

---

# Chainer Chemistry Documentation

*Release 0.5.0*

Preferred Networks, Inc.

Feb 08, 2019



---

## Contents

---

<b>1</b>	<b>Features</b>
----------	-----------------

<b>3</b>
----------



Chainer Chemistry is a collection of tools to train and run neural networks for tasks in biology and chemistry using Chainer .



- State-of-the-art deep learning neural network models (especially graph convolutions) for chemical molecules (NFP, GGNN, Weave, SchNet etc.)
- Preprocessors of molecules tailored for these models
- Parsers for several standard file formats (CSV, SDF etc.)
- Loaders for several well-known datasets (QM9, Tox21 etc.)

Introductory to deep learning for molecules and Chainer Chemistry is also available [here \(SlideShare\)](#).

## 1.1 Installation

### 1.1.1 Dependency

Following packages are required to install Chainer Chemistry and are automatically installed when you install the library by *pip* command.

- `chainer`
- `pandas`
- `scikit-learn`
- `tqdm`

Also, it uses following library, which you need to manually install.

- `rdkit`

See the [official document](#) for installation. If you have setup `anaconda`, you may install `rdkit` by following command:

```
$ conda install -c rdkit rdkit
```

### 1.1.2 Install via pip

It can be installed by pip command:

```
$ pip install chainer-chemistry
```

### 1.1.3 Install from source

The tarball of the source tree is available via `pip download chainer-chemistry`. You can use `setup.py` to install Chainer Chemistry from the tarball:

```
$ tar xzf chainer-chemistry-x.x.x.tar.gz
$ cd chainer-chemistry-x.x.x
$ python setup.py install
```

Install from the latest source from the master branch:

```
$ git clone https://github.com/pfnet-research/chainer-chemistry.git
$ pip install -e chainer-chemistry
```

### 1.1.4 Run example training code

The [official repository](#) provides examples of training several graph convolution networks. The code can be obtained by cloning the repository:

```
$ git clone https://github.com/pfnet-research/chainer-chemistry.git
```

The following code is how to train Neural Fingerprint (NFP) with the Tox21 dataset on CPU:

```
$ cd chainer-chemistry/examples/tox21
$ python train_tox21.py --method=nfp --gpu=-1 # set --gpu=0 if you have GPU
```

## 1.2 Tutorial

### 1.2.1 Abstract

In this tutorial, we predict Highest Occupied Molecular Orbital (HOMO) level of the molecules in [QM9 dataset](#) [1][2] by [Neural Finger Print \(NFP\)](#) [3][4]. We concentrate on explaining usage of Chainer Chemistry briefly and do not look over the detail of NFP implementation.

### 1.2.2 Tested Environment

- Chainer Chemistry  $\geq 0.0.1$  (See [Installation](#))
- Chainer  $\geq 2.0.2$
- CUDA  $\geq 8.0$ , CuPy  $\geq 1.0.3$  (Required only when using GPU)
  - For CUDA 9.0, CuPy  $\geq 2.0.0$  is required
- sklearn  $\geq 0.17.1$  (Only for preprocessing)



### 1.2.3 QM9 Dataset

QM9 is a publicly available dataset of small organic molecule structures and their simulated properties for data driven researches of material property prediction and chemical space exploration. It contains 133,885 stable small organic molecules made up of CHONF. The available properties are geometric, energetic, electronic, and thermodynamic ones.

In this tutorial, we predict HOMO level in the properties. Physically, we need quantum chemical calculations to compute HOMO level. From mathematical viewpoint it requires a solution of an internal eigenvalue problem for a Hamiltonian matrix. It is a big challenge to predict HOMO level accurately by a neural network, because the network should approximate both calculating the Hamiltonian matrix and solving the internal eigenvalue problem.

### 1.2.4 HOMO prediction by NFP

At first you should clone the library repository from [GitHub](#). There is a Python script `examples/qm9/train_qm9.py` in the repository. It executes a whole training procedure, that is, downloads QM9 dataset, preprocess it, define an NFP model and run training on them.

Execute the following commands on a machine satisfying the tested environment in environment.

```
~$ git clone git@github.com:pfnet-research/chainer-chemistry.git
~$ cd chainer-chemistry/examples/qm9/
```

Hereafter all shell commands should be executed in this directory.

If you are a beginner for Chainer, [Chainer hands-on](#) will greatly help you. Especially the explanation of inclusion relationship of Chainer classes in Sec. 4 in [Chap. 2](#) is helpful when you read the sample script.

Next the dataset preparation part and the model definition part in `train_qm9.py` are explained. If you are not interested in them, skip [Dataset Preparation](#) and [Model Definition](#), and jump to [Run](#).

#### Dataset Preparation

Chainer Chemistry accepts the same dataset type with Chainer, such as `chainer.datasets.SubDataset`. In this section we learn how to download QM9 dataset and use it as a Chainer dataset.

The following Python script downloads and saves the dataset in `.npz` format.

```
#!/usr/bin/env python
from chainer_chemistry import datasets as D
from chainer_chemistry.dataset.preprocessors import preprocess_method_dict
from chainer_chemistry.datasets import NumpyTupleDataset

preprocessor = preprocess_method_dict['nfp']()
dataset = D.get_qm9(preprocessor, labels='homo')
cache_dir = 'input/nfp_homo/'
os.makedirs(cache_dir)
NumpyTupleDataset.save(cache_dir + 'data.npz', dataset)
```

The last two lines save the dataset to `input/nfp_homo/data.npz` and we need not to download the dataset next time.

The following Python script read the dataset from the saved `.npz` file and split the data points into training and validation sets.

```
#!/usr/bin/env python
from chainer.datasets import split_dataset_random
from chainer_chemistry import datasets as D
from chainer_chemistry.dataset.preprocessors import preprocess_method_dict
from chainer_chemistry.datasets import NumpyTupleDataset

cache_dir = 'input/nfp_homo/'
dataset = NumpyTupleDataset.load(cache_dir + 'data.npz')
train_data_ratio = 0.7
train_data_size = int(len(dataset) * train_data_ratio)
train, val = split_dataset_random(dataset, train_data_size, 777)
print('train dataset size:', len(train))
print('validation dataset size:', len(val))
```

The function `split_dataset_random()` returns a tuple of two `chainer.datasets.SubDataset` objects (training and validation set). Now you have prepared training and validation data points and you can construct `chainer.iterator.Iterator` objects, needed for updaters in Chainer.

## Model Definition

In Chainer, a neural network model is defined as a `chainer.Chain` object.

Graph convolutional networks such as NFP are generally connection of graph convolution layers and multi perceptron layers. Therefore it is convenient to define a class which inherits `chainer.Chain` and compose two `chainer.Chain` objects corresponding to the two kind of layers.

Execute the following Python script and check you can define such a class. NFP and MLP are already defined `chainer.Chain` classes.

```
#!/usr/bin/env python
import chainer
from chainer_chemistry.models import MLP, NFP

class GraphConvPredictor(chainer.Chain):

    def __init__(self, graph_conv, mlp):
        super(GraphConvPredictor, self).__init__()
        with self.init_scope():
            self.graph_conv = graph_conv
            self.mlp = mlp

    def __call__(self, atoms, adjs):
        x = self.graph_conv(atoms, adjs)
        x = self.mlp(x)
        return x

n_unit = 16
conv_layers = 4
model = GraphConvPredictor(NFP(n_unit, n_unit, conv_layers),
                           MLP(n_unit, 1))
```

## Run

You have defined the dataset and the NFP model on Chainer. There are no other procedures specific to Chainer Chemistry. Hereafter you should just follow the usual procedures in Chainer to execute training.

The sample script `examples/qm9/train_qm9.py` contains all the procedures and you can execute training just by invoking the script. The following command starts training for 20 epochs and reports loss and accuracy during training. They are reported for each of `main` (dataset for training) and `validation` (dataset for validation).

The `--gpu 0` option is to utilize a GPU with device id = 0. If you do not have a GPU, set `--gpu -1` or just drop `--gpu 0` to use CPU for all the calculation. In most cases, calculation with GPU is much faster than that only with CPU.

```
~/chainer-chemistry/examples/qm9$ python train_qm9.py --method nfp --label homo --gpu 0
↪ 0 # If GPU is unavailable, set --gpu -1

Train NFP model...
epoch      main/loss    main/accuracy  validation/main/loss  validation/main/accuracy
↪ elapsed_time
1          0.746135   0.0336724     0.680088             0.0322597
↪ 58.4605
2          0.642823   0.0311715     0.622942             0.0307055
↪ 113.748
(...)
19         0.540646   0.0277585     0.532406             0.0276445
↪ 1052.41
20         0.537062   0.0276631     0.551695             0.0277499
↪ 1107.29
```

After finished, you will find log file in `result/` directory.

## Evaluation

In the loss and accuracy report, we are mainly interested in `validation/main/accuracy`. Although it decreases during training, the `accuracy` field is actually mean absolute error. The unit is Hartree. Therefore the last line means validation mean absolute error is 0.0277499 Hartree. See `scaled_abs_error()` function in `train_qm9.py` for the detailed definition of mean absolute error.

You can also train other type models like GGNN, SchNet or WeaveNet, and other target values like LUMO, dipole moment and internal energy, just by changing `--model` and `--label` options, respectively. See output of `python train_qm9.py --help`.

## 1.2.5 Using your own dataset

You can use your own dataset in Chainer Chemistry. [example/own\\_dataset](#) shows an example.

## 1.2.6 Reference

- [1] L. Ruddigkeit, R. van Deursen, L. C. Blum, J.-L. Reymond, Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17, *J. Chem. Inf. Model.* 52, 2864–2875, 2012.
- [2] R. Ramakrishnan, P. O. Dral, M. Rupp, O. A. von Lilienfeld, Quantum chemistry structures and properties of 134 kilo molecules, *Scientific Data* 1, 140022, 2014.
- [3] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems* (pp. 2224–2232).
- [4] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

## 1.3 Contribution guide

We welcome any type of contribution that helps to improve and promote Chainer Chemistry. Typical contribution includes:

- Send pull requests (PRs) to the [repository](#) (We recommend developers making PRs to read the [Development policy](#) before starting to implement).
- Report bugs or problems as [issues](#).
- Send questions to developer community sites like [Stackoverflow](#) or Chainer Slack ([en](#), [jp](#)).
- Write a blog post about Chainer Chemistry or its use case.

## 1.4 Development policy

In this section, we describe the development policy that the core developers follow. Developers who are thinking to send PRs to the repository are encouraged to read the following sections before starting implementation.

### 1.4.1 Versioning policy

Basically, we follow the [semantic versioning v2.0.0](#). In Chainer Chemistry, *public APIs* in the sense of semantic versioning are ones in [the document](#).

We follow these rules about versioning during the major version zero in addition to ones described in the the semantic versioning:

- We do not plan any scheduled releases.
- We do not plan any pre releases.
- We release the minor version when the core development team agrees. Typically, we do so when (1) sufficient number of features are added since the last minor release (2) the latest release cannot run the example code in the master branch of the repository (3) critical bugs are found. But we are not restricted to them.
- If we find critical bugs, we should release a patch version or a minor version that fixes them. The core development team will determine which version to release.

We do not have a concrete plan about versioning strategy after v1.0.0.

### 1.4.2 Compatibility policy

As an immediate consequence of the semantic versioning, we may break compatibility of public APIs including addition, deletion, and changes in their semantics anytime in the major version zero. Since APIs of Chainer Chemistry are still immature and unstable, we expect introduction of new features can sometime involve compatibility break. If we are faced with a dilemma between cost for backward compatibility and benefit of new features, we are likely to give up the former because we want to place importance on introducing new features as soon as possible. Of course, we care backward compatibility whenever it is easy and low-cost.

Like [ChainerCV](#), Chainer Chemistry provides several off-the-shelf deep learning models (e.g. Neural Finger Print) whose papers are available in such as arXiv or conferences related to machine learning. Although, most of published papers reports evaluation results of the models with publicly available datasets, we do *NOT* guarantee the reproducibility of experiments in the papers.

At some point, coding examples in the master branch of the official repository may not work even with the latest release. In that case, users are recommended to either use the example code of the latest release or update the library code to the master branch.

As of v0.3.0, we have introduced *BaseForwardModel*, which provides methods for serializing itself to and loading from a file. As these methods internally use `pickle`, portability of the class depends on that of pickling. Especially, serialized instances of *BaseForwardModel* made with older Chainer Chemistry may not be loaded with newer one, partly because we may change their internal structures for refactoring, performance improvement, and so on. See the document of *BaseForwardModel* and their subclasses (e.g. *Classifier*, *Regressor*).

### 1.4.3 Branch strategy

The official repository of Chainer Chemistry is <https://github.com/pfnet-research/chainer-chemistry>. We use the *master* branch of the repository for development. Therefore, developer who makes PRs should send them to the master branch.

During major version zero, we do not maintain any released versions. When a bug is found, changes for the bug should be merged to the next version (either minor or patch). If the bug is critical, we will release the next version as soon as possible.

### 1.4.4 Coding guideline

We basically adopt *PEP8* <<https://www.python.org/dev/peps/pep-0008/>> as a style guide. You can check it with *flake8*, which we can install by:

```
$ pip install flake8
```

and run with `flake8` command.

In addition to PEP8, we use upper camel case (e.g. `FooBar`) for class names and snake case (e.g. `foo_bar`) for function, method, variable and package names. Although we recommend developers to follow these rules as well, they are not mandatory.

For documents, we follow the [Google Python Style Guide](#) and compile it with [Napoleon](#), which is an extension of [Sphinx](#).

### 1.4.5 Testing guideline

Chainer Chemistry uses `pytest` as a unit-test framework. All unit tests are located in `tests/` directory. We can run tests with normal usage of `pytest`. For example, the following command runs all unit tests:

```
$ pytest tests
```

Some unit tests require GPUs, which are annotated with `@pytest.mark.gpu`. Therefore, you can skip them with `-m` option:

```
$ pytest -m "not gpu" tests
```

If a developer who writes a unit test that uses GPUs, you must annotate it with `@pytest.mark.gpu`.

Similarly, some unit tests take long time to complete. We annotated them with `@pytest.mark.slow` and can skip them with `-m` option:

```
$ pytest -m "not slow" tests
```

Any unit test that uses GPUs must be annotated with `@pytest.mark.slow`.

We can skip both GPU and slow tests with the following command:

```
$ pytest -m "not (gpu or slow)" tests
```

## 1.4.6 Terminology

In the context of machine learning, especially chemoinformatics, we use several terms such as feature, feature vectors, descriptor and so on to indicate representation of inputs. To avoid disambiguity and align naming convention within the library code, we use these terms in the following way:

- *Feature* is a representation of a sample of interest (typically molecules in Chainer Chemistry).
- *Label* is a target value of we want to predict.
- *Input feature* is a representation of a sample from which we want to predict the target value.

For example, consider a supervised learning task whose dataset consisting of input-output pairs  $((x_1, y_1), \dots, (x_N, y_N))$ , where  $N$  is the number of samples. In Chainer Chemistry `x_i` and `y_i` are called input feature and label, respectively and a pair of  $(x_i, y_i)$  is feature for each  $i$ .

## 1.4.7 Relation to Chainer

[Chainer](#) is a deep learning framework written in Python that features dynamic computational graph construction (the “define-by-run” paradigm) for flexible and intuitive model development. As the name indicates, Chainer Chemistry is an extension library of Chainer built on top of it. The core development team members of Chainer and that of Chainer Chemistry work together tightly.

## 1.5 API Reference

### 1.5.1 Dataset

#### Converters

---

```
chainer_chemistry.dataset.converters.  
concat_mols
```

---

#### Indexers

---

```
chainer_chemistry.dataset.indexer.  
BaseIndexer  
chainer_chemistry.dataset.indexer.  
BaseFeatureIndexer  
chainer_chemistry.dataset.indexers.  
NumpyTupleDatasetFeatureIndexer
```

---

#### Parsers

---

```
chainer_chemistry.dataset.parsers.  
BaseParser  
chainer_chemistry.dataset.parsers.  
CSVFileParser  
chainer_chemistry.dataset.parsers.  
SDFFileParser  
chainer_chemistry.dataset.parsers.  
DataFrameParser  
chainer_chemistry.dataset.parsers.  
SmilesParser
```

---

## Preprocessors

### Base preprocessors

---

```
chainer_chemistry.dataset.  
preprocessors.BasePreprocessor  
chainer_chemistry.dataset.  
preprocessors.MolPreprocessor
```

---

### Concrete preprocessors

---

```
chainer_chemistry.  
dataset.preprocessors.  
AtomicNumberPreprocessor  
chainer_chemistry.dataset.  
preprocessors.ECFPPreprocessor  
chainer_chemistry.dataset.  
preprocessors.GGNNPreprocessor  
chainer_chemistry.dataset.  
preprocessors.NFPPreprocessor  
chainer_chemistry.dataset.  
preprocessors.SchNetPreprocessor  
chainer_chemistry.dataset.  
preprocessors.WeaveNetPreprocessor
```

---

## Utilities

---

```
chainer_chemistry.  
dataset.preprocessors.  
MolFeatureExtractionError  
chainer_chemistry.dataset.  
preprocessors.type_check_num_atoms  
chainer_chemistry.  
dataset.preprocessors.  
construct_atomic_number_array  
chainer_chemistry.dataset.  
preprocessors.construct_adj_matrix
```

---

## Splitters

---

```
chainer_chemistry.dataset.splitters.  
RandomSplitter
```

---

```
chainer_chemistry.dataset.splitters.  
StratifiedSplitter
```

---

```
chainer_chemistry.dataset.splitters.  
ScaffoldSplitter
```

---

## 1.5.2 Datasets

### Dataset implementations

---

```
chainer_chemistry.datasets.  
NumpyTupleDataset
```

---

### Dataset loaders

---

```
chainer_chemistry.datasets.tox21.  
get_tox21
```

---

```
chainer_chemistry.datasets.qm9.  
get_qm9
```

---

```
chainer_chemistry.datasets.molnet.  
get_molnet_dataset
```

---

```
chainer_chemistry.datasets.molnet.  
get_molnet_dataframe
```

---

## 1.5.3 Functions

### Function implementations

---

```
chainer_chemistry.functions.matmul
```

---

```
chainer_chemistry.functions.  
mean_squared_error
```

---

```
chainer_chemistry.functions.  
mean_absolute_error
```

---

## 1.5.4 Iterators

### Iterator Implementations

---

```
chainer_chemistry.iterators.  
BalancedSerialIterator
```

---

```
chainer_chemistry.iterators.  
IndexIterator
```

---



### 1.5.5 Links

#### Link implementations

---

<code>chainer_chemistry.links.EmbedAtomID</code>
<code>chainer_chemistry.links.GraphLinear</code>

---

### 1.5.6 Models

#### Model implementations

---

<code>chainer_chemistry.models.NFP</code>
<code>chainer_chemistry.models.GGNN</code>
<code>chainer_chemistry.models.MLP</code>
<code>chainer_chemistry.models.SchNet</code>
<code>chainer_chemistry.models.WeaveNet</code>

---

#### Wrapper models

---

<code>chainer_chemistry.models.</code> <code>BaseForwardModel</code>
<code>chainer_chemistry.models.Classifier</code>
<code>chainer_chemistry.models.Regressor</code>

---

### 1.5.7 Utilities

### 1.5.8 Training

#### Extensions

---

<code>chainer_chemistry.training.</code> <code>extensions.batch_evaluator.</code> <code>BatchEvaluator</code>
<code>chainer_chemistry.training.</code> <code>extensions.roc_auc_evaluator.</code> <code>ROCAUCEvaluator</code>
<code>chainer_chemistry.training.</code> <code>extensions.prc_auc_evaluator.</code> <code>PRCAUCEvaluator</code>

---