torch.utils.data

At the heart of PyTorch data loading utility is the :class:`torch.utils.data.DataLoader` class. It represents a Python iterable over a dataset, with support for

- map-style and iterable-style datasets,
- `customizing data loading order <Data Loading Order and Sampler_>`_,
- automatic batching,
- single- and multi-process data loading,
- automatic memory pinning.

These options are configured by the constructor arguments of a :class:`~torch.utils.data.DataLoader`, which has signature:

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

The sections below describe in details the effects and usages of these options.

Dataset Types

The most important argument of :class:`~torch.utils.data.DataLoader` constructor is :attr:`dataset`, which indicates a dataset object to load data from. PyTorch supports two different types of datasets:

- map-style datasets,
- iterable-style datasets.

Map-style datasets

A map-style dataset is one that implements the :meth:`__getitem__` and :meth:`__len__` protocols, and represents a map from (possibly non-integral) indices/keys to data samples.

For example, such a dataset, when accessed with `dataset[idx]`, could read the `idx`-th image and its corresponding label from a folder on the disk.

See :class:`~torch.utils.data.Dataset` for more details.

Iterable-style datasets

An iterable-style dataset is an instance of a subclass of :class:`~torch.utils.data.IterableDataset` that implements the :meth:`__iter__` protocol, and represents an iterable over data samples. This type of datasets is particularly suitable for cases where random reads are expensive or even improbable, and where the batch size depends on the fetched data.

For example, such a dataset, when called `iter(dataset)`, could return a stream of data reading from a database, a remote server, or even logs generated in real time.

See :class:`~torch.utils.data.IterableDataset` for more details.

> **Note**
>
> When using an :class:`~torch.utils.data.IterableDataset` with multi-process data loading. The same dataset object is replicated on each worker process, and thus the replicas must be configured differently to avoid duplicated data. See :class:`~torch.utils.data.IterableDataset` documentations for how to achieve this.
>
>

Data Loading Order and :class:`~torch.utils.data.Sampler`

For iterable-style datasets, data loading order is entirely controlled by the user-defined iterable. This allows easier implementations of chunk-reading and dynamic batch size (e.g., by yielding a batched sample at each time).

The rest of this section concerns the case with map-style datasets. :class:`torch.utils.data.Sampler` classes are used to specify the sequence of indices/keys used in data loading. They represent iterable objects over the indices to datasets. E.g., in the common case with stochastic gradient decent (SGD), a :class:`~torch.utils.data.Sampler` could randomly permute a list of indices and yield each one at a time, or yield a small number of them for mini-batch SGD.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 83); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 83); *backlink*
>
> Unknown interpreted text role "class".

A sequential or shuffled sampler will be automatically constructed based on the :attr:`shuffle` argument to a :class:`~torch.utils.data.DataLoader`. Alternatively, users may use the :attr:`sampler` argument to specify a custom :class:`~torch.utils.data.Sampler` object that at each time yields the next index/key to fetch.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 92); *backlink*
>
> Unknown interpreted text role "attr".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 92); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 92); *backlink*
>
> Unknown interpreted text role "attr".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 92); *backlink*
>
> Unknown interpreted text role "class".

A custom :class:`~torch.utils.data.Sampler` that yields a list of batch indices at a time can be passed as the :attr:`batch_sampler` argument. Automatic batching can also be enabled via :attr:`batch_size` and :attr:`drop_last` arguments. See the next section for more details on this.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 97); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 97); *backlink*
>
> Unknown interpreted text role "attr".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 97); *backlink*
>
> Unknown interpreted text role "attr".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 97); *backlink*
>
> Unknown interpreted text role "attr".

Note

Neither :attr:`sampler` nor :attr:`batch_sampler` is compatible with iterable-style datasets, since such datasets have no notion of a key or an index.

Loading Batched and Non-Batched Data

:class:`~torch.utils.data.DataLoader` supports automatically collating individual fetched data samples into batches via arguments :attr:`batch_size`, :attr:`drop_last`, and :attr:`batch_sampler`.

Automatic batching (default)

This is the most common case, and corresponds to fetching a minibatch of data and collating them into batched samples, i.e., containing Tensors with one dimension being the batch dimension (usually the first).

When :attr:`batch_size` (default 1) is not None, the data loader yields batched samples instead of individual samples. :attr:`batch_size` and :attr:`drop_last` arguments are used to specify how the data loader obtains batches of dataset keys. For map-style datasets, users can alternatively specify :attr:`batch_sampler`, which yields a list of keys at a time.

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Note

The :attr:`batch_size` and :attr:`drop_last` arguments essentially are used to construct a :attr:`batch_sampler` from :attr:`sampler`. For map-style datasets, the :attr:`sampler` is either provided by user or constructed based on the :attr:`shuffle` argument. For iterable-style datasets, the :attr:`sampler` is a dummy infinite one. See `this section <Data Loading Order and Sampler_>`_ on more details on samplers.

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Unknown interpreted text role "attr".

Note

When fetching from iterable-style datasets with multi-processing, the :attr:`drop_last` argument drops the last non-full batch of each worker's dataset replica.

After fetching a list of samples using the indices from sampler, the function passed as the :attr:`collate_fn` argument is used to collate lists of samples into batches.

In this case, loading from a map-style dataset is roughly equivalent with:

```
for indices in batch_sampler:
    yield collate_fn([dataset[i] for i in indices])
```

and loading from an iterable-style dataset is roughly equivalent with:

```
dataset_iter = iter(dataset)
for indices in batch_sampler:
    yield collate_fn([next(dataset_iter) for _ in indices])
```

A custom :attr:`collate_fn` can be used to customize collation, e.g., padding sequential data to max length of a batch. See this section on more about :attr:`collate_fn`.

Disable automatic batching

In certain cases, users may want to handle batching manually in dataset code, or simply load individual samples. For example, it could be cheaper to directly load batched data (e.g., bulk reads from a database or reading continuous chunks of memory), or the batch size is data dependent, or the program is designed to work on individual samples. Under these scenarios, it's likely better to not use automatic batching (where :attr:`collate_fn` is used to collate the samples), but let the data loader directly return each member of the :attr:`dataset` object.

When both :attr:`batch_size` and :attr:`batch_sampler` are None (default value for :attr:`batch_sampler` is already None), automatic batching is disabled. Each sample obtained from the :attr:`dataset` is processed with the function passed as the :attr:`collate_fn` argument.

When automatic batching is disabled, the default :attr:`collate_fn` simply converts NumPy arrays into PyTorch Tensors, and keeps everything else untouched.

In this case, loading from a map-style dataset is roughly equivalent with:

```
for index in sampler:
    yield collate_fn(dataset[index])
```

and loading from an iterable-style dataset is roughly equivalent with:

```
for data in iter(dataset):
    yield collate_fn(data)
```

See this section on more about :attr:`collate_fn`.

Working with :attr:`collate_fn`

The use of :attr:`collate_fn` is slightly different when automatic batching is enabled or disabled.

When automatic batching is disabled, :attr:`collate_fn` is called with each individual data sample, and the output is yielded from the data loader iterator. In this case, the default :attr:`collate_fn` simply converts NumPy arrays in PyTorch tensors.

When automatic batching is enabled, :attr:`collate_fn` is called with a list of data samples at each time. It is expected to collate the input samples into a batch for yielding from the data loader iterator. The rest of this section describes behavior of the default :attr:`collate_fn` in this case.

For instance, if each data sample consists of a 3-channel image and an integral class label, i.e., each element of the dataset returns a tuple (`image`, `class_index`), the default :attr:`collate_fn` collates a list of such tuples into a single tuple of a batched image tensor and a batched class label Tensor. In particular, the default :attr:`collate_fn` has the following properties:

- It always prepends a new dimension as the batch dimension.
- It automatically converts NumPy arrays and Python numerical values into PyTorch Tensors.
- It preserves the data structure, e.g., if each sample is a dictionary, it outputs a dictionary with the same set of keys but batched Tensors as values (or lists if the values can not be converted into Tensors). Same for `list`s, `tuple`s, `namedtuple`s, etc.

Users may use customized :attr:`collate_fn` to achieve custom batching, e.g., collating along a dimension other than the first, padding sequences of various lengths, or adding support for custom data types.

Single- and Multi-process Data Loading

A :class:`~torch.utils.data.DataLoader` uses single-process data loading by default.

Within a Python process, the Global Interpreter Lock (GIL) prevents true fully parallelizing Python code across threads. To avoid blocking computation code with data loading, PyTorch provides an easy switch to perform multi-process data loading by simply setting the argument :attr:`num_workers` to a positive integer.

Single-process data loading (default)

In this mode, data fetching is done in the same process a :class:`~torch.utils.data.DataLoader` is initialized. Therefore, data loading may block computing. However, this mode may be preferred when resource(s) used for sharing data among processes (e.g., shared memory, file descriptors) is limited, or when the entire dataset is small and can be loaded entirely in memory. Additionally, single-process loading often shows more readable error traces and thus is useful for debugging.

Multi-process data loading

Setting the argument :attr:`num_workers` as a positive integer will turn on multi-process data loading with the specified number of loader worker processes.

Warning

After several iterations, the loader worker processes will consume the same amount of CPU memory as the parent process for all Python objects in the parent process which are accessed from the worker processes. This can be problematic if the Dataset contains a lot of data (e.g., you are loading a very large list of filenames at Dataset construction time) and/or you are using a lot of workers (overall memory usage is `number of workers * size of parent process`). The simplest workaround is to replace Python objects with non-refcounted representations such as Pandas, Numpy or PyArrow objects. Check out issue #13246 for more details on why this occurs and example code for how to workaround these problems.

In this mode, each time an iterator of a :class:`~torch.utils.data.DataLoader` is created (e.g., when you call `enumerate(dataloader)`), :attr:`num_workers` worker processes are created. At this point, the :attr:`dataset`, :attr:`collate_fn`, and :attr:`worker_init_fn` are passed to each worker, where they are used to initialize, and fetch data. This means that dataset access together with its internal IO, transforms (including :attr:`collate_fn`) runs in the worker process.

:func:`torch.utils.data.get_worker_info()` returns various useful information in a worker process (including the worker id, dataset replica, initial seed, etc.), and returns `None` in main process. Users may use this function in dataset code and/or :attr:`worker_init_fn` to individually configure each dataset replica, and to determine whether the code is running in a worker process. For example, this can be particularly helpful in sharding the dataset.

For map-style datasets, the main process generates the indices using :attr:`sampler` and sends them to the workers. So any shuffle randomization is done in the main process which guides loading by assigning indices to load.

For iterable-style datasets, since each worker process gets a replica of the :attr:`dataset` object, naive multi-process loading will often result in duplicated data. Using :func:`torch.utils.data.get_worker_info()` and/or :attr:`worker_init_fn`, users may configure each replica independently. (See :class:`~torch.utils.data.IterableDataset` documentations for how to achieve this.) For similar reasons, in multi-process loading, the :attr:`drop_last` argument drops the last non-full batch of each worker's iterable-style dataset replica.

Workers are shut down once the end of the iteration is reached, or when the iterator becomes garbage collected.

Warning

It is generally not recommended to return CUDA tensors in multi-process loading because of many subtleties in using CUDA and sharing CUDA tensors in multiprocessing (see :ref:`multiprocessing-cuda-note`). Instead, we recommend using automatic memory pinning (i.e., setting :attr:`pin_memory=True`), which enables fast data transfer to CUDA-enabled GPUs.

line 314); *backlink*

Unknown interpreted text role "attr".

Platform-specific behaviors

Since workers rely on Python :py:mod:`multiprocessing`, worker launch behavior is different on Windows compared to Unix.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 324);** *backlink*

Unknown interpreted text role "py:mod".

- On Unix, :func:`fork()` is the default :py:mod:`multiprocessing` start method. Using :func:`fork`, child workers typically can access the :attr:`dataset` and Python argument functions directly through the cloned address space.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 327);** *backlink*

  Unknown interpreted text role "func".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 327);** *backlink*

  Unknown interpreted text role "py:mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 327);** *backlink*

  Unknown interpreted text role "func".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 327);** *backlink*

  Unknown interpreted text role "attr".

- On Windows or MacOS, :func:`spawn()` is the default :py:mod:`multiprocessing` start method. Using :func:`spawn()`, another interpreter is launched which runs your main script, followed by the internal worker function that receives the :attr:`dataset`, :attr:`collate_fn` and other arguments through :py:mod:`pickle` serialization.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 331);** *backlink*

  Unknown interpreted text role "func".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 331);** *backlink*

  Unknown interpreted text role "py:mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line 331);** *backlink*

  Unknown interpreted text role "func".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`**, line**

This separate serialization means that you should take two steps to ensure you are compatible with Windows while using multi-process data loading:

- Wrap most of you main script's code within `if __name__ == '__main__':` block, to make sure it doesn't run again (most likely generating error) when each worker process is launched. You can place your dataset and :class:`~torch.utils.data.DataLoader` instance creation logic here, as it doesn't need to be re-executed in workers.

- Make sure that any custom :attr:`collate_fn`, :attr:`worker_init_fn` or :attr:`dataset` code is declared as top level definitions, outside of the `__main__` check. This ensures that they are available in worker processes. (this is needed since functions are pickled as references only, not `bytecode`.)

Randomness in multi-process data loading

By default, each worker will have its PyTorch seed set to `base_seed + worker_id`, where `base_seed` is a long generated by main process using its RNG (thereby, consuming a RNG state mandatorily) or a specified :attr:`generator`. However, seeds for other libraries may be duplicated upon initializing workers, causing each worker to return identical random numbers. (See :ref:`this section <dataloader-workers-random-seed>` in FAQ.).

In :attr:`worker_init_fn`, you may access the PyTorch seed set for each worker with either
:func:`torch.utils.data.get_worker_info().seed <torch.utils.data.get_worker_info>` or :func:`torch.initial_seed()`, and use it to seed
other libraries before data loading.

Memory Pinning

Host to GPU copies are much faster when they originate from pinned (page-locked) memory. See :ref:`cuda-memory-pinning` for
more details on when and how to use pinned memory generally.

For data loading, passing :attr:`pin_memory=True` to a :class:`~torch.utils.data.DataLoader` will automatically put the fetched data
Tensors in pinned memory, and thus enables faster data transfer to CUDA-enabled GPUs.

The default memory pinning logic only recognizes Tensors and maps and iterables containing Tensors. By default, if the pinning logic
sees a batch that is a custom type (which will occur if you have a :attr:`collate_fn` that returns a custom batch type), or if each element
of your batch is a custom type, the pinning logic will not recognize them, and it will return that batch (or those elements) without
pinning the memory. To enable memory pinning for custom batch or data type(s), define a :meth:`pin_memory` method on your
custom type(s).

See the example below.

Example:

```
class SimpleCustomBatch:
    def __init__(self, data):
```

```
        transposed_data = list(zip(*data))
        self.inp = torch.stack(transposed_data[0], 0)
        self.tgt = torch.stack(transposed_data[1], 0)

    # custom memory pinning method on custom type
    def pin_memory(self):
        self.inp = self.inp.pin_memory()
        self.tgt = self.tgt.pin_memory()
        return self

def collate_wrapper(batch):
    return SimpleCustomBatch(batch)

inps = torch.arange(10 * 5, dtype=torch.float32).view(10, 5)
tgts = torch.arange(10 * 5, dtype=torch.float32).view(10, 5)
dataset = TensorDataset(inps, tgts)

loader = DataLoader(dataset, batch_size=2, collate_fn=collate_wrapper,
                    pin_memory=True)

for batch_ndx, sample in enumerate(loader):
    print(sample.inp.is_pinned())
    print(sample.tgt.is_pinned())
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 417)

Unknown directive type "autoclass".

```
    .. autoclass:: DataLoader
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 418)

Unknown directive type "autoclass".

```
    .. autoclass:: Dataset
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 419)

Unknown directive type "autoclass".

```
    .. autoclass:: IterableDataset
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 420)

Unknown directive type "autoclass".

```
    .. autoclass:: TensorDataset
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 421)

Unknown directive type "autoclass".

```
    .. autoclass:: ConcatDataset
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 422)

Unknown directive type "autoclass".

```
    .. autoclass:: ChainDataset
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, line 423)

Unknown directive type "autoclass".

```
.. autoclass:: Subset
```

Unknown directive type "autofunction".

```
.. autofunction:: torch.utils.data.default_collate
```

Unknown directive type "autofunction".

```
.. autofunction:: torch.utils.data.default_convert
```

Unknown directive type "autofunction".

```
.. autofunction:: torch.utils.data.get_worker_info
```

Unknown directive type "autofunction".

```
.. autofunction:: torch.utils.data.random_split
```

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.Sampler
```

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.SequentialSampler
```

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.RandomSampler
```

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.SubsetRandomSampler
```

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.WeightedRandomSampler
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 433**)

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.BatchSampler
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 434**)

Unknown directive type "autoclass".

```
.. autoclass:: torch.utils.data.distributed.DistributedSampler
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 438**)

Unknown directive type "py:module".

```
.. py:module:: torch.utils.data.communication
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 442**)

Unknown directive type "py:module".

```
.. py:module:: torch.utils.data.datapipes
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 443**)

Unknown directive type "py:module".

```
.. py:module:: torch.utils.data.datapipes.dataframe
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 444**)

Unknown directive type "py:module".

```
.. py:module:: torch.utils.data.datapipes.iter
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 445**)

Unknown directive type "py:module".

```
.. py:module:: torch.utils.data.datapipes.map
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]data.rst`, **line 446**)

Unknown directive type "py:module".

```
.. py:module:: torch.utils.data.datapipes.utils
```

Docutils System Messages

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-`
`master\docs\source\[pytorch-master][docs][source]data.rst`**, line 11)**

Indirect hyperlink target "customizing data loading order" refers to target "data loading order and sampler", which does not exist.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-`
`master\docs\source\[pytorch-master][docs][source]data.rst`**, line 131)**

Indirect hyperlink target "this section" refers to target "data loading order and sampler", which does not exist.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-`
`master\docs\source\[pytorch-master][docs][source]data.rst`**, line 11); *backlink***

Unknown target name: "data loading order and sampler".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-`
`master\docs\source\[pytorch-master][docs][source]data.rst`**, line 131); *backlink***

Unknown target name: "data loading order and sampler".