An Introduction To MoleculeNet

By Bharath Ramsundar | Twitter

One of the most powerful features of DeepChem is that it comes "batteries included" with datasets to use. The DeepChem developer community maintains the MoleculeNet [1] suite of datasets which maintains a large collection of different scientific datasets for use in machine learning applications. The original MoleculeNet suite had 17 datasets mostly focused on molecular properties. Over the last several years, MoleculeNet has evolved into a broader collection of scientific datasets to facilitate the broad use and development of scientific machine learning tools.

These datasets are integrated with the rest of the DeepChem suite so you can conveniently access these through functions in the `dc.molnet` submodule. You've already seen a few examples of these loaders already as you've worked through the tutorial series. The full documentation for the MoleculeNet suite is available in our docs [2].

[1] Wu, Zhenqin, et al. "MoleculeNet: a benchmark for molecular machine learning." Chemical science 9.2 (2018): 513-530.

[2] https://deepchem.readthedocs.io/en/latest/moleculenet.html

Colab

This tutorial and the rest in this sequence can be done in Google colab. If you'd like to open this notebook in colab, you can use the following link.

Open in Colab

Setup

To run DeepChem within Colab, you'll need to run the following installation commands. You can of course run this tutorial locally if you prefer. In that case, don't run these cells since they will download and install DeepChem again on your local machine.

In [ ]:
```
!pip install --pre deepchem
```

We can now import the `deepchem` package to play with.

In [1]:
```
import deepchem as dc
dc.__version__
```

Out[1]:
```
'2.4.0-rc1.dev'
```

MoleculeNet Overview

In the last two tutorials we loaded the Delaney dataset of molecular solubilities. Let's load it one more time.

In [2]:
```
tasks, datasets, transformers = dc.molnet.load_delaney(featurizer='GraphConv', splitter='random')
```

Notice that the loader function we invoke `dc.molnet.load_delaney` lives in the `dc.molnet` submodule of MoleculeNet loaders. Let's take a look at the full collection of loaders available for us

In [3]:
```
[method for method in dir(dc.molnet) if "load_" in method ]
```

```
Out[3]: ['load_bace_classification',
         'load_bace_regression',
         'load_bandgap',
         'load_bbbc001',
         'load_bbbc002',
         'load_bbbp',
         'load_cell_counting',
         'load_chembl',
         'load_chembl25',
         'load_clearance',
         'load_clintox',
         'load_delaney',
         'load_factors',
         'load_function',
         'load_hiv',
         'load_hopv',
         'load_hppb',
         'load_kaggle',
         'load_kinase',
         'load_lipo',
         'load_mp_formation_energy',
         'load_mp_metallicity',
         'load_muv',
         'load_nci',
         'load_pcba',
         'load_pcba_146',
         'load_pcba_2475',
         'load_pdbbind',
         'load_pdbbind_from_dir',
         'load_pdbbind_grid',
         'load_perovskite',
         'load_ppb',
         'load_qm7',
         'load_qm7_from_mat',
         'load_qm7b_from_mat',
         'load_qm8',
         'load_qm9',
         'load_sampl',
         'load_sider',
         'load_sweet',
         'load_thermosol',
         'load_tox21',
         'load_toxcast',
         'load_uspto',
         'load_uv',
         'load_zinc15']
```

The set of MoleculeNet loaders is actively maintained by the DeepChem community and we work on adding new datasets to the collection. Let's see how many datasets there are in MoleculeNet today

```
In [4]: len([method for method in dir(dc.molnet) if "load_" in method ])
```

```
Out[4]: 46
```

MoleculeNet Dataset Categories

There's a lot of different datasets in MoleculeNet. Let's do a quick overview of the different types of datasets available. We'll break datasets into different categories and list loaders which belong to those categories. More details on each of these datasets can be found at https://deepchem.readthedocs.io/en/latest/moleculenet.html. The original MoleculeNet paper [1] provides details about a subset of these papers. We've marked these datasets as "V1" below. All remaining dataset are "V2" and not documented in the older paper.

Quantum Mechanical Datasets

MoleculeNet's quantum mechanical datasets contain various quantum mechanical property prediction tasks. The current set of quantum mechanical datasets includes QM7, QM7b, QM8, QM9. The associated loaders are

- `dc.molnet.load_qm7` : V1
- `dc.molnet.load_qm7b_from_mat` : V1
- `dc.molnet.load_qm8` : V1
- `dc.molnet.load_qm9` : V1

Physical Chemistry Datasets

The physical chemistry dataset collection contain a variety of tasks for predicting various physical properties of molecules.

- `dc.molnet.load_delaney` : V1. This dataset is also referred to as ESOL in the original paper.
- `dc.molnet.load_sampl` : V1. This dataset is also referred to as FreeSolv in the original paper.
- `dc.molnet.load_lipo` : V1. This dataset is also referred to as Lipophilicity in the original paper.
- `dc.molnet.load_thermosol` : V2.
- `dc.molnet.load_hppb` : V2.
- `dc.molnet.load_hopv` : V2. This dataset is drawn from a recent publication [3]

Chemical Reaction Datasets

These datasets hold chemical reaction datasets for use in computational retrosynthesis / forward synthesis.

- `dc.molnet.load_uspto`

Biochemical/Biophysical Datasets

These datasets are drawn from various biochemical/biophysical datasets that measure things like the binding affinity of compounds to proteins.

- `dc.molnet.load_pcba` : V1
- `dc.molnet.load_nci` : V2.
- `dc.molnet.load_muv` : V1
- `dc.molnet.load_hiv` : V1
- `dc.molnet.load_ppb` : V2.
- `dc.molnet.load_bace_classification` : V1. This loader loads the classification task for the BACE dataset from the original MoleculeNet paper.
- `dc.molnet.load_bace_regression` : V1. This loader loads the regression task for the BACE dataset from the original MoleculeNet paper.
- `dc.molnet.load_kaggle` : V2. This dataset is from Merck's drug discovery kaggle contest and is described in [4].
- `dc.molnet.load_factors` : V2. This dataset is from [4].
- `dc.molnet.load_uv` : V2. This dataset is from [4].
- `dc.molnet.load_kinase` : V2. This datset is from [4].

Molecular Catalog Datasets

These datasets provide molecular datasets which have no associated properties beyond the raw SMILES formula or structure. These types of datasets are useful for generative modeling tasks.

- `dc.molnet.load_zinc15` : V2
- `dc.molnet.load_chembl` : V2
- `dc.molnet.load_chembl25` : V2

Physiology Datasets

These datasets measure physiological properties of how molecules interact with human patients.

- `dc.molnet.load_bbbp` : V1
- `dc.molnet.load_tox21` : V1
- `dc.molnet.load_toxcast` : V1
- `dc.molnet.load_sider` : V1
- `dc.molnet.load_clintox` : V1
- `dc.molnet.load_clearance` : V2.

Structural Biology Datasets

These datasets contain 3D structures of macromolecules along with associated properties.

- `dc.molnet.load_pdbbind` : V1

Microscopy Datasets

These datasets contain microscopy image datasets, typically of cell lines. These datasets were not in the original MoleculeNet paper.

- `dc.molnet.load_bbbc001` : V2
- `dc.molnet.load_bbbc002` : V2
- `dc.molnet.load_cell_counting` : V2

Materials Properties Datasets

These datasets compute properties of various materials.

- `dc.molnet.load_bandgap` : V2
- `dc.molnet.load_perovskite` : V2
- `dc.molnet.load_mp_formation_energy` : V2
- `dc.molnet.load_mp_metallicity` : V2

[3] Lopez, Steven A., et al. "The Harvard organic photovoltaic dataset." Scientific data 3.1 (2016): 1-7.

[4] Ramsundar, Bharath, et al. "Is multitask deep learning practical for pharma?." Journal of chemical information and modeling 57.8 (2017): 2068-2076.

MoleculeNet Loaders Explained

All MoleculeNet loader functions take the form `dc.molnet.load_X`. Loader functions return a tuple of arguments `(tasks, datasets, transformers)`. Let's walk through each of these return values and explain what we get:

1. `tasks` : This is a list of task-names. Many datasets in MoleculeNet are "multitask". That is, a given datapoint has multiple labels associated with it. These correspond to different measurements or values associated with this datapoint.
2. `datasets` : This field is a tuple of three `dc.data.Dataset` objects `(train, valid, test)`. These correspond to the training, validation, and test set for this MoleculeNet dataset.
3. `transformers` : This field is a list of `dc.trans.Transformer` objects which were applied to this dataset during processing.

This is abstract so let's take a look at each of these fields for the `dc.molnet.load_delaney` function we invoked above. Let's start with `tasks`.

In [5]:
```
tasks
```

Out[5]:
```
['measured log solubility in mols per litre']
```

We have one task in this dataset which corresponds to the measured log solubility in mol/L. Let's now take a look at `datasets`:

In [6]:
```
datasets
```

Out[6]:
```
(<DiskDataset X.shape: (902,), y.shape: (902, 1), w.shape: (902, 1), ids: ['CCC(C)Cl' 'O=C1NC(=O)NC(=O)C1(C(C)C
)CC=C' 'Oc1ccccn1' ...
  'CCCCCCCC(=O)OCC' 'O=Cc1ccccc1' 'CCCC=C(CC)C=O'], task_names: ['measured log solubility in mols per litre']>,
 <DiskDataset X.shape: (113,), y.shape: (113, 1), w.shape: (113, 1), ids: ['CSc1nc(nc(n1)N(C)C)N(C)C' 'CC#N' 'C
CCCCCC#C' ... 'ClCCBr'
  'CCN(CC)C(=O)CSc1ccc(Cl)nn1' 'CC(=O)OC3CCC4C2CCC1=CC(=O)CCC1(C)C2CCC34C '], task_names: ['measured log solubi
lity in mols per litre']>,
 <DiskDataset X.shape: (113,), y.shape: (113, 1), w.shape: (113, 1), ids: ['CCCCc1c(C)nc(nc1O)N(C)C '
  'Cc3cc2nc1c(=O)[nH]c(=O)nc1n(CC(O)C(O)C(O)CO)c2cc3C'
  'CSc1nc(NC(C)C)nc(NC(C)C)n1' ... 'O=c1[nH]cnc2[nH]ncc12 '
  'CC(=C)C1CC=C(C)C(=O)C1' 'OC(C(=O)c1ccccc1)c2ccccc2'], task_names: ['measured log solubility in mols per litr
e']>)
```

As we mentioned previously, we see that `datasets` is a tuple of 3 datasets. Let's split them out.

In [7]:
```
train, valid, test = datasets
```

In [8]:
```
train
```

Out[8]:
```
<DiskDataset X.shape: (902,), y.shape: (902, 1), w.shape: (902, 1), ids: ['CCC(C)Cl' 'O=C1NC(=O)NC(=O)C1(C(C)C)
CC=C' 'Oc1ccccn1' ...
 'CCCCCCCC(=O)OCC' 'O=Cc1ccccc1' 'CCCC=C(CC)C=O'], task_names: ['measured log solubility in mols per litre']>
```

In [9]:
```
valid
```

Out[9]:
```
<DiskDataset X.shape: (113,), y.shape: (113, 1), w.shape: (113, 1), ids: ['CSc1nc(nc(n1)N(C)C)N(C)C' 'CC#N' 'CC
CCCCC#C' ... 'ClCCBr'
 'CCN(CC)C(=O)CSc1ccc(Cl)nn1' 'CC(=O)OC3CCC4C2CCC1=CC(=O)CCC1(C)C2CCC34C '], task_names: ['measured log solubil
ity in mols per litre']>
```

```
In [10]: test
```

```
Out[10]: <DiskDataset X.shape: (113,), y.shape: (113, 1), w.shape: (113, 1), ids: ['CCCCc1c(C)nc(nc1O)N(C)C '
          'Cc3cc2nc1c(=O)[nH]c(=O)nc1n(CC(O)C(O)C(O)CO)c2cc3C'
          'CSc1nc(NC(C)C)nc(NC(C)C)n1' ... 'O=c1[nH]cnc2[nH]ncc12 '
          'CC(=C)C1CC=C(C)C(=O)C1' 'OC(C(=O)c1ccccc1)c2ccccc2'], task_names: ['measured log solubility in mols per litre
         ']>
```

Let's peek into one of the datapoints in the `train` dataset.

```
In [11]: train.X[0]
```

```
Out[11]: <deepchem.feat.mol_graphs.ConvMol at 0x7fe1ef601438>
```

Note that this is a `dc.feat.mol_graphs.ConvMol` object produced by `dc.feat.ConvMolFeaturizer`. We'll say more about how to control choice of featurization shortly. Finally let's take a look at the `transformers` field:

```
In [12]: transformers
```

```
Out[12]: [<deepchem.trans.transformers.NormalizationTransformer at 0x7fe2029bdfd0>]
```

So we see that one transformer was applied, the `dc.trans.NormalizationTransformer`.

After reading through this description so far, you may be wondering what choices are made under the hood. As we've briefly mentioned previously, datasets can be processed with different choices of "featurizers". Can we control the choice of featurization here? In addition, how was the source dataset split into train/valid/test as three different datasets?

You can use the 'featurizer' and 'splitter' keyword arguments and pass in different strings. Common possible choices for 'featurizer' are 'ECFP', 'GraphConv', 'Weave' and 'smiles2img' corresponding to the `dc.feat.CircularFingerprint`, `dc.feat.ConvMolFeaturizer`, `dc.feat.WeaveFeaturizer` and `dc.feat.SmilesToImage` featurizers. Common possible choices for 'splitter' are `None`, 'index', 'random', 'scaffold' and 'stratified' corresponding to no split, `dc.splits.IndexSplitter`, `dc.splits.RandomSplitter`, `dc.splits.SingletaskStratifiedSplitter`. We haven't talked much about splitters yet, but intuitively they're a way to partition a dataset based on different criteria. We'll say more in a future tutorial.

Instead of a string, you also can pass in any `Featurizer` or `Splitter` object. This is very useful when, for example, a Featurizer has constructor arguments you can use to customize its behavior.

```
In [13]: tasks, datasets, transformers = dc.molnet.load_delaney(featurizer="ECFP", splitter="scaffold")
```

```
In [14]: (train, valid, test) = datasets
```

```
In [15]: train
```

```
Out[15]: <DiskDataset X.shape: (902, 1024), y.shape: (902, 1), w.shape: (902, 1), ids: ['CC(C)=CCCC(C)=CC(=O)' 'CCCC=C'
         'CCCCCCCCCCCCCC' ...
          'Nc2cccc3nc1ccccc1cc23 ' 'C1CCCCCC1' 'OC1CCCCCC1'], task_names: ['measured log solubility in mols per litre']>
```

```
In [16]: train.X[0]
```

```
Out[16]: array([0., 0., 0., ..., 0., 0., 0.])
```

Note that unlike the earlier invocation we have numpy arrays produced by `dc.feat.CircularFingerprint` instead of `ConvMol` objects produced by `dc.feat.ConvMolFeaturizer`.

Give it a try for yourself. Try invoking MoleculeNet to load some other datasets and experiment with different featurizer/split options and see what happens!

Congratulations! Time to join the Community!

Congratulations on completing this tutorial notebook! If you enjoyed working through the tutorial, and want to continue working with DeepChem, we encourage you to finish the rest of the tutorials in this series. You can also help the DeepChem community in the following ways:

Star DeepChem on GitHub

This helps build awareness of the DeepChem project and the tools for open source drug discovery that we're trying to build.

Join the DeepChem Discord

The DeepChem Discord hosts a number of scientists, developers, and enthusiasts interested in deep learning for the life sciences. Join the conversation!