实验5 CNN识别MNIST数据集

南京信息工程大学
计算机学院

应龙

2024年 秋季

实验内容和要求

实验内容：

- 安装 pytorch + torchvision 环境。
- 离线下载 MNIST 数据集。
- 使用 pytorch + torchvision 处理和加载下载到本地的 MNIST 数据集。
- 使用 CNN 对 MNIST 数据集中的手写字符进行分类。

要求：

- 格式同前面的实验。
- 写出程序主要功能模块和算法流程，总结各函数的功能和输入输出，说明所使用的CNN网络结构，阐述认为重要的算法功能模块设计实现的思想。
- 算法源代码可运行。在算法源代码中进行必要的注释，对实现算法、各个函数的功能和输入输出、重要变量的作用进行说明。
- 实验结果: (1) 绘制训练阶段的损失变化曲线; (2)记录每个epoch 测试集上的平均 loss 和 Accuracy; (3)多分类评价指标 Accuracy, macro-F1。

MNIST 数据集

MNIST 数据集可在
http://yann.lecun.com/exdb/mnist/
获取, 它包含四个部分:
Training set images: train-images-idx3-ubyte.gz (9.9 MB, 解压后 47 MB, 包含 60,000 个样本)
Training set labels: train-labels-idx1-ubyte.gz (29 KB, 解压后 60 KB, 包含 60,000 个标签)
Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 解压后 7.8 MB, 包含 10,000 个样本)
Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 解压后 10 KB, 包含 10,000 个标签)

MNIST 数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST). 训练集 (training set) 由来自 250 个不同人手写的数字构成, 其中 50% 是高中学生, 50% 来自人口普查局 (the Census Bureau) 的工作人员. 测试集(test set) 也是同样比例的手写数字数据.

MNIST 数据集

CLASS torchvision.datasets.MNIST(root: str, train: bool = True, transform: Optional[Callable] = None, target_transform: Optional[Callable] = None, download: bool = False)[SOURCE]

Parameters:

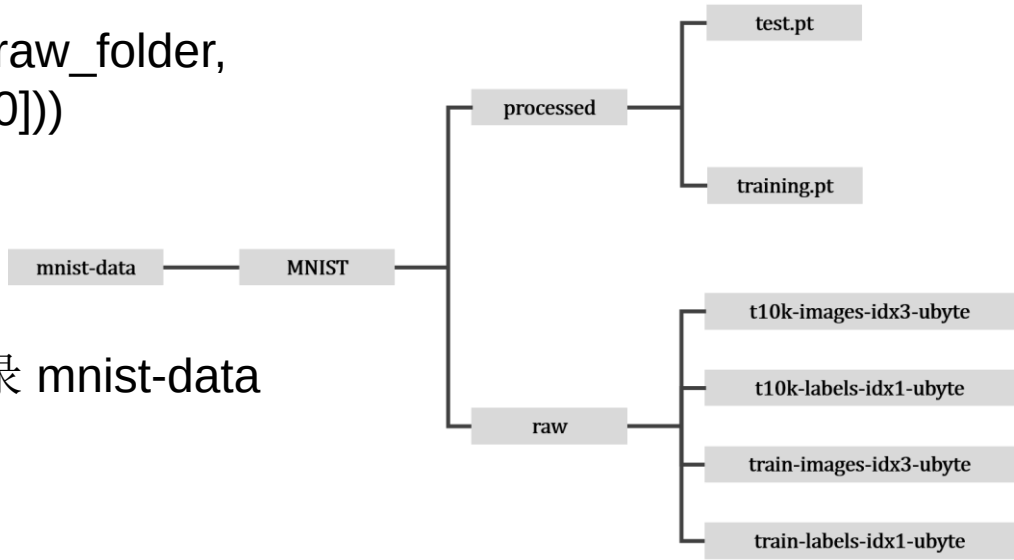
- root (string) – Root directory of dataset where MNIST/raw/train-images-idx3-ubyte and MNIST/raw/t10k-images-idx3-ubyte exist.
- train (bool, optional) – If True, creates dataset from train-images-idx3-ubyte, otherwise from t10k-images-idx3-ubyte.
- download (bool, optional) – If True, downloads the dataset from the internet and puts it in root directory. If dataset is already downloaded, it is not downloaded again.
- transform (callable, optional) – A function/transform that takes in an PIL image and returns a transformed version. E.g, transforms.RandomCrop
- target_transform (callable, optional) – A function/transform that takes in the target and transforms it.

MNIST 数据集

SOURCE CODE FOR TORCHVISION.DATASETS.MNIST

https://pytorch.org/vision/stable/_modules/torchvision/datasets/mnist.html#MNIST

```
    @property
    def raw_folder(self) -> str:
        return os.path.join(self.root, self.__class__.__name__, "raw")
    @property
    def processed_folder(self) -> str:
        return os.path.join(self.root, self.__class__.__name__, "processed")
    @property
 def _check_exists(self) -> bool:
        return all(
            check_integrity(os.path.join(self.raw_folder,
os.path.splitext(os.path.basename(url))[0]))
            for url, _ in self.resources
        )
```



根目录 mnist-data

Pytorch Dataset

torch.utils.data.Dataset 是一个表示数据集的抽象类。任何自定义的数据集都需要继承这个类并覆写相关方法。
所谓数据集，其实就是一个负责处理索引(index)到样本(sample)映射的一个类(class)。
Pytorch提供两种数据集： Map式数据集 Iterable式数据集

- Map式数据集

一个Map式的数据集必须要重写getitem(self, index),len(self) 两个内建方法，用来表示从索引到样本的映射（Map）.

这样一个数据集dataset，举个例子，当使用dataset[idx]命令时，可以在硬盘中读取数据集中第idx张图片以及其标签（如果有的话）;len(dataset)则会返回这个数据集的容量。

- Iterable式数据集

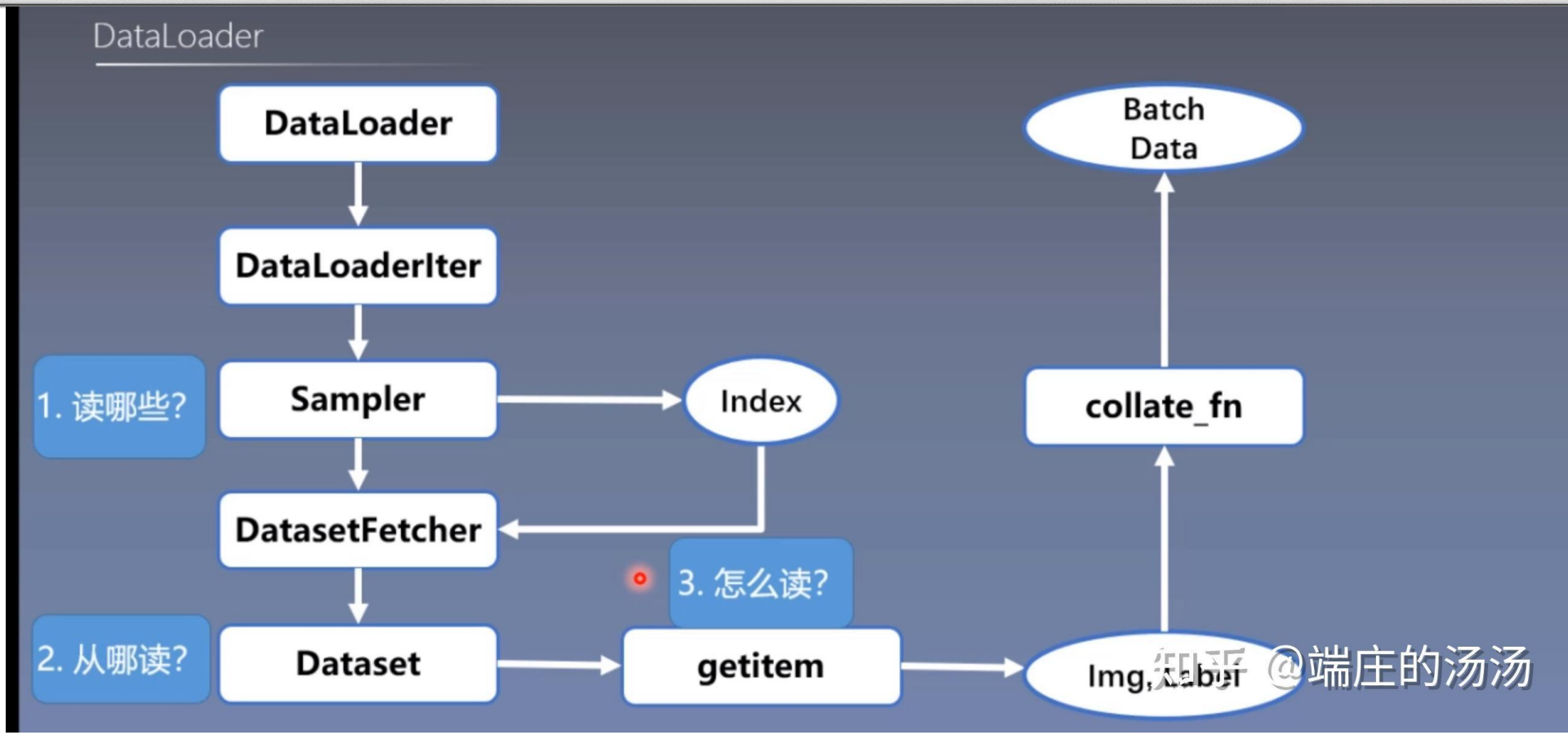
An iterable-style dataset is an instance of a subclass of IterableDataset that implements the __iter__() protocol, and represents an iterable over data samples.
这种数据集主要用于数据大小未知，或者以流的形式的输入，本地文件不固定的情况，需要以迭代的方式来获取样本索引。
And where the batch size depends on the fetched data.

https://pytorch.org/docs/stable/data.html

Pytorch Dataset

```
class CustomDataset(data.Dataset):#需要继承data.Dataset
    def __init__(self):
        # TODO
        # 1. Initialize file path or list of file names.
        pass
    def __getitem__(self, index):
        # TODO
        # 1. Read one data from file (e.g. using numpy.fromfile, PIL.Image.open).
        # 2. Preprocess the data (e.g. torchvision.Transform).
        # 3. Return a data pair (e.g. image and label).
        #这里需要注意的是，第一步：read one data，是一个data
        pass
    def __len__(self):
        # You should change 0 to the total size of your dataset.
        return 0
```

Pytorch DataLoader



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).

https://pytorch.org/docs/stable/data.html

Pytorch DataLoader

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)

Parameters

- ✓ dataset (Dataset) – dataset from which to load the data.
- ✓ batch_size (int, optional) – how many samples per batch to load (default: 1).
- ✓ shuffle (bool, optional) – set to True to have the data reshuffled at every epoch (default: False).
- ✓ sampler (Sampler or Iterable, optional) – defines the strategy to draw samples from the dataset. Can be any Iterable with __len__ implemented. If specified, shuffle must not be specified.
- ✓ batch_sampler (Sampler or Iterable, optional) – like sampler, but returns a batch of indices at a time. Mutually exclusive with batch_size, shuffle, sampler, and drop_last.
- ✓ collate_fn (Callable, optional) – merges a list of samples to form a mini-batch of Tensor(s). Used when using batched loading from a map-style dataset.
- ✓ drop_last (bool, optional) – set to True to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If False and the size of dataset is not divisible by the batch size, then the last batch will be smaller. (default: False)

Pytorch DataLoader

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)

Parameters

- ✓ num_workers (int, optional) – how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process. (default: 0)
- ✓ timeout (numeric, optional) – if positive, the timeout value for collecting a batch from workers. Should always be non-negative. (default: 0)
- ✓ worker_init_fn (Callable, optional) – If not None, this will be called on each worker subprocess with the worker id (an int in [0, num_workers - 1]) as input, after seeding and before data loading. (default: None)
- ✓ prefetch_factor (int, optional, keyword-only arg) – Number of batches loaded in advance by each worker. 2 means there will be a total of 2 * num_workers batches prefetched across all workers. (default value depends on the set value for num_workers. If value of num_workers=0 default is None. Otherwise, if value of num_workers > 0 default is 2).
- ✓ persistent_workers (bool, optional) – If True, the data loader will not shut down the worker processes after a dataset has been consumed once. This allows to maintain the workers Dataset instances alive. (default: False)

Pytorch DataLoader

在 PyTorch 中，DataLoader 使用 num_workers 参数来控制加载数据时使用的子进程数量。通过这种方式，数据加载可以并行进行，从而提高效率，特别是对于需要较多数据预处理的任务（如图像增强或从磁盘读取数据）。

核心机制

主进程分配任务：主进程创建 DataLoader 实例，并通过 num_workers 参数指定需要创建的子进程数量。每个子进程负责加载数据集中的部分数据（通过索引分配）。

子进程加载数据：每个子进程调用 Dataset.__getitem__ 方法来加载和处理数据。如果设置了 collate_fn，子进程会将加载的数据组合成批次。

数据队列与预取：主进程和子进程之间通过队列（multiprocessing.Queue）通信。子进程会将加载的数据放入队列，主进程从队列中读取批次数据。

预取机制（由 prefetch_factor 控制）允许子进程提前加载数据，减少训练时的等待时间。

迭代与同步：主进程在每次迭代时，从队列中获取下一个批次的数据，传递给模型。子进程继续加载后续批次的数据，保持一定的并行度。

Pytorch DataLoader

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)

- ✓ multiprocessing_context (str or multiprocessing.context.BaseContext, optional) – If None, the default multiprocessing context of your operating system will be used. (default: None)
- ✓ generator (torch.Generator, optional) – If not None, this RNG will be used by RandomSampler to generate random indexes and multiprocessing to generate base_seed for workers. (default: None)
- ✓ pin_memory (bool, optional) – If True, the data loader will copy Tensors into device/CUDA pinned memory before returning them. If your data elements are a custom type, or your collate_fn returns a batch that is a custom type, see the example below.
- ✓ pin_memory_device (str, optional) – the device to pin_memory to if pin_memory is True.

Pytorch DataLoader

pin_memory 是 torch.utils.data.DataLoader 的一个可选参数。如果将其设置为 True，数据加载器会在返回数据之前将张量复制到“固定内存”（Pinned Memory）中。

什么是固定内存（Pinned Memory）？固定内存是指被操作系统锁定的内存区域，这些内存不会被交换到磁盘（即，不会进入虚拟内存），因此可以直接进行硬件操作。在 CUDA 编程中，固定内存（或页锁定内存，Page-Locked Memory）允许 GPU 更快速地访问数据，因为它支持直接内存访问（Direct Memory Access, DMA），减少了数据从主机到设备传输的延迟。
当数据加载器需要将数据从 CPU 主机传输到 GPU 时：未使用固定内存： 数据会被存储在普通的分页内存（Paged Memory）中。在这种情况下，数据首先需要通过临时缓冲区传输到固定内存，然后再传输到 GPU。使用固定内存： 数据直接从固定内存传输到 GPU，减少了传输步骤和延迟。通过启用 pin_memory，你可以显著加速数据加载到 GPU 的过程，从而提高整体的训练效率，尤其是在数据加载成为瓶颈时。

何时使用 pin_memory?
推荐场景：使用 GPU 训练模型。数据加载时间较长，可能成为瓶颈。数据从 CPU 内存传输到 GPU 是主要的性能瓶颈。
不推荐场景：仅使用 CPU 进行训练。数据量较小，传输开销可以忽略不计。内存资源有限，固定内存可能导致资源不足。