

Introduction to DataLoader and Dataset

Read through [link](#)

Common Object in DataLoader

- Sampler: Randomly choosing index per iteration. It would yield indices when `batch_size` is not `None` .
 - For `IterableDataset` , it would keep yielding None(s) per iteration using [InfiniteConstantSampler](#)
- Fetcher: Taking single index or a batch of indices, and returning corresponding data from Dataset. It would invoke `collate_fn` over each batch of data and drop the remaining unfilled batch if `drop_last` is set.
 - For `IterableDataset` , it would simply get next batch-size elements as a batch.

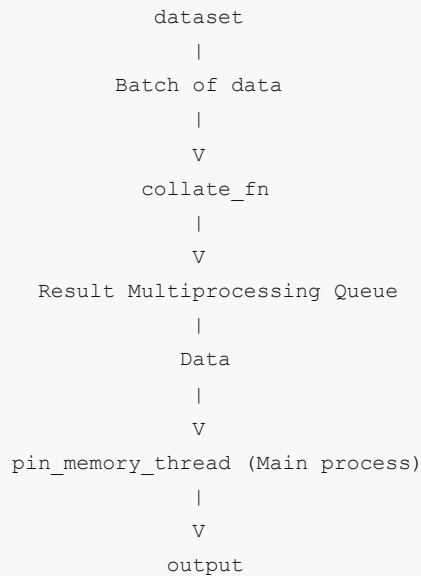
Data/Control flow in DataLoader

- Single Process:

```
Sampler
  |
index/indices
  |
  v
Fetcher
  |
index/indices
  |
  v
dataset
  |
  v
collate_fn
  |
  v
output
```

- Multiple processes:

```
Sampler (Main process)
  |
  index/indices
  |
  v
Index Multiprocessing Queue (one healthy worker)
  |
  index/indices
  |
  v
Fetcher (Worker process)
  |
  index/indices
  |
  v
```



This is just a general data and control flow in DataLoader. There are multiple further detailed functionalities like prefetching, worker_status, and etc.

Common gotchas for DataLoader

Most of common questions for DataLoader come from multiple workers as multiprocessing is enabled.

- Default multiprocessing methods are different across platforms based on Python (<https://docs.python.org/3/library/multiprocessing.html#contexts-and-start-methods>)
 - Control randomness per worker using `worker_init_fn`. Otherwise, DataLoader either becomes non-deterministic when using spawn or shares same random state for each worker when using fork.
 - COW in fork (Copy-on-access in Python fork). The simplest solution for the implementation of Dataset is to use Tensor or NumPy array to replace Python arbitrary objects like list and dict.
- Difference between Map-style Dataset and Iterable-style Dataset
 - Map-style Dataset can utilize the indices sampled from main process to get automatic sharding.
 - Iterable-style Dataset requires users to manually implement sharding inside `__iter__` method using `torch.utils.data.get_worker_info()`. Please check the [example](#).
- Shuffle is not enabled for Iterable-style Dataset. If needed, users need to implement the shuffle utilities inside `IterableDataset` class. (This is solved by TorchData project)

Introduction to next-generation Data API (TorchData)

Read through [link](#) and [link](#) Expected features:

- Automatic/Dynamic sharding
- Determinism Control
- Snapshotting
- DataFrame integration
- etc.

Lab for DataLoader and DataPipe

Goto N1222094 for Data Lab