《机器学习编程实践》课程——DAY 7

课程内容

- 数据集加载的过程（包括自定义 **collate** 函数）

一、 数据集加载

展示了在 PyTorch 中处理数据集的不同方法，重点关注如何自定义 collate 函数

1 构建图像数据集类

```
class my_dataset(Dataset):
  # 初始化
  def __init__(self, path, preprocess):
    self.preprocess = preprocess
    self.image_paths = []
    self.labels = []
    label_list = os.listdir(path) # 获得类别名称

    for label in label_list:
      image_folder = os.path.join(path, label)
      for file_names in os.listdir(image_folder):
        if file_names.endswith(("png", "jpg", "jpeg")):
          self.image_paths.append(os.path.join(image_folder, file_names))
          # 将标签转换为数字
          self.labels.append(label_list.index(label))

  def __len__(self):
    return len(self.image_paths)

  def __getitem__(self, item):
    image = Image.open(self.image_paths[item])
    image = self.preprocess(image) # 图像预处理
    label = self.labels[item]
    return image, label
```

2 定义数据加载器

```
transform = transforms.Compose([
     transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994,
        0.2010)),
])
image_data = my_dataset(r"D:\dataset\cifar100_images\train", transform)
# 进行数据加载
image_loader = torch.utils.data.DataLoader(image_data, batch_size=128,
    shuffle=True, num_workers=0) # 使用默认collate_fn函数

for batch in image_loader:
   x, y = batch
   print(x.shape, y.shape)
   break
```

其中 torch.utils.data.DataLoader 中初始化函数的各参数介绍：

```
def __init__(
   self,
   dataset: Dataset[_T_co], # 必需参数，指要加载的数据集对象
   batch_size: Optional[int] = 1, # 批次大小（若使用batch_sampler可不设置）
   shuffle: Optional[bool] = None, # 是否打乱数据顺序
   # 自定义采样器（如果指定sampler，则shuffle必须为False）
   sampler: Union[Sampler, Iterable, None] = None,
   # 自定义批次采样器（会覆盖batch_size和sampler）
   batch_sampler: Union[Sampler[list], Iterable[list], None] = None,
   num_workers: int = 0, # 数据加载子进程数（0=主进程加载）
   # 批处理函数（数据收集器）：自定义如何采样数据（本次课程重点）
   collate_fn: Optional[_collate_fn_t] = None,
   # 是否将数据复制到CUDA固定内存（加速GPU传输）
   pin_memory: bool = False, # 指定设备（如"cuda"）


   drop_last: bool = False, # 是否丢弃最后不足batch_size的批次
   timeout: float = 0, # 获取数据的超时时间（秒）
   worker_init_fn: Optional[_worker_init_fn_t] = None, # worker初始化函数
   multiprocessing_context=None, 多进程上下文（如"spawn"）
   generator=None, # 随机数生成器（控制shuffle的随机性）
```

```
    *,
    prefetch_factor: Optional[int] = None,
    persistent_workers: bool = False,
    pin_memory_device: str = "",
    in_order: bool = True,
):
    # 记录 DataLoader 类被初始化的次数
    torch._C._log_api_usage_once("python.data_loader")
```

3 collate_fn 函数

3.1 pytorch 默认数据收集器可处理的数据

```
# 仍然是张量类型，但会加一个维度：batch_size
 :class:'torch.Tensor' -> :class:'torch.Tensor'

# 下面各类型都会先转换成tensor
NumPy Arrays -> :class:'torch.Tensor'
'float' -> :class:'torch.Tensor'
'int`' -> :class:'torch.Tensor'
"""
Example with a batch of 'int':
>>> default_collate([0, 1, 2, 3])
tensor([0, 1, 2, 3])
"""

# 下面各类型不会转换
'str' -> 'str' (unchanged)
'bytes' -> 'bytes' (unchanged)
# mapping：相当于字典，执行时键值K不动、对V递归执行collate_fn
'Mapping[K, V_i]' -> 'Mapping[K, default_collate([V_1, V_2, ...])]'
"""
example with mapping:
data1={"input":torch.Tensor([1,2]),"output":3}
data2={"input":torch.Tensor([1,3]),"output":2}
data3={"input":torch.Tensor([1,4]),"output":1}
batch={"input":collate_fn([torch.Tensor([1,2]),torch.Tensor([1,3]),torch.Tensor([1,4])],
   "output":collate_fn([3,2,1])}
"""
```

```
'NamedTuple[V1_i, V2_i, ...]' -> 'NamedTuple[default_collate([V1_1, V1_2,
    ...]),default_collate([V2_1, V2_2, ...]), ...]'
"""
Example with 'NamedTuple' inside the batch:
>>> Point = namedtuple('Point', ['x', 'y'])
>>> default_collate([Point(0, 0), Point(1, 1)])
Point(x=tensor([0, 1]), y=tensor([0, 1]))
"""

'Tuple'
"""
# Example with 'Tuple' inside the batch:
>>> default_collate([(0, 1), (2, 3)])
[tensor([0, 2]), tensor([1, 3])]
"""

'List'
"""
# Example with 'List' inside the batch:
>>> default_collate([[0, 1], [2, 3]])
[tensor([0, 2]), tensor([1, 3])]
"""

'Sequence[V1_i, V2_i, ...]' -> 'Sequence[default_collate([V1_1, V1_2,
    ...]),default_collate([V2_1, V2_2, ...]), ...]'
```

观察发现，要求 list 的元素**个数相同**，个数不同就会报错
因此最好自定义 collate_fn 函数

3.2 自定义 collate_fn 函数

```
# 以LLM文本生成数据集为例，每个batch包含多组prompt-label
# prompts_batch = [item["prompts"] for item in batch]
# labels_batch = [item["labels"] for item in batch]
def custom_collate_fn(batch):
   prompts_batch = []
   labels_batch = []
   for item in batch:
      prompts_batch += item["prompts"]
```

```
    for item in batch:
        labels_batch += item["labels"]
    return {"prompts": prompts_batch, "labels": labels_batch}
```

使用示例：

```
my_loader = torch.utils.data.DataLoader(my_data, batch_size=4,
    shuffle=False, num_workers=0, collate_fn=custom_collate_fn)
```

3.3 set_format 的作用

确保列表类型数据可以正确加载

```
# 添加句首/句尾标记
text_data = text_data.map(add_eos_to_examples, batched=False,
    remove_columns=text_data.column_names)
print(text_data[0])

# 特征转换（tokenization）
text_data = text_data.map(convert_to_features, batched=True,
    remove_columns=text_data.column_names)
print(text_data[0])

# set_format自动将数据转换为 PyTorch Tensor
text_data.set_format(type="torch")
print(text_data[0])
text_loader = torch.utils.data.DataLoader(text_data, batch_size=4,
    shuffle=False, num_workers=0)
try:
    for batch in text_loader:
        print(batch)
        break
except Exception as e:
    print(e)
```

set_format 的作用：

假设 batch_size 为 4，是 list 数据, 分别为：[0,0,0,0,0]，[1,1,1,1,1]，[2,2,2,2,2],[3,3,3,3,3]

- **使用 set_format：**

  自动将数据转换为 PyTorch Tensor：

  tensor([0,0,0,0,0]),tensor([1,1,1,1,1]),tensor([2,2,2,2,2]),tensor([3,3,3,3,3])

输入 DataLoader 作用于默认 collate_fn 函数的结果：

tensor([0,0,0,0,0],[1,1,1,1,1],[2,2,2,2,2],[3,3,3,3,3])

- **不使用 set_format：**

  保持 python 原生格式，即 list 类型：

  [0,0,0,0,0]，[1,1,1,1,1]，[2,2,2,2,2],[3,3,3,3,3]

  输入 DataLoader 作用于默认 collate_fn 函数的结果：

  [tensor([0,1,2,3]),tensor([0,1,2,3]),tensor([0,1,2,3]),tensor([0,1,2,3]),tensor([0,1,2,3])]

  此时不具有实际意义（因为将一个列表元素拆分开来），期望是以整个列表出现

解决方案是自定义 collate_fn