

# MPNN Code Digest

Shasha Feng

# A Digest of “get\_acl9\_data.py”

```

def to_graph(mol):
    # number of atoms in each molecule
    n_atoms = mol.get_num_atoms()

    # atom features
    atom_feat = mol.get_atom_features()

    # convert 1-of-K encoding to index
    atom_feat = np.argmax(atom_feat, axis=1)

    # edge features, shape N X N X 8
    # first 6 channels: bond-wise adjacency matrix
    # 6-th channel: is-in-the-same-ring adjacency matrix
    # 7-th channel: distance matrix
    pair_feat = mol.get_pair_features()

    return atom_feat, pair_feat[:, :, :6]

```

- It returns the atom feature and the first 6 edge features (bond-wise adjacency matrix);
- Function 'to\_graph' is used in 'dump\_data' function.
- The returned pair feature is of size (N, N, 6)

```

def dump_data(dataset, tag='train'):
    count = 0
    global num_graphs
    print('Dump {} data!'.format(tag))
    for (X_b, y_b, w_b, ids_b) in dataset.iterbatches(
        batch_size=1, deterministic=True, pad_batches=False):
        assert len(X_b) == 1


        if count % 100 == 0:
            print('{:.1f} %\r'.format(100 * count / float(len(dataset))), end='')

    data_dict = {}
    for im, mol in enumerate(X_b):
        global num_nodes, num_edges
        node_feat, adj_s = to_graph(mol)
        data_dict['node_feat'] = node_feat
        adj_simple = np.sum(adj_s, axis=2)
        D_list, V_list, L_list = get_multi_graph_laplacian_eigs(
            adj_s, graph_laplacian_type='L4', use_eigen_decomp=True, is_sym=True)
        D, V, L4 = get_graph_laplacian_eigs(
            adj_simple,
            graph_laplacian_type='L4',
            use_eigen_decomp=True,
            is_sym=True)

        L6 = get_laplacian(adj_simple, graph_laplacian_type='L6')
        L7 = get_laplacian(adj_simple, graph_laplacian_type='L7')

```

adj\_s: N\*N\*6;  
 adj\_simple: N\*N  
 Compressed on the 3rd dimension;



- The initial edge feature matrix is compressed on 3rd dim, used for later SVD.

(Continue from last page function `'dump_data()'`)

```
data_dict['L_multi'] = np.stack(L_list, axis=2)
data_dict['L_simple_4'] = L4
data_dict['L_simple_6'] = L6
data_dict['L_simple_7'] = L7
```

What's format of D and V?

*# N.B.: for some edge type, adjacency matrix may be diagonal*

```
data_dict['D_simple'] = D if D is not None else np.ones(adjs.shape[0])
data_dict['V_simple'] = V if V is not None else np.eye(adjs.shape[0])
data_dict['D_multi'] = D_list
data_dict['V_multi'] = V_list
```

```
num_nodes += node_feat.shape[0]
num_edges += np.sum(adj_simple) / 2.0
```

```
num_graphs += 1.0
data_dict['label'] = y_b
data_dict['label_weight'] = w_b
```

```
pickle.dump(
    data_dict,
    open(
        os.path.join(save_dir, 'QM8_preprocess_{}_{:07d}.p'.format(
            tag, count)), 'wb'))
```

```
count += 1    #count: for print purpose
```

```
print('100.0 %%')
```

- The 'y\_b', 'w\_b' are label and label weight;
- Final result is stored in a dict;
- Batchsize=1, each molecule is processed and pickled;

```

def load_qm8(featurizer='MP',
            split='random',
            reload=True,
            move_mean=True):
    """
    MP: message passing; Only MP is added here.
    Deepchem supports featurizers: 'CoulombMatrix', 'BPSymmetryFunctionInput',
        'MP', 'Raw', 'ECFP', 'GraphConv', 'Weave';
    reload: True/False; If True, the 2nd time it loads directly from disk, instead of
    calculating again;
    move_mean: True/False; For data transformation;
    """
    # Set up workdir #
    # data_dir = '/tmp', this is a system folder for temporary files
    data_dir = dc.utils.get_data_dir()      # /tmp : a system folder for temporary files

    if reload:
        if move_mean:
            dir_name = "all/" + featurizer + "/" + str(split)
        else:
            dir_name = "all/" + featurizer + "_mean_unmoved/" + str(split)
    save_dir = os.path.join(data_dir, dir_name)
    # save_dir = '/tmp/qm8/MP/random'

    # Get the sdf data ready, "all.sdf", "all.sdf.csv" two files in the same folder;
    dataset_file = 'ac19/10.sdf'

```

- Specify the directory for saving the data;

```

# Get to know the tasks                                # the entries in csv file
qm8_tasks = [
    "property_0", "property_1", "property_2", "property_3", "property_4",
    "property_5", "property_6", "property_7", "property_8", "property_9",
    "property_10", "property_11"
]

# If reload, not need to calculate #
if reload:
    loaded, all_dataset, transformers = dc.utils.save.load_dataset_from_disk(
        save_dir)
    if loaded:
        return qm8_tasks, all_dataset, transformers

# Featurizer defined: I omitted other featurizers
if featurizer == 'MP':
    featurizer = dc.featurizer.WeaveFeaturizer(
        graph_distance=False, explicit_H=True)    # Define a featurizer from deepchem's
                                                    weave featurizer

# Utilize deepchem's tool to load SDF file
loader = dc.data.SDFLoader(
    tasks=qm8_tasks,
    smiles_field="smiles",
    mol_field="mol",
    featurizer=featurizer)

dataset = loader.featurize(dataset_file)          # Featurize it

```

- Utilize dc tool to load sdf file and featurize it
- Format of 'dataset'?

```

# Select a splitting method
if split == None:
    raise ValueError()

splitters = {
    'index': dc.splits.IndexSplitter(),
    'random': dc.splits.RandomSplitter(),
    'stratified': dc.splits.SingletaskStratifiedSplitter(task_number=0)
}
splitter = splitters[split]
train_dataset, valid_dataset, test_dataset = splitter.train_valid_test_split(
    dataset)
print('split finished')

# Define a tranformation method
transformers = [
    dc.trans.NormalizationTransformer(
        transform_y=True, dataset=train_dataset, move_mean=move_mean)
]
for transformer in transformers:
    train_dataset = transformer.transform(train_dataset)
    valid_dataset = transformer.transform(valid_dataset)
    test_dataset = transformer.transform(test_dataset)
print('transformation finished')

if reload:
    dc.utils.save.save_dataset_to_disk(
        save_dir, train_dataset, valid_dataset, test_dataset, transformers)
return qm8_tasks, (train_dataset, valid_dataset, test_dataset), transformers

```

- Split the data;
- Transform data by normalization and use the same transformer and save it for future;



```

if __name__ == '__main__':
    tasks, datasets, transformers = load_qm8(featurizer='MP') #dc.molnet.load_qm8(
        # featurizer='MP', reload=False)
    train_dataset, dev_dataset, test_dataset = datasets
    pickle.dump(datasets, open('ac19/dc_feat_datasets.p', 'wb'))

    dump_data(train_dataset, 'train')
    dump_data(dev_dataset, 'dev')
    dump_data(test_dataset, 'test')

    mean_label = transformers[0].y_means
    std_label = transformers[0].y_stds

    print('mean = {}'.format(mean_label))
    print('std = {}'.format(std_label))
    print('average nodes per graph = {}'.format(num_nodes / num_graphs))
    print('average edges per graph = {}'.format(num_edges / num_graphs))

    meta_data = {'mean': mean_label, 'std': std_label}
    pickle.dump(meta_data, open(os.path.join(data_folder, 'QM8_meta.p'), 'wb'))

```

- Because it is a normalization transformer, we store its mean and std, saved to meta\_data in file name 'QM8\_meta.p';
- The real dataset has been pickled during the 'dump\_data' function call;

## Features saved in dataset

```
In [6]: dataset = pickle.load(open('test/QM8_preprocess_10-data_0000000.p', 'rb'
...: ))
In [7]: list(dataset.keys())
Out[7]:
['node_feat',
 'L_multi',
 'L_simple_4',
 'L_simple_6',
 'L_simple_7',
 'D_simple',
 'V_simple',
 'D_multi',
 'V_multi',
 'label',
 'label_weight']
```

Table X. Feature of dataset.

Feature name	Dimension	Sample
'node_feat'	(25,)	array([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29])
'L_multi'	(25, 25, 6)	array([[[0.2, 1. , 1. , 1. , 1. , 1. ], [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. ]], ...)
'L_simple_4'	(25, 25)	array([[0.2, 0.2, 0., 0., 0., 0., 0., 0., 0., 0., 0.31622777, 0.31622777, 0.31622777, 0., 0., 0., 0., 0., 0., 0., 0., 0.], ...)
'L_simple_6'	(25, 25)	
'L_simple_7'	(25, 25)	
'D_simple'	(25,)	
'V_simple'	(25, 25)	
'D_multi'	(6, 25)	
'V_multi'	(6, 25, 25)	
'label'	(1, 12)	array([[ 1.34733044, 1.0736227 , 1.01121982, 1.38320704, 0.07164872, 1.38333597, 0.82290116, 1.38329448, 1.38333597, 0.97845919, 1.50322275, 0.84904055]])
'label_weight'	(1, 12)	array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], [1., 1., 1.]])

MPNN only used  
L\_simple\_4