Tutorial 1: The Basic Tools of the Deep Life Sciences

Welcome to DeepChem's introductory tutorial for the deep life sciences. This series of notebooks is a step-by-step guide for you to get to know the new tools and techniques needed to do deep learning for the life sciences. We'll start from the basics, assuming that you're new to machine learning and the life sciences, and build up a repertoire of tools and techniques that you can use to do meaningful work in the life sciences.

Scope: This tutorial will encompass both the machine learning and data handling needed to build systems for the deep life sciences.

Colab

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

Open in Colab

Why do the DeepChem Tutorial?

1) Career Advancement: Applying AI in the life sciences is a booming industry at present. There are a host of newly funded startups and initiatives at large pharmaceutical and biotech companies centered around AI. Learning and mastering DeepChem will bring you to the forefront of this field and will prepare you to enter a career in this field.

2) Humanitarian Considerations: Disease is the oldest cause of human suffering. From the dawn of human civilization, humans have suffered from pathogens, cancers, and neurological conditions. One of the greatest achievements of the last few centuries has been the development of effective treatments for many diseases. By mastering the skills in this tutorial, you will be able to stand on the shoulders of the giants of the past to help develop new medicine.

3) Lowering the Cost of Medicine: The art of developing new medicine is currently an elite skill that can only be practiced by a small core of expert practitioners. By enabling the growth of open source tools for drug discovery, you can help democratize these skills and open up drug discovery to more competition. Increased competition can help drive down the cost of medicine.

Getting Extra Credit

If you're excited about DeepChem and want to get more involved, there are some things that you can do right now:

- Star DeepChem on GitHub! - https://github.com/deepchem/deepchem
- Join the DeepChem forums and introduce yourself! - https://forum.deepchem.io

- Say hi on the DeepChem gitter - https://gitter.im/deepchem/Lobby
- Make a YouTube video teaching the contents of this notebook.

Prerequisites

This tutorial sequence will assume some basic familiarity with the Python data science ecosystem. We will assume that you have familiarity with libraries such as Numpy, Pandas, and TensorFlow. We'll provide some brief refreshers on basics through the tutorial so don't worry if you're not an expert.

Setup

The first step is to get DeepChem up and running. We recommend using Google Colab to work through this tutorial series. You'll need to run the following commands to get DeepChem installed on your colab notebook. Note that this will take something like 5 minutes to run on your colab instance.

```
!curl -Lo conda_installer.py https://raw.githubusercontent.com/deepchem/deepchem/maste
import conda_installer
conda_installer.install()
!/root/miniconda/bin/conda info -e
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3490  100  3490    0     0   9640      0 --:--:-- --:--:-- --:--:--  9640
add /root/miniconda/lib/python3.6/site-packages to PYTHONPATH
python version: 3.6.9
fetching installer from https://repo.continuum.io/miniconda/Miniconda3-latest-Li
done
installing miniconda to /root/miniconda
done
installing rdkit, openmm, pdbfixer
added conda-forge to channels
added omnia to channels
done
conda packages installation finished!
# conda environments:
#
base                  *  /root/miniconda
```

```
!pip install --pre deepchem
```

```
Collecting deepchem
  Downloading https://files.pythonhosted.org/packages/c9/02/174188efbfc546aa840e
     |████████████████████████████████| 542kB 4.2MB/s
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (
```

```
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-pacl
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-pacl
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages
Installing collected packages: deepchem
Successfully installed deepchem-2.5.0.dev20210205230509
```

You can of course run this tutorial locally if you prefer. In this case, don't run the above cell since it will download and install Anaconda on your local machine. In either case, we can now import the `deepchem` package to play with.

```
import deepchem as dc
dc.__version__
```

```
'2.5.0.dev'
```

Training a Model with DeepChem: A First Example

Deep learning can be used to solve many sorts of problems, but the basic workflow is usually the same. Here are the typical steps you follow.

1. Select the data set you will train your model on (or create a new data set if there isn't an existing suitable one).
2. Create the model.
3. Train the model on the data.
4. Evaluate the model on an independent test set to see how well it works.
5. Use the model to make predictions about new data.

With DeepChem, each of these steps can be as little as one or two lines of Python code. In this tutorial we will walk through a basic example showing the complete workflow to solve a real world scientific problem.

The problem we will solve is predicting the solubility of small molecules given their chemical formulas. This is a very important property in drug development: if a proposed drug isn't soluble enough, you probably won't be able to get enough into the patient's bloodstream to have a therapeutic effect. The first thing we need is a data set of measured solubilities for real molecules. One of the core components of DeepChem is MoleculeNet, a diverse collection of chemical and molecular data sets. For this tutorial, we can use the Delaney solubility data set.

```
tasks, datasets, transformers = dc.molnet.load_delaney(featurizer='GraphConv')
train_dataset, valid_dataset, test_dataset = datasets
```

I won't say too much about this code right now. We will see many similar examples in later tutorials. There are two details I do want to draw your attention to. First, notice the `featurizer` argument passed to the `load_delaney()` function. Molecules can be represented in many ways. We therefore tell it which representation we want to use, or in more technical language, how to "featurize" the data. Second, notice that we actually get three different data sets: a training set, a validation set, and a test set. Each of these serves a different function in the standard deep learning workflow.

Now that we have our data, the next step is to create a model. We will use a particular kind of model called a "graph convolutional network", or "graphconv" for short.

```
model = dc.models.GraphConvModel(n_tasks=1, mode='regression', dropout=0.2)
```

Here again I will not say much about the code. Later tutorials will give lots more information about `GraphConvModel`, as well as other types of models provided by DeepChem.

We now need to train the model on the data set. We simply give it the data set and tell it how many epochs of training to perform (that is, how many complete passes through the data to make).

```
model.fit(train_dataset, nb_epoch=100)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
```

```
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
```

If everything has gone well, we should now have a fully trained model! But do we? To find out, we must evaluate the model on the test set. We do that by selecting an evaluation metric and calling `evaluate()` on the model. For this example, let's use the Pearson correlation, also known as r^2, as our metric. We can evaluate it on both the training set and test set.

```
metric = dc.metrics.Metric(dc.metrics.pearson_r2_score)
print("Training set score:", model.evaluate(train_dataset, [metric], transformers))
print("Test set score:", model.evaluate(test_dataset, [metric], transformers))
```

```
Training set score: {'pearson_r2_score': 0.9102931879249282}
Test set score: {'pearson_r2_score': 0.6539062458926771}
```

Notice that it has a higher score on the training set than the test set. Models usually perform better on the particular data they were trained on than they do on similar but independent data. This is called "overfitting", and it is the reason it is essential to evaluate your model on an independent test set.

Our model still has quite respectable performance on the test set. For comparison, a model that produced totally random outputs would have a correlation of 0, while one that made perfect predictions would have a correlation of 1. Our model does quite well, so now we can use it to make predictions about other molecules we care about.

Since this is just a tutorial and we don't have any other molecules we specifically want to predict, let's just use the first ten molecules from the test set. For each one we print out the chemical structure (represented as a SMILES string) and the predicted solubility.

```
solubilities = model.predict_on_batch(test_dataset.X[:10])
for molecule, solubility in zip(test_dataset.ids, solubilities):
    print(solubility, molecule)
```

```
[-1.4672143] c1cc2ccc3cccc4ccc(c1)c2c34
[1.2829525] Cc1cc(=O)[nH]c(=S)[nH]1
[-0.36151502] Oc1ccc(cc1)C2(OC(=O)c3ccccc23)c4ccc(O)cc4
[-1.8508666] c1ccc2c(c1)cc3ccc4cccc5ccc2c3c45
[-1.2642734] C1=Cc2cccc3cccc1c23
[1.7804294] CC1CO1
[-0.03542721] CCN2c1ccccc1N(C)C(=S)c3cccnc23
[-0.5321215] CC12CCC3C(CCc4cc(O)ccc34)C2CCC1=O
[-1.165889] Cn2cc(c1ccccc1)c(=O)c(c2)c3cccc(c3)C(F)(F)F
[-0.2509461] ClC(Cl)(Cl)C(NC=O)N1C=CN(C=C1)C(NC=O)C(Cl)(Cl)Cl
```

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 Gitter

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

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.