.ds_name}"
axis_titles = {'x' : 'epochs', 'y' : 'accuracy'}

plot_multiple_lines(x    ,
                    y_list ,
                    series ,
                    title,
                    axis_titles
                    )
```

```
Epoch Loss:    1 | Train Loss: 0.73239028,  Valid Loss: 0.77657787, Train Acc: 0.76732750,  Valid Acc: 0.74675348
Epoch Loss:    2 | Train Loss: 0.72750152,  Valid Loss: 0.77355930, Train Acc: 0.76835784,  Valid Acc: 0.75113432
Epoch Loss:    3 | Train Loss: 0.72507157,  Valid Loss: 0.77762395, Train Acc: 0.77045097,  Valid Acc: 0.75380886
Epoch Loss:    4 | Train Loss: 0.71760781,  Valid Loss: 0.77903379, Train Acc: 0.77209868,  Valid Acc: 0.75443379
Epoch Loss:    5 | Train Loss: 0.71710667,  Valid Loss: 0.77389474, Train Acc: 0.77126018,  Valid Acc: 0.75248004
Epoch Loss:    6 | Train Loss: 0.71475361,  Valid Loss: 0.77200514, Train Acc: 0.77297555,  Valid Acc: 0.75866014
Epoch Loss:    7 | Train Loss: 0.71063956,  Valid Loss: 0.77509033, Train Acc: 0.77360587,  Valid Acc: 0.75788487
Epoch Loss:    8 | Train Loss: 0.70734463,  Valid Loss: 0.77558210, Train Acc: 0.77534899,  Valid Acc: 0.75825118
Epoch Loss:    9 | Train Loss: 0.70378817,  Valid Loss: 0.77566290, Train Acc: 0.77650209,  Valid Acc: 0.75711124
Epoch Loss:   10 | Train Loss: 0.69933341,  Valid Loss: 0.77736609, Train Acc: 0.77783019,  Valid Acc: 0.75389995
Epoch Loss:   11 | Train Loss: 0.70019081,  Valid Loss: 0.77248456, Train Acc: 0.77790509,  Valid Acc: 0.75053966
Epoch Loss:   12 | Train Loss: 0.69537860,  Valid Loss: 0.77997290, Train Acc: 0.77785713,  Valid Acc: 0.75710646
Epoch Loss:   13 | Train Loss: 0.69422614,  Valid Loss: 0.77699963, Train Acc: 0.78071580,  Valid Acc: 0.74285389
Epoch Loss:   14 | Train Loss: 0.69143222,  Valid Loss: 0.77870869, Train Acc: 0.77903534,  Valid Acc: 0.75856268
Epoch Loss:   15 | Train Loss: 0.68944052,  Valid Loss: 0.77707785, Train Acc: 0.78150491,  Valid Acc: 0.74742554
Epoch Loss:   16 | Train Loss: 0.68706532,  Valid Loss: 0.77786105, Train Acc: 0.78016282,  Valid Acc: 0.75141206
Epoch Loss:   17 | Train Loss: 0.68462285,  Valid Loss: 0.77515439, Train Acc: 0.78075539,  Valid Acc: 0.75110437
Epoch Loss:   18 | Train Loss: 0.68038076,  Valid Loss: 0.77845218, Train Acc: 0.78413173,  Valid Acc: 0.74900410
Epoch Loss:   19 | Train Loss: 0.67939495,  Valid Loss: 0.78031144, Train Acc: 0.78288199,  Valid Acc: 0.75036286
Epoch Loss:   20 | Train Loss: 0.67749010,  Valid Loss: 0.78311333, Train Acc: 0.78385855,  Valid Acc: 0.75386397
Epoch Loss:   21 | Train Loss: 0.67366149,  Valid Loss: 0.78470827, Train Acc: 0.78527493,  Valid Acc: 0.75969852
Epoch Loss:   22 | Train Loss: 0.67285100,  Valid Loss: 0.78446101, Train Acc: 0.78589280,  Valid Acc: 0.75926025
Epoch Loss:   23 | Train Loss: 0.66822092,  Valid Loss: 0.78526478, Train Acc: 0.78737867,  Valid Acc: 0.74583168
Epoch Loss:   24 | Train Loss: 0.66962925,  Valid Loss: 0.78383007, Train Acc: 0.78647905,  Valid Acc: 0.74984466
Epoch Loss:   25 | Train Loss: 0.66802396,  Valid Loss: 0.78460065, Train Acc: 0.78734428,  Valid Acc: 0.74834188
Epoch Loss:   26 | Train Loss: 0.66396534,  Valid Loss: 0.78701638, Train Acc: 0.78785583,  Valid Acc: 0.75775399
Epoch Loss:   27 | Train Loss: 0.66007336,  Valid Loss: 0.79189390, Train Acc: 0.79097583,  Valid Acc: 0.73997008
Epoch Loss:   28 | Train Loss: 0.66131608,  Valid Loss: 0.78866710, Train Acc: 0.78963344,  Valid Acc: 0.74930062
Epoch Loss:   29 | Train Loss: 0.65970367,  Valid Loss: 0.79493414, Train Acc: 0.79049663,  Valid Acc: 0.74994115
Epoch Loss:   30 | Train Loss: 0.65739403,  Valid Loss: 0.79902763, Train Acc: 0.79032744,  Valid Acc: 0.75055589
```


Accuracy - ssiddi - drugcombdb
accuracy
epochs
0.79
0.78
0.77
0.76
0.75
0.74
0
5
10
15
20
25
train_accuracy
validation_accuracy

Validation accuracy went down and plateaued around 75%, less performant than the metric without removing outliers. Next , we will train on the same data, but exclude the context computation.

Links to YouTube Videos:

Intro: https://youtu.be/x7ii7IJ-1MU

Full : https://youtu.be/XihDBVrsj4A

References

Using grep to replace a string: https://stackoverflow.com/questions/15402770/how-to-search-and-replace-using-grep

Find and replace text within a file using commands: https://askubuntu.com/questions/20414/find-and-replace-text-within-a-file-using-commands

Visualizing molecules built from SMILES strings in Jupyter Notebooks using molSimplify: https://hjkgrp.mit.edu/tutorials/2021-11-03-visualizing-molecules-built-smiles-strings-jupyter-notebooks-using-molsimplify/

Hashed Morgan Fingerprint To Numpy Array: https://stackoverflow.com/questions/54809506/how-can-i-compute-a-count-morgan-fingerprint-as-numpy-array