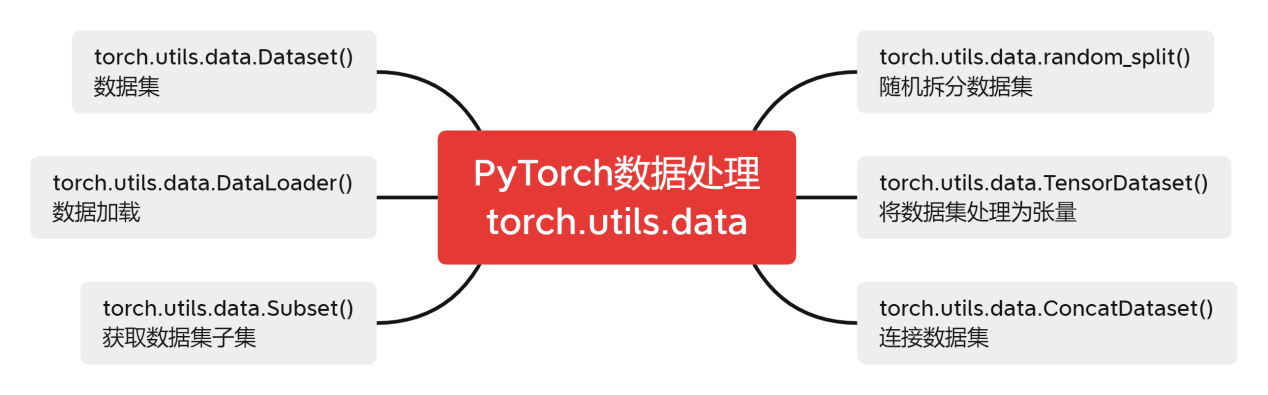
**Pytorch实现卷积神经网络五-数据加载**

数据是整个项目的基础，没有数据任何的深度学习项目都是没有灵魂的。因此我们将使用几篇文章来讲述一下在PyTorch中如何进行数据的加载处理和使用的。考虑到PyTorch中相关的模块数据比较多，因此我们先从PyTorch如何进行非图像数据的加载开始进行讲述。

**一、常用的数据操作方法**

在PyTorch中，数据加载和处理的模块主要是依靠torch.ultis.data来完成，torch.ultis.data模块中主要有以下的类：

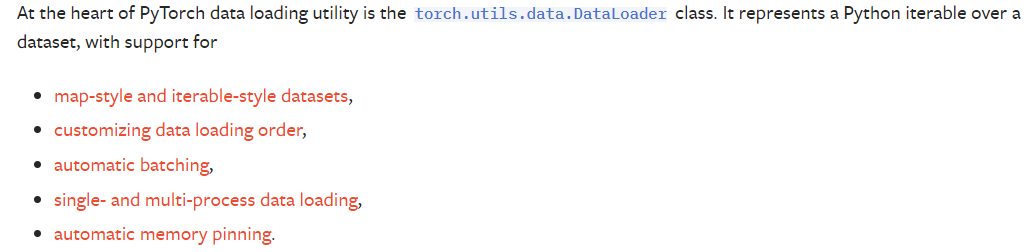


**首先PyTorch的数据加载模块，一共涉及到Dataset，Sampler，Dataloader三个类，它们负责不同的任务：**

1. Dataset负责对原始数据源进行封装，将其封装成Python可识别的数据结构，提供提取数据接口。Dataset共有Map-style datasets和Iterable-style datasets两种。
2. Sampler（采样器）决定采样方式，其负责提供一种提取DataSet数据集所有元素索引的策略。
3. Dataloader负责加载数据，同时支持map-style和iterable-style Dataset，支持单进程/多进程，支持设置loading order, batch size, pin memory等加载参数。

总结即为Dataloader负责总的调度，命令Sampler定义遍历索引的方式，然后用索引去Dataset中提取元素，实现了对给定数据集的遍历。

在官网中有这么一段总结：



什么意思呢？就是PyTorch中数据加载的核心是torch.ultis.data.DataLoader()这个类，这个类的声明在我们安装的这个路径中可以进行查看：

...\site-packages\torch\utils\data\dataloader.py

torch.ultis.data.DataLoader()其是一种Python形式的可进行迭代的数据集类，支持：

（1）map-style datasets和iterable-style datasets

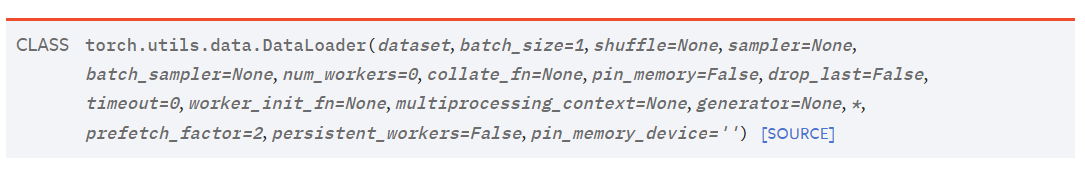
（2）自定义数据加载顺序

（3）自动把数据整理成batch序列

（4）单线程和多线程读取处理

（5）自动内存锁页

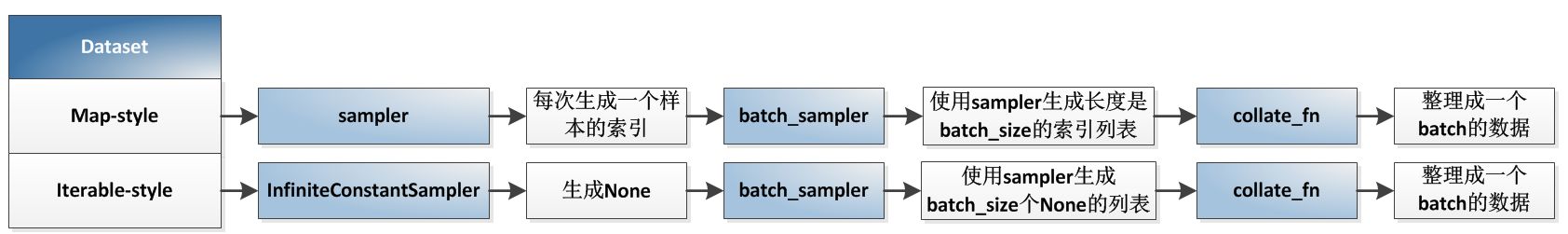
那么上述5个支持项到底支持了什么呢？我们简单来看一下，首先我们给出DataLoader()类的接口定义：



**主要参数的含义如下：**

|  |  |
| --- | --- |
| **参数名称** | **参数含义** |
| dataset | 加载的数据集。 |
| batch\_size | 每一个batch加载的样本数量。 |
| shuffle | 为True时将使用RandomSample进行随机索引。 |
| sampler | 自定义从数据集中取样本的策略，如果指定这个参数，那么shuffle必须为False |
| batch\_sampler | 定义从数据集中加载数据所采用的策略，如果指定的话，shuffle必须为False；batch\_sample类似，表示一次返回一个batch的index。 |
| num\_workers | 表示开启多少个线程去加载数据，默认为0表示使用主程序。 |
| collate\_fn | 将一个list的sample组成一个mini-batch的函数。 |
| pin\_memory | 如果为True，则DataLoader 在将张量返回之前将其复制到 CUDA 固定的内存中。 |
| drop\_last | 如果数据集大小不能被批处理大小整除，设置为True将删除最后一个未完成的批处理。如果为False且数据集的大小不能被批处理大小整除，则最后一批将更小。 |
| timeout | 如果为正值，则从workers处收集批处理的超时值，应始终为非负数，超过这个时间还没读取到数据的话就会报错。 |
| worker\_init\_fn | 如果不是None，将对每个worker子进程使用worker id ( [0, num\_workers - 1])作为输入调用。 |
| prefetch\_factor | 每个worker提前加载的sample数量。 |
| persistent\_workers | 如果为True，数据加载器将不会在数据集被使用一次后关闭工作进程。 |

这里参考知乎博主：Fang Suk的关于pytorch读取数据的整体处理流程图：



作者总结到：

（1）先用采样器采样，采样一次得到一个样本的索引（iterable-style dataset不能通过索引取值，所以使用的是一个虚假的采样器，每次生成None）。

（2）使用batch\_sampler生成长度为batch\_size的索引列表（实际是使用sampler采样batch\_size次）。

（3）使用collate\_fn将batch\_size长度的列表整理成batch样本（tensor格式）。

**1.1 map-style\iterable-style datasets**

主要涉及到的参数是 dataset。

按照官方的说法，map-style是一种实现了\_\_getitem()\_\_和\_\_len()\_\_协议的数据集，代表的是索引/键向数据集样本的映射。本质上来说，数据集本身在一开始是没有读入程序中，在需要使用数据集的时候，通过索引或键的方式从硬盘中进行读取。访问 dataset[idx] 即访问 idx 对应的数据。比如说dataset[5]，就表示第5个样本。

另按照官方的说法，iterable-style dataset可迭代型数据集是torch.utils.data.IterableDataset的子类，实现了\_\_iter\_\_()协议，表示对数据样本的一轮迭代。该数据集类似python的可迭代对象，使用iter()方法会得到一个迭代器，每次调用next()会得到下一个样本。这种数据集无法使用索引的方式获取元素，在dataloader的实现中，对于可迭代类型的数据集会使用一个虚假采样器InfiniteConstantSampler，每次调用都返回None。

**1.2 自定义数据加载顺序**

主要涉及的参数为shuffle, sampler, batch\_sampler, collate\_fn。

torch.utils.data.Sampler类用于指定数据加载中使用的索引/键的顺序。它们代表数据集索引上的可迭代对象。

对于map-style数据集，sampler可以在数据读取时指定读取的顺序。

对于iterable-style数据集，数据加载顺序完全由用户定义的iterable控制。

值得注意的是顺序或随机采样器将根据 [DataLoader](https://pytorch.org/docs/1.12/data.html" \l "torch.utils.data.DataLoader" \o "torch.utils.data.DataLoader) 的参数自动构造，即通过指定DataLoader的shuffle参数来确定是顺序读取还是乱序读取。

1. 如果shuffle=True，DotaLoader会自动构造一个RandomSampler采样器。
2. 如果shuffle=False，会构造SequentialSample采样器。

**采样器返回的是样本的索引，而不是真正的数据。**

**1.3 自动把数据整理成batch序列**

主要涉及的参数为batch\_size, batch\_sampler, collate\_fn, drop\_last。

官网介绍到，DataLoader支持通过参数batch\_size、drop\_last、batch\_sampler和collate\_fn自动批量获取的数据样本。

根据DataLoader()类的定义可以看出参数**batch\_size**(默认值1)不是None，此时dataloader默认返回批量的样本（batch\_size默认为1）。Batch\_size和drop\_last参数用于指定数据加载器如何获取批量数据集键。对于map-style的数据集，用户也可以指定batch\_sampler，其一次返回批量大小的索引列表。

当参数batch\_size和batch\_sample均为None时，会关闭自动批量化操作。此时会将采样的单个样本传给collate\_fn函数，此时DataLoader 使用 collate\_fn 参数将样本列表整理成batch。

**1.4 单线程和多线程读取处理**

主要涉及的参数为：主要涉及到的参数有 num\_workers, worker\_init\_fn

这个比较简单，为了避免在加载数据时阻塞计算代码，PyTorch 提供了一个简单的开关，只需将参数设置 num\_workers 为正整数即可执行多进程数据加载，设置为 0 时执行单线程数据加载。

至于其中很多的细节，如torch.utils.data.get\_worker\_info()的作用大家可以参考官方文档进行进一步的学习。

**1.5自动内存锁页**

主要涉及到的参数为: pin\_memory

pin\_memory是指锁页内存，内存分为锁页和不锁页，锁页内存存的东西在任何情况下都不会与机器的虚拟内存（虚拟内存就是硬盘）进行交换。不锁页内存在主机内存不足时，数据会存放到虚拟内存。

这就显然得到一个结论所谓的锁页内存就是说，我们不允许系统将某些内存里的数据交换至虚拟内存，毋庸置疑这将会提升进程的运行速度。但是也会是内存的存储占用消耗很多。

如果pin\_memory=True，那么生成的数据都会放在锁页内存上，此时将tensor拷贝到GPU的显存会更快，这个参数默认是Flase。

另外DataLoader 通过指定 prefetch\_factor来进行数据的预取，这里就不展开了。

1. **总结**

本次文章介绍了pytorch数据读取的核心是torch.utils.data.DataLoader类具有的特性：

（1）支持map-style datasets和iterable-style datasets

（2）自定义数据读取顺序

（3）自动批量化

（4）单线程/多线程读取

（5）自动内存锁页

并简单的介绍了这些知识点，下次文章中我们一起来看看torch.utils.data模块中的类，看看实际上PyTorch加载数据是怎样进行操作的。

1. **参考文档**

https://pytorch.org/docs/1.12/

<https://zhuanlan.zhihu.com/p/270028097>

<https://zhuanlan.zhihu.com/p/357847233>

https://blog.csdn.net/g11d111/article/details/81504637