Wire-Cell Python

With a focus on “wcpy dnn”

Brett Viren

June 11, 2025

Topics

- Introduction to `wire-cell-python`.
- Focus on the `wirecell.dnn` module and `wcpy dnn` command.
- Specifics about how DNNROI is implemented in this mini-framework.
- Future work needed.

Reference info in backup slides:

- Packaging and installation for users and developers.

Overview of wire-cell-python

Some of the features

- `wirecell.*` Python module tree and `wcpy` command line program.
- Standard Python packaging, enabling flexible installation and development.
- Converters to produce WCT “data” configuration files.
  - response, wires, noise, ...
- Visualize and summarize WCT inputs/outputs.
  - Interoperate with the Wire-Cell Bee online event display.
  - Produce plots and images with `matplotlib`, Paraview, SVG.
- Train and evaluate Deep Neural Networks including DNNROI.
- Provide CLI tools used by some WCT unit tests.
- And more....

Command line interface(s)

The **new** unified command line interface: `wcpy`

One command to collect all “mains” from the `wirecell.*` modules as sub-commands.

- Bundles the older `wirecell-*` command line programs into one.

Why the change?

A single `wcpy` collects all the top-level help into a single `wcpy --help` command.

- Better command discovery and documentation.
- Enables more uniform conventions (eg log handling, default args).

General usage help

```
$ wcpy
Usage: wcpy [OPTIONS] COMMAND [ARGS]...

  Main wcpy

Options:
  --help  Show this message and exit.

Commands:
  aux       Commands for Wire-Cell Toolkit's "aux" package.
  bee       Wire Cell Bee helpers
  dnn       Wire Cell Deep Neural Network commands.
  gen       Wire Cell Signal Simulation Commands
  img       Wire-Cell Toolkit commands related to imaging.
  ls4gan    Wire Cell Toolkit Utility Commands
  pgraph    Wire Cell Signal Processing Features
  plot      wirecell-plot command line interface
  pytorch   wirecell-pytorch command line interface
  sigproc   Wire Cell Signal Processing
  test      Wire Cell Test Commands
  util      Wire Cell Toolkit Utility Commands
  validate  Wire Cell Validation
```

Wire Cell Python support for DNNs

A focus on the sub-command:

```
$ wcpy dnn
```

and the module:

```
wirecell.dnn
```

These provide **DNN training** and related functionality.

- Designed to accommodate a variety of **different**:
  - DNN architectures and variants
  - datasets
  - tasks
  - etc
- While providing a uniform user interface and configuration.

Initial implementation provides DNNROI.

```
$ wcpy dnn --help
Usage: wcpy dnn [OPTIONS] COMMAND [ARGS]...

  Wire Cell Deep Neural Network commands.

Options:
  -l, --log-output TEXT  log to a file [default:stdout]
  -L, --log-level TEXT   set logging level [default:info]
  -h, --help             Show this message and exit.

Commands:
  dump         Dump info about a checkpoint file.
  dump-config
  extract      Extract samples from a dataset.
  plot3p1      Plot 3 layers from first tensor and 1 image from second.
  train        Train a model.
  vizmod       Produce a text summary and if -o/--output given also a..
```

Main command: **wcpy dnn train […]**

Concept and structure of **wirecell.dnn**

Parameterized DNN model definitions: `wirecell.dnn.models.*`

Currently **UNet** in published and DNNROI variants are provided.

Support for Torch datasets: `wirecell.dnn.data.*`

Flexible, parameterized **HDF5** datasets **with caching**.

Training “engines”: `wirecell.dnn.train`

Currently one implemented: `Classifier`.

An **app** bundles network, dataset and engine: `wirecell.dnn.apps.*`

So far we have one app: **DNNROI**.

Example config files: `wirecell/dnn/cfg/*.cfg`

Specify options and local information for easy reproduction by others.

The "app" assembles the parts needed for training

A **wirecell.dnn** "app" is simply a Python module providing these attributes:

`.Network` an `torch.nn.Module` class providing the network architecture.

`.Dataset` a torch `Dataset` class representing training data.

`.Trainer` a class with `loss()`, `evaluate()` and `epoch()` methods

- *eg* `wirecell.dnn.train.Classifier`

`.Criterion` a loss function like `torch.nn.BCELoss`.

HDF5 support `wirecell.dnn.data.hdf5`

Flexible loading of training data from HDF5 files.

- Handles users' different file naming and data organization conventions.

Main classes in the `.hdf5` module:

`.Multi` a set of `Single`'s, yields a tuple of tensor,

`.Single` a `Domain` that yields a single tensor from files.

`.Domain` apply a `ReMatcher` and optional transform, implement caching

`.ReMatcher` match HDF5 file and path names to tensors with **regular expressions**.

Support for other file formats can be implemented.

dnnroi.Dataset

The DNNROI `Dataset` is an "`hdf5.Multi`".

- `Rec` and `Tru` **datasets** as `hdf5.Single`'s.
- Corresponding **data transforms**, also named `Rec` and `Tru`.
- These **match** up to specific data in users' files via regular expressions.
- Matching is **configurable**, examples for known datasets included.

dnnroi.Network

The full DNNROI Network is almost literally just this code:

```python
from wirecell.dnn.models.unet import UNet
class Network(nn.Module):

  def __init__(self):
      super().__init__()
      self.unet = UNet(n_channels=3, n_classes=1,
                       batch_norm=True, bilinear=True, padding=True)
  def forward(self, x):
      x = self.unet(x)
      return torch.sigmoid(x)
```

- Default constructor, UNet() produces the published version of UNet.
    - DNNROI variant in in/out dimensions an application of sigmoid().
- Other available constructor parameters: shape and nskips.
    - Can easily construct smaller/larger UNets.

```
$ wcpy dnn train --help
Usage: wcpy dnn train [OPTIONS] [FILES]...

  Train a model.

Options:
  -e, --epochs INTEGER            Number of epochs over which to train.  This
                                  is a relative count if the training starts
                                  with a -l/--load'ed state.
  -b, --batch INTEGER             Batch size
  -d, --device TEXT               The compute device
  --cache / --no-cache            Cache data in memory
  --debug-torch / --no-debug-torch
                                  Debug torch-level problems
  --checkpoint-save TEXT          Checkpoint path.  An {epoch} pattern can be
                                  given to use the absolute epoch number
  --checkpoint-modulus INTEGER    Checkpoint modulus.  If checkpoint path is
                                  given, the training is checkpointed ever
                                  this many epochs..
  -a, --app TEXT                  The application name
  -l, --load TEXT                 File name providing the initial model state
                                  dict (def=None - construct fresh)
  -s, --save TEXT                 File name to save model state dict after
                                  training (def=None - results not saved)
  --train-ratio FLOAT             Fraction of samples to use for training
                                  (default=1.0, no evaluation loss calculated)
  -c, --config TEXT               Set Configuration file.
  -h, --help                      Show this message and exit.
```

Example DNNROI training

Prepare

```
$ cp wirecell/dnn/cfg/hyu-pdvd.cfg my-pdvd.cfg
$ emacs my-pdvd.cfg
```

Can fully drive training with configuration file

```
$ wcpy dnn train -c my-pdvd.cfg
```

Or, **override** info from configuration file

```
$ wirecell-dnn -L debug train --device cuda --cache \
               --epochs 10 \
               --config my-pdvd.cfg \
               --checkpoint-save 'my-pdvd-{epoch}.pt' \
               --save my-pdvd.pt \
               /path/to/my/data/files/*.h5
```

Example DNNROI configuration file

```
$ ls wirecell/dnn/cfg

hyu-pdvd.cfg  renny-pdhd.cfg

$ cat  wirecell/dnn/cfg/renny-pdhd.cfg
[train]
app=dnnroi
device=gpu
files=/nfs/data/1/bviren/dnnroi/data/renney/train_data_PDHD_fixedbug_separ
batch=10
epochs=25

[dataset]
tru_path_res=[r'/(\d+)/frame_deposplat\d']
```

A single `[dataset]` config line allows matching the different conventions in the "renny" (Sergei) dataset compared to the default (Haiwang) dataset.

- This one maps info held in file path to determine type of data.

Future development

wcpy dnn is ready for use **now** for "standard" DNNROI training tasks.

Development of more features is expected

- More DNNROI configurability: chunking, rebin, crop, choice of network.
  - Core has some of this but it is not exposed to config layer yet.
  - Capture more configurations (eg, SBND/ICARUS's).
- Codify additional network architectures.
  - Eg, MobileNet.
- Add support for datasets in Zenodo
  - And upload popular DNNROI datasets to Zenodo.
- Add "apps" for other tasks (eg, GNN-based ideas, etc).
- Better documentation (as always).
- [Your favorite thing here]

If something that you want is missing, please add it!

$\mathcal{FIN}$

Packaging of `wire-cell-python`

`wire-cell-python` provides “standard” Python packaging.

- Use your “favorite method” to install/develop/use.
- Virtual env, `pip`, `uv`, etc.

Recomend to use `uv`

- `uv` is a relatively new Python packaging tool but already one of the best.

Installing uv itself - pick your favorite method.

Official method - installs to $HOME/.local/bin/

```
$ curl -LsSf https://astral.sh/uv/install.sh | sh
```

Or, do same but manually if you don't trust "curl | sh"

Download binary tar file from https://github.com/astral-sh/uv/releases, unpack, move executables to $PATH.

Or, install into your existing virtual environment

```
$ pip install uv
```

Or, if on BNL WCWC computers it's already installed

```
/usr/local/bin/uv
```

Using `uv` to install `wire-cell-python` - for end users

No commitment one-shot running

```
$ uv run --with git+https://github.com/wirecell/wire-cell-python wcpy
```

Install for easier reuse

```
$ uv tool install git+https://github.com/wirecell/wire-cell-python
$ wcpy
```

Later upgrade or removal

```
$ uv tool upgrade wirecell
$ uv tool uninstall wirecell
```

Optional dependencies

Add “`--with torch`” to “`uv run`” or “`uv tool`”.

```
$ uv tool --with torch install git+https://github.com/wirecell/wire-cel
$ wcpy dnn
```

Installing with uv - for developers

Basic setup to directly run from source

```
$ git clone git@github.com:WireCell/wire-cell-python.git
$ cd wire-cell-python
$ uv sync
$ uv run wcpy
```

Include optional dependencies (ie torch)

```
$ uv sync --all-extras
```

Activate the virtual env to avoid needing `uv run` command prefix

```
$ source .venv/bin/activate
$ wcpy
```