Advanced model training using hyperopt

In the Advanced Model Training tutorial we have already taken a look into hyperparameter optimasation using GridHyperparamOpt in the deepchem pacakge. In this tutorial, we will take a look into another hyperparameter tuning library called hyperopt.

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 and Hyperopt 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 and Hyperopt in your local machine again.

In [1]:
```
!pip install deepchem
!pip install hyperopt
```

```
Collecting deepchem
  Downloading deepchem-2.6.1-py3-none-any.whl (608 kB)
     |▌                               | 10 kB 31.6 MB/s eta 0:00:01
     |█                               | 20 kB 27.2 MB/s eta 0:00:01
     |█▌                              | 30 kB 11.2 MB/s eta 0:00:01
     |██                              | 40 kB 8.9 MB/s eta 0:00:01
     |██▋                             | 51 kB 5.3 MB/s eta 0:00:01
     |███▏                            | 61 kB 5.4 MB/s eta 0:00:01
     |███▊                            | 71 kB 5.4 MB/s eta 0:00:01
     |████▎                           | 81 kB 6.1 MB/s eta 0:00:01
     |████▉                           | 92 kB 6.2 MB/s eta 0:00:01
     |█████▍                          | 102 kB 5.2 MB/s eta 0:00:01
     |█████▉                          | 112 kB 5.2 MB/s eta 0:00:01
     |██████▍                         | 122 kB 5.2 MB/s eta 0:00:01
     |███████                         | 133 kB 5.2 MB/s eta 0:00:01
     |███████▌                        | 143 kB 5.2 MB/s eta 0:00:01
     |████████                        | 153 kB 5.2 MB/s eta 0:00:01
     |████████▋                       | 163 kB 5.2 MB/s eta 0:00:01
     |█████████▏                      | 174 kB 5.2 MB/s eta 0:00:01
     |█████████▋                      | 184 kB 5.2 MB/s eta 0:00:01
     |██████████▏                     | 194 kB 5.2 MB/s eta 0:00:01
     |██████████▊                     | 204 kB 5.2 MB/s eta 0:00:01
     |███████████▎                    | 215 kB 5.2 MB/s eta 0:00:01
     |███████████▉                    | 225 kB 5.2 MB/s eta 0:00:01
     |████████████▍                   | 235 kB 5.2 MB/s eta 0:00:01
     |████████████▉                   | 245 kB 5.2 MB/s eta 0:00:01
     |█████████████▍                  | 256 kB 5.2 MB/s eta 0:00:01
     |██████████████                  | 266 kB 5.2 MB/s eta 0:00:01
     |██████████████▌                 | 276 kB 5.2 MB/s eta 0:00:01
     |███████████████                 | 286 kB 5.2 MB/s eta 0:00:01
     |███████████████▋                | 296 kB 5.2 MB/s eta 0:00:01
     |████████████████▏               | 307 kB 5.2 MB/s eta 0:00:01
     |████████████████▋               | 317 kB 5.2 MB/s eta 0:00:01
     |█████████████████▏              | 327 kB 5.2 MB/s eta 0:00:01
     |█████████████████▊              | 337 kB 5.2 MB/s eta 0:00:01
     |██████████████████▎             | 348 kB 5.2 MB/s eta 0:00:01
     |██████████████████▉             | 358 kB 5.2 MB/s eta 0:00:01
     |███████████████████▍            | 368 kB 5.2 MB/s eta 0:00:01
     |███████████████████▉            | 378 kB 5.2 MB/s eta 0:00:01
     |████████████████████▌           | 389 kB 5.2 MB/s eta 0:00:01
     |█████████████████████           | 399 kB 5.2 MB/s eta 0:00:01
     |█████████████████████▌          | 409 kB 5.2 MB/s eta 0:00:01
     |██████████████████████          | 419 kB 5.2 MB/s eta 0:00:01
     |██████████████████████▋         | 430 kB 5.2 MB/s eta 0:00:01
     |███████████████████████▏        | 440 kB 5.2 MB/s eta 0:00:01
     |███████████████████████▋        | 450 kB 5.2 MB/s eta 0:00:01
     |████████████████████████▏       | 460 kB 5.2 MB/s eta 0:00:01
     |████████████████████████▊       | 471 kB 5.2 MB/s eta 0:00:01
     |█████████████████████████▎      | 481 kB 5.2 MB/s eta 0:00:01
     |█████████████████████████▉      | 491 kB 5.2 MB/s eta 0:00:01
     |██████████████████████████▍     | 501 kB 5.2 MB/s eta 0:00:01
     |███████████████████████████     | 512 kB 5.2 MB/s eta 0:00:01
     |███████████████████████████▋    | 522 kB 5.2 MB/s eta 0:00:01
     |████████████████████████████▏   | 532 kB 5.2 MB/s eta 0:00:01
     |████████████████████████████▋   | 542 kB 5.2 MB/s eta 0:00:01
     |█████████████████████████████▏  | 552 kB 5.2 MB/s eta 0:00:01
     |█████████████████████████████▊  | 563 kB 5.2 MB/s eta 0:00:01
     |██████████████████████████████▎ | 573 kB 5.2 MB/s eta 0:00:01
     |██████████████████████████████▉ | 583 kB 5.2 MB/s eta 0:00:01
     |███████████████████████████████▍| 593 kB 5.2 MB/s eta 0:00:01
     |████████████████████████████████| 604 kB 5.2 MB/s eta 0:00:01
     |████████████████████████████████| 608 kB 5.2 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from deepchem) (1.4.1)
Collecting numpy>=1.21
  Downloading numpy-1.21.5-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (15.7 MB)
     |████████████████████████████████| 15.7 MB 25.3 MB/s
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from deepchem) (1.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from deepchem) (1.3.5)
Collecting rdkit-pypi
  Downloading rdkit_pypi-2021.9.4-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20.6 MB)
     |████████████████████████████████| 20.6 MB 1.4 MB/s
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from deepchem) (1.1.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->deepchem) (2
018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->de
epchem) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->
pandas->deepchem) (1.15.0)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from rdkit-pypi->deepchem) (7.1
.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn
->deepchem) (3.1.0)
Installing collected packages: numpy, rdkit-pypi, deepchem
  Attempting uninstall: numpy
    Found existing installation: numpy 1.19.5
    Uninstalling numpy-1.19.5:
      Successfully uninstalled numpy-1.19.5
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
yellowbrick 1.3.post1 requires numpy<1.20,>=1.16.0, but you have numpy 1.21.5 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
Successfully installed deepchem-2.6.1 numpy-1.21.5 rdkit-pypi-2021.9.4
Requirement already satisfied: hyperopt in /usr/local/lib/python3.7/dist-packages (0.1.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (from hyperopt) (2.6.3)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from hyperopt) (0.16.0)
Requirement already satisfied: pymongo in /usr/local/lib/python3.7/dist-packages (from hyperopt) (4.0.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from hyperopt) (1.4.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from hyperopt) (1.21.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from hyperopt) (4.62.3)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from hyperopt) (1.15.0)
```

Hyperparameter Optimization via hyperopt

Let's start by loading the HIV dataset. It classifies over 40,000 molecules based on whether they inhibit HIV replication.

In [2]:
```python
import deepchem as dc
tasks, datasets, transformers = dc.molnet.load_hiv(featurizer='ECFP', split='scaffold')
train_dataset, valid_dataset, test_dataset = datasets
```

```
'split' is deprecated.  Use 'splitter' instead.
```

Now, lets import the hyperopt library, which we will be using to fund the best parameters

In [3]:
```python
from hyperopt import hp, fmin, tpe, Trials
```

Then we have to declare a dictionary with all the hyperparameters and their range that you will be tuning them in. This dictionary will serve as the search space for the hyperopt. Some basic ways of declaring the ranges in the dictionary are:

- hp.choice('label',[*choices*]) : this is used to specify a list of choices
- hp.uniform('label' ,low=*low_value* ,high=*high_value*) : this is used to specify a uniform distibution between the low and high values. The values between them can be any real number, not necessaarily an integer.

Here, we are going to use a multitaskclassifier to classify the HIV dataset and hence the appropriate search space is as follows.

In [ ]:
```python
search_space = {
    'layer_sizes': hp.choice('layer_sizes',[[500], [1000], [2000],[1000,1000]]),
    'dropouts': hp.uniform('dropout',low=0.2, high=0.5),
    'learning_rate': hp.uniform('learning_rate',high=0.001, low=0.0001)
}
```

We should then declare a function to be minimized by the hyperopt. So, here we should use the function to minimize our multitaskclassifier model. Additionally, we are using a validation callback to validate the classifier for every 1000 steps, then we are passing the best score as the return. The metric used here is 'roc_auc_score', which needs to be maximized. To maximize a non-negative value is equivalent to minimize its opposite number, hence we are returning the negative of the validation score.

In [ ]:
```
import tempfile
#tempfile is used to save the best checkpoint later in the program.

metric = dc.metrics.Metric(dc.metrics.roc_auc_score)

def fm(args):
  save_dir = tempfile.mkdtemp()
  model = dc.models.MultitaskClassifier(n_tasks=len(tasks),n_features=1024,layer_sizes=args['layer_sizes'],drop
  #validation callback that saves the best checkpoint, i.e the one with the maximum score.
  validation=dc.models.ValidationCallback(valid_dataset, 1000, [metric],save_dir=save_dir,transformers=transfor

  model.fit(train_dataset, nb_epoch=25,callbacks=validation)

  #restoring the best checkpoint and passing the negative of its validation score to be minimized.
  model.restore(model_dir=save_dir)
  valid_score = model.evaluate(valid_dataset, [metric], transformers)

  return -1*valid_score['roc_auc_score']
```

Here, we are calling the fmin function of the hyperopt, where we pass on the function to be minimized, the algorithm to be followed, max number of evals and a trials object. The Trials object is used to keep All hyperparameters, loss, and other information, this means you can access them after running optimization. Also, trials can help you to save important information and later load and then resume the optimization process.

Moreover, for the algorithm there are three choice which can be used without any additional configuration. they are :-

- Random Search - rand.suggest
- TPE (Tree Parzen Estimators) - tpe.suggest
- Adaptive TPE - atpe.suggest

In [ ]:
```
trials=Trials()
best = fmin(fm,
                space= search_space,
                algo=tpe.suggest,
                max_evals=15,
                trials = trials)
```

```
  0%|          | 0/15 [00:00<?, ?it/s, best loss: ?]Step 1000 validation: roc_auc_score=0.777648
Step 2000 validation: roc_auc_score=0.755485
Step 3000 validation: roc_auc_score=0.739519
Step 4000 validation: roc_auc_score=0.764756
Step 5000 validation: roc_auc_score=0.757006
Step 6000 validation: roc_auc_score=0.752609
Step 7000 validation: roc_auc_score=0.763002
Step 8000 validation: roc_auc_score=0.749202
  7%|▋         | 1/15 [05:37<1:18:46, 337.58s/it, best loss: -0.7776476459925534]Step 1000 validation: roc_auc_s
core=0.750455
Step 2000 validation: roc_auc_score=0.783594
Step 3000 validation: roc_auc_score=0.775872
Step 4000 validation: roc_auc_score=0.768825
Step 5000 validation: roc_auc_score=0.769555
Step 6000 validation: roc_auc_score=0.765324
Step 7000 validation: roc_auc_score=0.771146
Step 8000 validation: roc_auc_score=0.760138
 13%|█▎        | 2/15 [07:05<41:16, 190.51s/it, best loss: -0.7835939030962179]  Step 1000 validation: roc_auc_s
core=0.744178
Step 2000 validation: roc_auc_score=0.765406
Step 3000 validation: roc_auc_score=0.76532
Step 4000 validation: roc_auc_score=0.769255
Step 5000 validation: roc_auc_score=0.77029
Step 6000 validation: roc_auc_score=0.768024
Step 7000 validation: roc_auc_score=0.764157
Step 8000 validation: roc_auc_score=0.756805
 20%|██        | 3/15 [09:40<34:53, 174.42s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.714572
Step 2000 validation: roc_auc_score=0.770712
Step 3000 validation: roc_auc_score=0.777914
Step 4000 validation: roc_auc_score=0.76923
Step 5000 validation: roc_auc_score=0.774823
Step 6000 validation: roc_auc_score=0.775927
Step 7000 validation: roc_auc_score=0.777054
Step 8000 validation: roc_auc_score=0.778508
```

```
 27%|██▋       | 4/15 [12:12<30:22, 165.66s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.743939
Step 2000 validation: roc_auc_score=0.759478
Step 3000 validation: roc_auc_score=0.738839
Step 4000 validation: roc_auc_score=0.751084
Step 5000 validation: roc_auc_score=0.740504
Step 6000 validation: roc_auc_score=0.753612
Step 7000 validation: roc_auc_score=0.71802
Step 8000 validation: roc_auc_score=0.761025
 33%|███▎      | 5/15 [17:40<37:21, 224.16s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.74099
Step 2000 validation: roc_auc_score=0.767516
Step 3000 validation: roc_auc_score=0.767338
Step 4000 validation: roc_auc_score=0.775691
Step 5000 validation: roc_auc_score=0.768731
Step 6000 validation: roc_auc_score=0.755029
Step 7000 validation: roc_auc_score=0.767115
Step 8000 validation: roc_auc_score=0.764744
 40%|████      | 6/15 [22:48<37:54, 252.71s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.713761
Step 2000 validation: roc_auc_score=0.759518
Step 3000 validation: roc_auc_score=0.765853
Step 4000 validation: roc_auc_score=0.771976
Step 5000 validation: roc_auc_score=0.772762
Step 6000 validation: roc_auc_score=0.773206
Step 7000 validation: roc_auc_score=0.775565
Step 8000 validation: roc_auc_score=0.768521
 47%|████▋     | 7/15 [27:53<35:58, 269.84s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.717178
Step 2000 validation: roc_auc_score=0.754258
Step 3000 validation: roc_auc_score=0.767905
Step 4000 validation: roc_auc_score=0.762917
Step 5000 validation: roc_auc_score=0.766162
Step 6000 validation: roc_auc_score=0.767581
Step 7000 validation: roc_auc_score=0.770746
Step 8000 validation: roc_auc_score=0.77597
 53%|█████▎    | 8/15 [30:36<27:29, 235.64s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.74314
Step 2000 validation: roc_auc_score=0.757408
Step 3000 validation: roc_auc_score=0.76668
Step 4000 validation: roc_auc_score=0.768104
Step 5000 validation: roc_auc_score=0.746377
Step 6000 validation: roc_auc_score=0.745282
Step 7000 validation: roc_auc_score=0.74113
Step 8000 validation: roc_auc_score=0.734482
 60%|██████    | 9/15 [36:53<28:00, 280.04s/it, best loss: -0.7835939030962179]Step 1000 validation: roc_auc_sco
re=0.743204
Step 2000 validation: roc_auc_score=0.76912
Step 3000 validation: roc_auc_score=0.769981
Step 4000 validation: roc_auc_score=0.784163
Step 5000 validation: roc_auc_score=0.77536
Step 6000 validation: roc_auc_score=0.779237
Step 7000 validation: roc_auc_score=0.782344
Step 8000 validation: roc_auc_score=0.779085
 67%|██████▋   | 10/15 [38:23<18:26, 221.33s/it, best loss: -0.7841634210268469]Step 1000 validation: roc_auc_sc
ore=0.743565
Step 2000 validation: roc_auc_score=0.765063
Step 3000 validation: roc_auc_score=0.75284
Step 4000 validation: roc_auc_score=0.759978
Step 5000 validation: roc_auc_score=0.74255
Step 6000 validation: roc_auc_score=0.721809
Step 7000 validation: roc_auc_score=0.729863
Step 8000 validation: roc_auc_score=0.73075
 73%|███████▎  | 11/15 [44:07<17:15, 258.91s/it, best loss: -0.7841634210268469]Step 1000 validation: roc_auc_sc
ore=0.695949
Step 2000 validation: roc_auc_score=0.765082
Step 3000 validation: roc_auc_score=0.756256
Step 4000 validation: roc_auc_score=0.771923
Step 5000 validation: roc_auc_score=0.758841
Step 6000 validation: roc_auc_score=0.759393
Step 7000 validation: roc_auc_score=0.765971
Step 8000 validation: roc_auc_score=0.747064
 80%|████████  | 12/15 [48:54<13:21, 267.23s/it, best loss: -0.7841634210268469]Step 1000 validation: roc_auc_sc
ore=0.757871
Step 2000 validation: roc_auc_score=0.765296
Step 3000 validation: roc_auc_score=0.769748
Step 4000 validation: roc_auc_score=0.776487
Step 5000 validation: roc_auc_score=0.775009
Step 6000 validation: roc_auc_score=0.779539
Step 7000 validation: roc_auc_score=0.763165
Step 8000 validation: roc_auc_score=0.772093
 87%|████████▋ | 13/15 [50:22<07:06, 213.15s/it, best loss: -0.7841634210268469]Step 1000 validation: roc_auc_sc
ore=0.720166
```

```
Step 2000 validation: roc_auc_score=0.768489
Step 3000 validation: roc_auc_score=0.782853
Step 4000 validation: roc_auc_score=0.785556
Step 5000 validation: roc_auc_score=0.78583
Step 6000 validation: roc_auc_score=0.786569
Step 7000 validation: roc_auc_score=0.779249
Step 8000 validation: roc_auc_score=0.783423
 93%|█████████▎| 14/15 [51:52<02:55, 175.93s/it, best loss: -0.7865693280913189]Step 1000 validation: roc_auc_sc
ore=0.743232
Step 2000 validation: roc_auc_score=0.762007
Step 3000 validation: roc_auc_score=0.771809
Step 4000 validation: roc_auc_score=0.755023
Step 5000 validation: roc_auc_score=0.769812
Step 6000 validation: roc_auc_score=0.769867
Step 7000 validation: roc_auc_score=0.777354
Step 8000 validation: roc_auc_score=0.775313
100%|██████████| 15/15 [56:47<00:00, 227.13s/it, best loss: -0.7865693280913189]
```

The code below is used to print the best hyperparameters found by the hyperopt.

```
In [ ]: print("Best: {}".format(best))
```

```
Best: {'dropout': 0.3749846096922802, 'layer_sizes': 0, 'learning_rate': 0.0007544819475363869}
```

The hyperparameter found here may not be necessarily the best one, but gives a general idea on which parameters are effective. To get mroe accurate results, one has to increase the number of validation epochs and the epochs the model fit. But doing so may increase the time in finding the best hyperparameters.

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!