SLURM_gen

This package automates the process of generating large amounts of data, providing a clean interface between your simulation and the SLURM workload manager. It also manages the datasets you choose to generate, and allows easy access to cached simulations that load quickly. If you need more data than you have, SLURM_gen lets you know how many more samples need to be generated, and how much compute time it will take.

Installation

```
pip install -e .  # don't forget the period
```

Usage

SLURM_gen provides a simple command line interface to

- generate data samples,
- assign those samples to a particular dataset name, like 'train' or 'test', and

- track the number of samples generated for various datasets and parameters.

You can define your own datasets simply by writing a function that outputs feature-label pairs. Define that function in a file called `datasets.py`, and point SLURM_gen at the directory containing that file.

Example

Here we'll show how to define a simple dataset, generate some samples, and access them.

Define the generator

Start by using the `DefaultParamObject` class and the `@dataset` decorator to define a new dataset. These definitions should be placed in a Python file called `datasets.py`.

```
# example/datasets.py
import math
import random

from slurm_gen import DefaultParamObject, dataset


class NoisySineParams(DefaultParamObject):
    """Attributes defining parameters to the noisy_sine experiment."""

    # leftmost allowed value for x
    left = -1

    # rightmost allowed value for x
    right = 1

    # standard deviation of noise to add to sin(x)
    std_dev = 0.1


# we can specify extra SLURM batch parameters here
options = "--qos=test"


# here we also tell SLURM_gen to request 1GB of memory and save every 50 sample
@dataset(NoisySineParams, "1GB", 50, options)
def noisy_sine(size, params):
    """Create samples from a noisy sine wave.

    Args:
        size (int): number of samples to generate.
        params (NoisySineParams): parameters to the experiment.
    Yields:
        (float): x-value.
        (float): y-value plus noise.
    """
    for _ in range(size):
        x = random.uniform(params.left, params.right)
        yield x, math.sin(x) + random.normalvariate(mu=0, sigma=params.std_dev)
```

The `NoisySineParams` defines the possible configuration parameters that the generator can accept, as well as the default values for those parameters. When generating or accessing samples, we can specify non-default values for any of these parameters.

The `@dataset` decorator converts `noisy_sine` into a dataset which can be used by the `slurm_gen.generate` module to create cache files containing arbitrary numbers of samples. We can define as many functions as we like in `datasets.py`, and all those marked with `@dataset` will be usable in SLURM_gen.

Generate samples

Now that we've defined the generator, we can generate some samples for that dataset like this:

```
cd example/  # the directory containing datasets.py
python -m slurm_gen.generate noisy_sine -n 1000 --njobs 3 --time "10"
python -m slurm_gen.generate noisy_sine -n 1000 --njobs 3 --params "{'left': 0,
```

In the first example above, we submitted 3 SLURM jobs, splitting the 1000 samples evenly among them. Since we had no samples for this dataset yet, we had to provide `--time`. In the second example, we omitted the `--time` argument, and a time duration three standard deviations above the mean of previous runs was used, adapted to the number of samples per job. In the second example we also set some configuration parameters to non-default values.

Managing samples

We can list the available samples from the command line:

```
cd example/
python -m slurm_gen.list
```

The output will look like this:

```
noisy_sine:
Param set #0:
    left#-1|right#1| raw: 1000
        std_dev#0.1|
Param set #1:
    left#0|right#1| raw: 1000
       std_dev#0.5|
```

We can see the samples for the "noisy_sine" dataset divided into sets by the parameters given.

If we want to move some of those samples into a group labeled "train", we can do so like this:

```
cd example/
python -m slurm_gen.move noisy_sine 700 train -p 0
```

The `-p` argument identifies which parameter set to use. You can also use a dictionary of values as the identifier, by passing a string that will be evaluated as a dictionary.

After the move, the output of `python -m slurm_gen.list` will be

```
noisy_sine:
Param set #0:
    left#-1|right#1| raw: 300
        std_dev#0.1| train: unprocessed(700)
Param set #1:
    left#0|right#1| raw: 1000
       std_dev#0.5|
```

Once you've moved samples into a labeled group, you can't move them back. This is to avoid accidentally mixing samples between groups, possibly inflating the accuracy of machine learning models.

Preprocessing samples

You may have noticed that `slurm_gen.list` noted 700 "unprocessed" samples. Once samples are in a group, you can apply preprocessors to them. Preprocessors must be defined in the same `datasets.py` file. To continue the example, add the following preprocessor for our `noisy_sine` dataset.

```python
# added to datasets.py
@noisy_sine.preprocessor
def square_both(X, y):
    """Square both the inputs and the outputs."""
    return [ex ** 2 for ex in X], [wai ** 2 for wai in y]
```

Note that the preprocessor is defined for one particular dataset. If the same preprocessor needs to be defined for multiple datasets, just add the decorators one after the other.

Preprocess some samples from 'train' by running the following command:

```
python -m slurm_gen.preprocess noisy_sine square_both train 600 -p 0
```

After the data is preprocessed, the output of `python -m slurm_gen.list` will be

```
noisy_sine:
Param set #0:
    left#-1|right#1| raw: 300
                   | train: unprocessed(700)
                   |      : square_both(600)
Param set #1:
    left#0|right#1| raw: 1000
       std_dev#0.5|
```

Accessing the samples

To access the samples within Python, use the `get_dataset` function:

```python
from slurm_gen import Cache

# load those 700 samples as a training set
X, y = Cache("./example/")["noisy_sine"][0]["train"].get(700)
```

Object hierarchy

You saw in the example above that we accessed the data by indexing into the `Cache` object. Here we describe the object hierarchy used by SLURM_gen within the Python environment.

```
Cache
 '- Dataset
     '- ParamSet
         '- Group
             :- raw data, accessed with `.get()`
             '- PreprocessedData
                 '- preprocessed samples, accessed with `.get()`
```

Datasets can be indexed from a `Cache` object in the following ways:

- by name, with a string (e.g. "noisy_sine")
- by number, in order of declaration (e.g. 0)
- by the actual dataset imported from `datasets.py` (e.g. `from datasets import noisy_sine; Cache()[noisy_sine]`)

ParamSets can be indexed from a `Dataset` object in the following ways:

- by number, as printed with `python -m slurm_gen.list`
- by the string used as the directory name for the parameter set (e.g. "left#-1|right#1|std_dev#0.1")
- by the dict of parameter values (e.g. {"left": -1, "right": -1, "std_dev": 0.1})
- by a `DefaultParamObject` (e.g. `NoisySineParams(left=-1, right=-1, std_dev=0.1)`)

Groups can be indexed from a `ParamSet` object only by the string used as the name for the group (e.g. "train"). The `Group` object has a `.get()` method that unprocessed samples associated with the group. By default it returns all of them, but you can specify how many you want as a parameter.

`PreprocessedData` objects can be indexed from a `Group` object in the following ways:

- by preprocessor name (e.g. "square_both")
- by the actual preprocessor imported from `datasets.py` (e.g. `from datasets import square_both; group[square_both]`)

The `PreprocessedData` object has a `.get()` method implementing the same functionality as that of `Group`.

TODO

- Be more efficient with keeping track of the sizes of the datasets.

- Be able to preprocess on a SLURM job.

Releases 1

v0.2 Latest
on Feb 6, 2020

Packages

No packages published

Languages

● Python 100.0%