# DeepPurpose

***Release 0.0.1***

**Sep 23, 2022**

# Background

Welcome! This is the documentation for DeepPurpose. DeepPurpose is a Deep Learning Based Drug Repurposing and Virtual Screening Toolkit (using PyTorch). It allows very easy usage (only one line of code!) for non-computational domain researchers to be able to obtain a list of potential drugs using deep learning while facilitating deep learning method research in this topic by providing a flexible framework (less than 10 lines of codes!) and baselines. The Github repository is located here.

CHAPTER 1

# Features of DeepPurpose

DeepPurpose is a Deep Learning Based Drug Repurposing and Virtual Screening Toolkit (using PyTorch). It allows very easy usage (only one line of code!) for non-computational domain researchers to be able to obtain a list of potential drugs using deep learning while facilitating deep learning method research in this topic by providing a flexible framework (less than 10 lines of codes!) and baselines. The Github repository is located here.

## 1.1 Features

- For non-computational researchers, ONE line of code from raw data to output drug repurposing/virtual screening result, aiming to allow wet-lab biochemists to leverage the power of deep learning. The result is ensembled from five pretrained deep learning models!
- For computational researchers, 15+ powerful encodings for drugs and proteins, ranging from deep neural network on classic cheminformatics fingerprints, CNN, transformers to message passing graph neural network, with 50+ combined models! Most of the combinations of the encodings are not yet in existing works. All of these under 10 lines but with lots of flexibility! Switching encoding is as simple as changing the encoding names!
- Realistic and user-friendly design:
  - automatic identification to do drug target binding affinity (regression) or drug target interaction prediction (binary) task.
  - support cold target, cold drug settings for robust model evaluations and support single-target high throughput sequencing assay data setup.
  - many dataset loading/downloading/unzipping scripts to ease the tedious preprocessing, including antiviral, COVID19 targets, BindingDB, DAVIS, KIBA, ...
  - many pretrained checkpoints.
  - easy monitoring of training process with detailed training metrics output such as test set figures (AUCs) and tables, also support early stopping.
  - detailed output records such as rank list for repurposing result.

- various evaluation metrics: ROC-AUC, PR-AUC, F1 for binary task, MSE, R-squared, Concordance Index for regression task.
- label unit conversion for skewed label distribution such as Kd.
- time reference for computational expensive encoding.
- PyTorch based, support CPU, GPU, Multi-GPUs.

# CHAPTER 2

# What is drug repurposing, virtual screening and drug-target interaction prediction?

## 2.1 Drug Repurposing

Drug repurposing aims to repivot an existing drug to a new therapy.

## 2.2 Virtual Screening

**Virtual screening means to use computer software to automatically screen** a huge space of potential drug-target pairs to obtain a predicted binding score.

## 2.3 Drug-Target Interaction

Both of these tasks are able to save cost, time, and facilitate drug discovery. Deep learning has shown strong performance in repurposing and screening. It relies on the accurate and fast prediction of a fundamental task: drug-target interaction prediction. DTI prediction task aims to predict the input drug target pair's interaction probability or binding score. Given a powerful DTI model that is able to generalize over a new unseen dataset, we can then extend to repurposing/screening. For repurposing, given a new target of interest, we can first pair it to a repurposing drug library. Then this list of input drug-target pairs is fed into the trained DTI model, which will output the predicted binding score. Similarly, for virtual screening, given a list of screening drug-target pairs we want, the DTI model can output the predicted interaction binding scores. We can then rank the predicted outcome based on their binding scores and test the top-k options in the wet lab after manual inspection. DeepPurpose automates this process. By only requiring one line of code, it aggregates five pretrained deep learning models and retrieves a list of ranked potential outcomes.

Identifying Drug-Target Interactions (DTI) will greatly narrow down the scope of search of candidate medications, and thus can plays a pivotal role in drug discovery. Drugs usually interact with one or more proteins to achieve their functions. However, discovering novel interactions between drugs and target proteins is crucial for the development of new drugs, since the aberrant expression of proteins may cause side effects of drugs.

Considering that in vitro experiments are extremely costly and time-consuming, high efficiency computational prediction methods could serve as promising strategies for drug-target interaction (DTI) prediction. In this project, our goal is to focus on deep learning approaches for drug-target interaction (DTI) prediction.

# CHAPTER 3

# Download Code & Install

## 3.1 Download Code

```
$ git clone https://github.com/kexinhuang12345/DeepPurpose.git
$ ###  Download code repository
$
$
$ cd DeepPurpose
$ ### Change directory to DeepPurpose
```

## 3.2 First time usage: setup conda environment

```
$ conda env create -f environment.yml
$ ## Build virtual environment with all packages installed using conda
$
$ conda activate DeepPurpose
$ ##  Activate conda environment
$
$
$ conda deactivate ### exit
```

## 3.3 Second time and later

```
$ conda activate DeepPurpose
$ ##  Activate conda environment
$
$
$ conda deactivate ### exit
```

# CHAPTER 4

# Case Study

- **1a. Antiviral Drugs Repurposing for SARS-CoV2 3CLPro, using One Line.**

Given a new target sequence (e.g. SARS-CoV2 3CL Protease), retrieve a list of repurposing drugs from a curated drug library of 81 antiviral drugs. The Binding Score is the Kd values. Results aggregated from five pretrained model on BindingDB dataset!

```
from DeepPurpose import oneliner
oneliner.repurpose(*load_SARS_CoV2_Protease_3CL(), *load_antiviral_drugs())
```

- **1b. New Target Repurposing using Broad Drug Repurposing Hub, with One Line.**

Given a new target sequence (e.g. MMP9), retrieve a list of repurposing drugs from Broad Drug Repurposing Hub, which is the default. Results also aggregated from five pretrained model! Note the drug name here is the Pubchem CID since some drug names in Broad is too long.

```
from DeepPurpose import oneliner
oneliner.repurpose(*load_MMP9())
```

- **2. Repurposing using Customized training data, with One Line.**

Given a new target sequence (e.g. SARS-CoV 3CL Pro), training on new data (AID1706 Bioassay), and then retrieve a list of repurposing drugs from a proprietary library (e.g. antiviral drugs). The model can be trained from scratch or finetuned from the pretraining checkpoint!

```
from DeepPurpose import oneliner
from DeepPurpose.dataset import *

oneliner.repurpose(*load_SARS_CoV_Protease_3CL(), *load_antiviral_drugs(no_cid =␣
↪True),  *load_AID1706_SARS_CoV_3CL(), \
        split='HTS', convert_y = False, frac=[0.8,0.1,0.1], pretrained = False, agg =
↪'max_effect')
```

- 3. **A Framework for Drug Target Interaction Prediction, with less than 10 lines of codes.**

Under the hood of one model from scratch, a flexible framework for method researchers:

```
from DeepPurpose import models
from DeepPurpose.utils import *
from DeepPurpose.dataset import *

# Load Data, an array of SMILES for drug,
# an array of Amino Acid Sequence for Target
# and an array of binding values/0-1 label.
# e.g. ['Cc1ccc(CNS(=O)(=O)c2ccc(s2)S(N)(=O)=O)cc1', ...],
#      ['MSHHWGYGKHNGPEHWHKDFPIAKGERQSPVDIDTH...', ...],
#      [0.46, 0.49, ...]
# In this example, BindingDB with Kd binding score is used.
X_drug, X_target, y  = process_BindingDB(download_BindingDB(SAVE_PATH),
                                         y = 'Kd',
                                         binary = False,
                                         convert_to_log = True)

# Type in the encoding names for drug/protein.
drug_encoding, target_encoding = 'MPNN', 'Transformer'

# Data processing, here we select cold protein split setup.
train, val, test = data_process(X_drug, X_target, y,
                                drug_encoding, target_encoding,
                                split_method='cold_protein',
                                frac=[0.7,0.1,0.2])

# Generate new model using default parameters;
# also allow model tuning via input parameters.
config = generate_config(drug_encoding, target_encoding, \
                                                    transformer_n_layer_target = 8)
net = models.model_initialize(**config)

# Train the new model.
# Detailed output including a tidy table storing
#    validation loss, metrics, AUC curves figures and etc.
#    are stored in the ./result folder.
net.train(train, val, test)

# or simply load pretrained model from a model directory path
#   or reproduced model name such as DeepDTA
net = models.model_pretrained(MODEL_PATH_DIR or MODEL_NAME)

# Repurpose using the trained model or pre-trained model
# In this example, loading repurposing dataset using
# Broad Repurposing Hub and SARS-CoV 3CL Protease Target.
X_repurpose, drug_name, drug_cid = load_broad_repurposing_hub(SAVE_PATH)
target, target_name = load_SARS_CoV_Protease_3CL()

_ = models.repurpose(X_repurpose, target, net, drug_name, target_name)

# Virtual screening using the trained model or pre-trained model
X_repurpose, drug_name, target, target_name = \
                ['CCCCCCCOc1cccc(c1)C([O-])=O', ...], ['16007391', ...], \
                ['MLARRKPVLPALTINPTIAEGPSPTSEGASEANLVDLQKKLEEL...', ...],\
                ['P36896', 'P00374']

_ = models.virtual_screening(X_repurpose, target, net, drug_name, target_name)
```

- 4. **Virtual Screening with Customized Training Data with One Line**

Given a list of new drug-target pairs to be screened, retrieve a list of drug-target pairs with top predicted binding scores.

```
from DeepPurpose import oneliner
oneliner.virtual_screening(['MKK...LIDL', ...], ['CC1=C...C4)N', ...])
```

# CHAPTER 5

# DeepPurpose.models

## 5.1 Classifier

```
class DeepPurpose.models.Classifier(nn.Sequential)
```

Classifier (Source) is to make the prediction for DBTA, it serve as a basic component of class DBTA.

**constructor** create an instance of Classifier.

```
__init__(self, model_drug, model_protein, **config)
```

- **model_drug** (DeepPurpose.models.XX) - Encoder model for drug. XX can be "transformer", "MPNN", "CNN", "CNN_RNN" . . . ,
- **model_protein** (DeepPurpose.models.XX) - Encoder model for protein. XX can be "transformer", "CNN", "CNN_RNN" . . . ,
- **config** (kwargs, keyword arguments) - specify the parameter of classifier.

**Calling functions** implement the feedforward procedure of Classifier.

```
forward(self, v_D, v_P)
```

- **v_D** (many types) - input feature for drug encoder model, like "DeepPurpose.models.transformer", "DeepPurpose.models.CNN", "DeepPurpose.models.CNN_RNN", "DeepPurpose.models.MPNN".
- **v_P** (many types) - input feature for protein encoder model, like "DeepPurpose.models.transformer", "DeepPurpose.models.CNN", "DeepPurpose.models.CNN_RNN".

## 5.2 Drug Target Binding Affinity (DTBA) Model

```
class DeepPurpose.models.DBTA
```

**Drug Target Binding Affinity (DBTA)** (Source) include all component, including drug encoder, target encoder and classifier/regressor.

**constructor** create an instance of DBTA.

```
__init__(self, **config)
```

- **config (kwargs, keyword arguments) - specify the parameter of DBTA.**
  - **drug_encoding** (str) - Encoder mode for drug. It can be "transformer", "MPNN", "CNN", "CNN_RNN" ...,
  - **target_encoding** (str) - Encoder mode for protein. It can be "transformer", "CNN", "CNN_RNN" ...,
  - **result_folder** (str) - directory that store the learning log/results.
  - **concrete parameter for encoder model** (repeated)

**test_** include all the test procedure.

```
test_(self, data_generator, model, repurposing_mode = False, test = False):
```

- **data_generator** (iterator) - iterator of torch.utils.data.DataLoader. It can be test data or validation data.
- **model** (DeepPurpose.models.Classifier) - model of DBTA.
- **repurposing_mode** (bool) - If repurposing_mode is True, then do repurposing. Otherwise, do compute the accuracy (including AUC score).
- **test** (bool) - If test is True, plot ROC-AUC and PR-AUC curve. Otherwise, pass.

**train** include all the training procedure.

```
train(self, train, val, test = None, verbose = True)
```

- **train** (torch.utils.data.dataloader) - Train data loader
- **val** (torch.utils.data.dataloader) - Valid data loader
- **test** (torch.utils.data.dataloader) - Test data loader
- **verbose** (bool) - If verbose is True, then print training record every 100 iterations.

**predict** include all the inference procedure.

```
predict(self, df_data)
```

- **df_data** (pd.DataFrame) - specify data that we need to predict.

**save_model** save the well-trained model to specific directory.

```
save_model(self, path_dir)
```

- **path_dir** (str, a directory) - the path where model is saved.

**load_pretrained** load the well-trained model so that we are able to make inference directly and don't have to train model from scratch.

```
load_pretrained(self, path)
```

- **path** (str, a directory) - the path where model is loaded.

## 5.3 Transformer

```
DeepPurpose.models.transformer(nn.Sequential)
```

Transformer (Source) can be used to encode both drug and protein on SMILES.

**constructor** create an instance of Transformer.

```
__init__(self, encoding, **config)
```

- **encoding** (string, "drug" or "protein") - specify input type of the model, "drug" or "protein".
- **config (kwargs, keyword arguments) - specify the parameter of transformer. The keys include**
  - transformer_dropout_rate (float) - dropout rate of transformer.
  - input_dim_drug (int) - input dimension when encoding drug.
  - transformer_emb_size_drug (int) - dimension of embedding in input layer when encoding drug.
  - transformer_n_layer_drug (int) - number of layers in transformer when encoding drug.
  - **todo**

**Calling functions** implement the feedforward procedure of MPNN.

```
forward(self, v)
```

- **v** (tuple of length 2) - input feature of transformer. v[0] (np.array) is index of atoms. v[1] (np.array) is the corresponding mask.

## 5.4 Message Passing Neural Network (MPNN)

```
class DeepPurpose.models.MPNN(nn.Sequential)
```

Message Passing Neural Network (MPNN) (Source) encode drug in its graph representation.

**constructor** create an instance of MPNN class.

```
__init__(self, mpnn_hidden_size, mpnn_depth)
```

- **mpnn_hidden_size** (int) - specify dimension of hidden layer in MPNN, e.g, mpnn_hidden_size = 256.
- **mpnn_depth** (int) - specify depth of MPNN, e.g., mpnn_depth = 3.

**Calling functions** implement the feedforward procedure of MPNN.

```
forward(self, feature)
```

- **feature (tuple of length 5)**
  - feature[0] (torch.Tensor) - atom-level feature
  - feature[1] (torch.Tensor) - bond-level feature
  - feature[2] (torch.Tensor) - neighbor information of every atom
  - feature[3] (torch.Tensor) - neighbor information of every bond
  - feature[4] (torch.Tensor) - store number of atoms and bonds for each molecule in a batch

## 5.5 CNN+RNN

```
class DeepPurpose.models.CNN_RNN(nn.Sequential)
```

CNN_RNN (Source) means a GRU/LSTM on top of a CNN on SMILES.

**constructor** create an instance of CNN_RNN

```
__init__(self, encoding, **config)
```

- **encoding** (string, "drug" or "protein") - specify input type, "drug" or "protein".
- **config (kwargs, keyword arguments) - specify the parameter of transformer. The keys include**
  - cnn_drug_filters (list, each element is int) - specify the size of filter when encoding drug, e.g., cnn_drug_filters = [32,64,96].
  - cnn_drug_kernels (list, each element is int) - specify the size of kernel when encoding drug, e.g., cnn_drug_kernels = [4,6,8].
  - rnn_drug_hid_dim (int) - specify the hidden dimension of RNN when encoding drug, e.g., rnn_drug_hid_dim = 64.
  - rnn_drug_n_layers (int) - specify number of layer in RNN when encoding drug, .e.g, rnn_drug_n_layers = 2.
  - rnn_drug_bidirectional (bool) - specify if RNN is bidirectional when encoding drug, .e.g, rnn_drug_bidirectional = True.
  - hidden_dim_drug (int) - specify the hidden dimension when encoding drug, e.g., hidden_dim_drug = 256.
  - cnn_target_filters (list, each element is int) - specify the size of filter when encoding protein, e.g, cnn_target_filters = [32,64,96].
  - cnn_target_kernels (list, each element is int) - specify the size of kernel when encoding protein, e.g, cnn_target_kernels = [4,8,12].
  - hidden_dim_protein (int) - specify the hidden dimension when encoding protein, e.g., hidden_dim_protein = 256.
  - rnn_target_hid_dim (int) - specify hidden dimension of RNN when encoding protein, e.g., rnn_target_hid_dim = 64.
  - rnn_target_n_layers (int) - specify the number of layer in RNN when encoding protein, e.g., rnn_target_n_layers = 2.
  - rnn_target_bidirectional (bool) - specify if RNN is bidirectional when encoding protein, e.g., rnn_target_bidirectional = True

**Calling functions** implement the feedforward procedure of CNN_RNN.

```
forward(self, v)
```

- **v** (torch.Tensor) - input feature of CNN_RNN.

## 5.6 CNN

```
class DeepPurpose.models.CNN(nn.Sequential)
```

CNN (Convolutional Neural Network) (Source) can be used to encode both drug and protein on SMILES.

**constructor** create an instance of CNN.

```
__init__(self, encoding, **config)
```

- **encoding** (string, "drug" or "protein") - specify input type of model, "drug" or "protein".
- **config (kwargs, keyword arguments) - specify the parameter of CNN. The keys include**
  - cnn_drug_filters (list, each element is int) - specify the size of filter when encoding drug, e.g., cnn_drug_filters = [32,64,96].
  - cnn_drug_kernels (list, each element is int) - specify the size of kernel when encoding drug, e.g., cnn_drug_kernels = [4,6,8].
  - hidden_dim_drug (int) - specify the hidden dimension when encoding drug, e.g., hidden_dim_drug = 256.
  - cnn_target_filters (list, each element is int) - specify the size of filter when encoding protein, e.g, cnn_target_filters = [32,64,96].
  - cnn_target_kernels (list, each element is int) - specify the size of kernel when encoding protein, e.g, cnn_target_kernels = [4,8,12].
  - hidden_dim_protein (int) - specify the hidden dimension when encoding protein, e.g., hidden_dim_protein = 256.

**Calling functions** implement the feedforward procedure of CNN.

```
forward(self, v)
```

- **v** (torch.Tensor) - input feature of CNN.

## 5.7 MLP

```
class DeepPurpose.models.MLP(nn.Sequential)
```

Multi-Layer Perceptron (MLP) (Source) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer.

**constructor** create an instance of MLP

```
__init__(self, input_dim, hidden_dim, hidden_dims)
```

- **input_dim** (int) - dimension of input feature.
- **hidden_dim** (int) - dimension of hidden layer.

**Calling functions** implement the feedforward procedure of MLP.

```
forward(self, v)
```

- **v** (torch.Tensor) - input feature of MLP.

# CHAPTER 6

# DeepPurpose.dataset

We have downloaded most of the small dataset in the repository.

## 6.1 read_file_training_dataset_bioassay

**read_file_training_dataset_bioassay** load bioarray dataset, with one target sequence and multiple drugs and their interaction score with the target.

```
dataset.read_file_training_dataset_bioassay(path)
```

- **path** (str, a directory) - the path of bioassay dataset file. We have requirement on format of file. First line is target sequence. From 2nd line to n-th line, each line a SMILES and interaction score with target sequence. Example: ./toy_data/AID1706.txt

## 6.2 read_file_training_dataset_drug_target_pairs

**read_file_training_dataset_drug_target_pairs** load drug target pairs dataset. We have requirement on format of file. Each line contains a drug SMILES and target sequence and their interaction score. Example: ./toy_data/dti.txt

```
dataset.read_file_training_dataset_drug_target_pairs(path)
```

- **path** (str, a directory) - the path of drug target pairs dataset file. We have requirement on format of file. First line is target sequence. From 2nd line to n-th line, each line a SMILES and interaction score with target sequence. Example: ./toy_data/AID1706.txt

## 6.3 read_file_virtual_screening_drug_target_pairs

**read_file_virtual_screening_drug_target_pairs** load virtual screening drug target pairs dataset. We have requirement on format of file. Each line contains a drug SMILES and target sequence. Example: ./toy_data/dti.txt

```
dataset.read_file_virtual_screening_drug_target_pairs(path)
```

- **path** (str, a directory) - the path of virtual screening drug target pairs dataset file.

## 6.4 load bioarray dataset (read_file_training_dataset_bioassay)

**read_file_repurposing_library** load drug repurposing dataset. We have requirement on format of file. Each line contains a drug SMILES and its name. Example: ./toy_data/??

```
dataset.read_file_repurposing_library(path)
```

- **path** (str, a directory) - the path of drug repurposing dataset file.

## 6.5 read_file_target_sequence

**read_file_target_sequence** load drug repurposing dataset. We have requirement on format of file. The file only have one line. The line contains target name and target sequence. Example: ./toy_data/??

```
dataset.read_file_target_sequence(path)
```

- **path** (str, a directory) - the path of target sequence dataset file.

## 6.6 download_DrugTargetCommons

**download_DrugTargetCommons** load DrugTargetCommons dataset, save it to a specific path. If the path doesn't exist, create the folder.

```
dataset.download_DrugTargetCommons(path)
```

- **path** (str, a directory) - the path that save DrugTargetCommons dataset file. Example: "./data".

## 6.7 process_BindingDB

**process_BindingDB** processes BindingDB dataset.

```
dataset.process_BindingDB(path = None, df = None, y = 'Kd', binary = False, convert_
↪to_log = True, threshold = 30)
```

- **path** (str, a directory) - the path that save BindingDB dataset file. Example: "./data/BindingDB_All.tsv".
- **df** (pandas.DataFrame) - Dataframe that contains input data, if first parameter "path" is None, use the "df".
- **y** (str; can be "Kd", "Ki", "IC50" or "EC50") - specify the binding score.
- **binary** (bool) - If binary is True, formulate prediction task as a binary classification task. Otherwise, formulate the prediction task as a regression task.
- **convert_to_log** (bool) - If True, convert the target score to logspace for easier regression'
- **threshold** (float) - The threshold that select target score ??

## 6.8 load_process_DAVIS

**load_process_DAVIS** load DAVIS dataset.

```
dataset.load_process_DAVIS(path = './data', binary = False, convert_to_log = True,␣
↪threshold = 30)
```

- **path** (str, a directory) - the path that save DAVIS dataset file. Example: "./data".
- **binary** (bool) - If binary is True, formulate prediction task as a binary classification task. Otherwise, formulate the prediction task as a regression task.
- **convert_to_log** (bool) - If True, convert the target score to logspace for easier regression'
- **threshold** (float) - The threshold that select target score ??

## 6.9 load_process_KIBA

**load_process_KIBA** load KIBA dataset.

```
load_process_KIBA(path = './data', binary = False, threshold = 9):
```

- **path** (str, a directory) - the path that save KIBA dataset file. Example: "./data".
- **binary** (bool) - If binary is True, formulate prediction task as a binary classification task. Otherwise, formulate the prediction task as a regression task.
- **threshold** (float) - The threshold that select target score ??

## 6.10 load_AID1706_txt_file

**load_AID1706_txt_file** load KIBA dataset.

```
load_AID1706_txt_file(path = './data')
```

- **path** (str, a directory) - the path that save AID1706 dataset file. Example: "./data".

## 6.11 load_AID1706_SARS_CoV_3CL

**load_AID1706_SARS_CoV_3CL** load AID1706_SARS_CoV_3CL dataset.

```
load_AID1706_SARS_CoV_3CL(path = './data', binary = True, threshold = 15, balanced =␣
↪True, oversample_num = 30, seed = 1)
```

- **path** (str, a directory) - the path that save AID1706_SARS_CoV_3CL dataset file. Example: "./data".
- **binary** (bool) - If binary is True, formulate prediction task as a binary classification task. Otherwise, formulate the prediction task as a regression task.
- **threshold** (float) - The threshold that select target score ??
- **balanced** (bool) - If True, do oversampling to make number of positive and negative samples equal.
- **oversample_num** (int) - control the oversample rate.

- **seed** (int) - random seed in oversample.

## 6.12 load_antiviral_drugs

**load_antiviral_drugs** load antiviral drugs dataset.

```
load_antiviral_drugs(path = './data', no_cid = False)
```

- **path** (str, a directory) - the path that save antiviral drugs dataset file. Example: "./data".
- **no_cid** (bool) - If False, including "Pubchem CID".

## 6.13 load_broad_repurposing_hub

**load_broad_repurposing_hub** load repurposing dataset.

```
load_broad_repurposing_hub(path = './data'):
```

- **path** (str, a directory) - the path that save repurposing dataset file. Example: "./data".

CHAPTER 7

# DeepPurpose.chemutils

## 7.1 DeepPurpose.chemutils.onek_encoding_unk

Given an atom and an allowable atom set, allowable atom set contains a special symbol for unknown atom. The target of onek_encoding_unk function is to transform the atom into one-hot vector. If the atom doesn't exist in the allowable atom set, the use label it as unknown atom.

```
def onek_encoding_unk(x, allowable_set):
    if x not in allowable_set:
        x = allowable_set[-1]
    return list(map(lambda s: x == s, allowable_set))
```

## 7.2 DeepPurpose.chemutils.atom_features

Given an atom in molecular graph, return its feature based on the atom itself, its degree and other information.

```
def atom_features(atom):
    return torch.Tensor(onek_encoding_unk(atom.GetSymbol(), ELEM_LIST)
            + onek_encoding_unk(atom.GetDegree(), [0,1,2,3,4,5])
            + onek_encoding_unk(atom.GetFormalCharge(), [-1,-2,1,2,0])
            + onek_encoding_unk(int(atom.GetChiralTag()), [0,1,2,3])
            + [atom.GetIsAromatic()])
```

## 7.3 DeepPurpose.chemutils.bond_features

Given a bond in molecular graph, return its feature based on the bond itself, its connection information.

```
def bond_features(bond):
    bt = bond.GetBondType()
    stereo = int(bond.GetStereo())
    fbond = [bt == Chem.rdchem.BondType.SINGLE, bt == Chem.rdchem.BondType.DOUBLE, bt␣
↪== Chem.rdchem.BondType.TRIPLE, bt == Chem.rdchem.BondType.AROMATIC, bond.
↪IsInRing()]
    fstereo = onek_encoding_unk(stereo, [0,1,2,3,4,5])
    return torch.Tensor(fbond + fstereo)
```

# CHAPTER 8

# DeepPurpose.oneliner

## 8.1 DeepPurpose.oneliner.repurpose

text:todo

```
def repurpose(target, target_name = None,
                        X_repurpose = None,
                        drug_names = None,
                        train_drug = None,
                        train_target = None,
                        train_y = None,
                        save_dir = './save_folder',
                        pretrained_dir = None,
                        finetune_epochs = 10,
                        finetune_LR = 0.001,
                        finetune_batch_size = 32,
                        convert_y = True,
                        subsample_frac = 1,
                        pretrained = True,
                        split = 'random',
                        frac = [0.7,0.1,0.2],
                        agg = 'agg_mean_max',
                        output_len = 30):
```

## 8.2 DeepPurpose.oneliner.virtual_screening

text:todo

```
def virtual_screening(
                target,
                X_repurpose = None,
```

(continues on next page)

(continued from previous page)

```
                target_name = None,
                drug_names = None,
                train_drug = None,
                train_target = None,
                train_y = None,
                save_dir = './save_folder',
                pretrained_dir = None,
                finetune_epochs = 10,
                finetune_LR = 0.01,
                finetune_batch_size = 32,
                convert_y = True,
                subsample_frac = 1,
                pretrained = True,
                split = 'random',
                frac = [0.7,0.1,0.2],
                agg = 'agg_mean_max',
                output_len = 30):
```

CHAPTER 9

# DeepPurpose.model_helper

todo

CHAPTER 10

# DeepPurpose.utils

CHAPTER 11

# Drug Target Binding Affinity (DTBA) Model

CHAPTER 12

# Drug/Target Encoder

## 12.1 Drug encoding

| Drug Encodings | Description |
|---|---|
| Morgan | Extended-Connectivity Fingerprints |
| Pubchem | Pubchem Substructure-based Fingerprints |
| Daylight | Daylight-type fingerprints |
| rdkit_2d_normalized | Normalized Descriptastorus |
| CNN | Convolutional Neural Network on SMILES |
| CNN_RNN | A GRU/LSTM on top of a CNN on SMILES |
| Transformer | Transformer Encoder on ESPF |
| MPNN | Message-passing neural network |

## 12.2 Target encoding

| Target Encodings | Description |
|---|---|
| AAC | Amino acid composition up to 3-mers |
| PseudoAAC | Pseudo amino acid composition |
| Conjoint_triad | Conjoint triad features |
| Quasi-seq | Quasi-sequence order descriptor |
| CNN | Convolutional Neural Network on target seq |
| CNN_RNN | A GRU/LSTM on top of a CNN on target seq |
| Transformer | Transformer Encoder on ESPF |

## 12.3 Encoder Model

| Encoder Model | Description |
|---|---|
| CNN | Convolutional Neural Network on SMILES |
| CNN_RNN | A GRU/LSTM on top of a CNN on SMILES |
| Transformer | Transformer Encoder on SMILES |
| MPNN | Message Passing Neural Network on Molecular Graph |
| MLP | MultiLayer Perceptron on fix-dim feature vector |

## 12.4 Technical Details

First, we describe the common modules we import in DeepPurpose.

```
import torch
from torch.autograd import Variable
import torch.nn.functional as F
from torch import nn
import numpy as np
import pandas as pd
```

CHAPTER 13

# Processing Data

We have downloaded most of the small dataset in the repository.

## 13.1 Drug-Target Binding Benchmark Dataset

We list public **Drug-Target Binding Benchmark Dataset** that is supported by DeepPurpose and corresponding downloading and processing function.

| Dataset | downloading and processing Function |
|---|---|
| BindingDB | download_BindingDB() to download the data and process_BindingDB() to process the data |
| DAVIS | load_process_DAVIS() to download and process the data |
| KIBA | load_process_KIBA() to download and process the data |

- **Download Link**
  - BindingDB
  - DAVIS
  - KIBA

## 13.2 Repurposing Dataset

We list public **Repurposing Dataset** that is supported by DeepPurpose and corresponding downloading and processing function.

| Dataset | downloading and processing Function |
|---|---|
| Curated Antiviral Drugs Library | load_antiviral_drugs() to load and process the data |
| Broad Repurposing Hub | load_broad_repurposing_hub() downloads and process the data |

- **Download Link**

  - Curated Antiviral Drugs Library
  - Broad Repurposing Hub

## 13.3 Bioassay Data for COVID-19

| Dataset | downloading and processing Function |
|---|---|
| AID1706 | load_AID1706_SARS_CoV_3CL() to load and process |

- **Download Link**
  - AID1706

## 13.4 COVID-19 Targets

| Dataset | downloading and processing Function |
|---|---|
| SARS-CoV 3CL Protease | load_SARS_CoV_Protease_3CL() |
| SARS-CoV2 3CL Protease | load_SARS_CoV2_Protease_3CL() |
| SARS_CoV2 RNA Polymerase | load_SARS_CoV2_RNA_polymerase() |
| SARS-CoV2 Helicase | load_SARS_CoV2_Helicase() |
| SARS-CoV2 3to5_exonuclease | load_SARS_CoV2_3to5_exonuclease() |
| SARS-CoV2 endoRNAse | load_SARS_CoV2_endoRNAse() |

# CHAPTER 14

# Configuration

**generate_config** generate all the configuration that can be used in learning and inference.

```
utils.generate_config(
       drug_encoding,
       target_encoding,
       result_folder = "./result/",
       input_dim_drug = 1024,
       input_dim_protein = 8420,
       hidden_dim_drug = 256,
       hidden_dim_protein = 256,
       cls_hidden_dims = [1024, 1024, 512],
       mlp_hidden_dims_drug = [1024, 256, 64],
       mlp_hidden_dims_target = [1024, 256, 64],
       batch_size = 256,
       train_epoch = 10,
       test_every_X_epoch = 20,
       LR = 1e-4,
       transformer_emb_size_drug = 128,
       transformer_intermediate_size_drug = 512,
       transformer_num_attention_heads_drug = 8,
       transformer_n_layer_drug = 8,
       transformer_emb_size_target = 128,
       transformer_intermediate_size_target = 512,
       transformer_num_attention_heads_target = 8,
       transformer_n_layer_target = 4,
       transformer_dropout_rate = 0.1,
       transformer_attention_probs_dropout = 0.1,
       transformer_hidden_dropout_rate = 0.1,
       mpnn_hidden_size = 50,
       mpnn_depth = 3,
       cnn_drug_filters = [32,64,96],
       cnn_drug_kernels = [4,6,8],
       cnn_target_filters = [32,64,96],
       cnn_target_kernels = [4,8,12],
```

(continues on next page)

(continued from previous page)

```
        rnn_Use_GRU_LSTM_drug = 'GRU',
        rnn_drug_hid_dim = 64,
        rnn_drug_n_layers = 2,
        rnn_drug_bidirectional = True,
        rnn_Use_GRU_LSTM_target = 'GRU',
        rnn_target_hid_dim = 64,
        rnn_target_n_layers = 2,
        rnn_target_bidirectional = True
        )
```

- **drug_encoding** (str) - Encoder mode for drug. It can be "transformer", "MPNN", "CNN", "CNN_RNN" . . . ,
- **target_encoding** (str) - Encoder mode for protein. It can be "transformer", "CNN", "CNN_RNN" . . . ,
- **input_dim_drug** (int) - Dimension of input drug feature.
- **input_dim_protein** (int) - Dimension of input protein feature.
- **hidden_dim_drug** (int) - Dimension of hidden layer of drug feature.
- **hidden_dim_protein** (int) - Dimension of hidden layer of protein feature.
- **batch_size** (int) - batch size
- **train_epoch** (int) - training epoch
- **test_every_X_epoch** (int) - test every X epochs
- **LR** (float) - Learning rate.
- **cls_hidden_dims** (list of int) - hidden dimensions of classifier.
- **mlp_hidden_dims_drug** (list of int) - hidden dimension of MLP when encoding drug.
- **mlp_hidden_dims_target** (list of int) - hidden dimension of MLP when encoding protein.
- **transformer_emb_size_drug** (int) - embedding size of transformer when encoding drug.
- **transformer_intermediate_size_drug** (int) -
- **transformer_num_attention_heads_drug** (int) -
- **transformer_n_layer_drug** (int) -
- **transformer_emb_size_target** (int) -
- **transformer_intermediate_size_target** (int) -
- **transformer_num_attention_heads_target** (int) -
- **transformer_n_layer_target** (int) -
- **transformer_dropout_rate** (float) -

CHAPTER 15

# Utility Function