Handling Missing Data in PyTorch DataLoader

Introduction

This document explains how to handle missing data in PyTorch's DataLoader by using a custom collate function. We will use a custom dataset with variable-length sequences and some missing data as an example.

Code Example

```
import torch
from torch.utils.data import DataLoader, Dataset
from torch.nn.utils.rnn import pad_sequence

# Sample dataset with missing data
data = [torch.tensor([1, 2]), None, torch.tensor([3, 4])]
labels = torch.tensor([0, 1, 0])

class MissingDataDataset(Dataset):
    def __getitem__(self, index):
        return data[index], labels[index]

    def __len__(self):
        return len(data)

dataset = MissingDataDataset()

def handle_missing_collate(batch):
    data, labels = zip(*batch)
    data = [d if d is not None else torch.zeros(2) for d in
       data]
    data = torch.stack(data)
    labels = torch.tensor(labels)
    return data, labels

dataloader = DataLoader(dataset, batch_size=2, shuffle=True,
   collate_fn=handle_missing_collate)

for batch in dataloader:
    print(batch)
```

Explanation

1. Dataset Definition with Missing Data

The dataset contains some missing data entries:

- `data = [tensor([1, 2]), None, tensor([3, 4])]`
- `labels = tensor([0, 1, 0])`

2. Custom Dataset Class

The `MissingDataDataset` class inherits from `torch.utils.data.Dataset` and implements the `__getitem__` and `__len__` methods.

```
class MissingDataDataset(Dataset):
    def __getitem__(self, index):
        return data[index], labels[index]

    def __len__(self):
        return len(data)
```

3. Custom Collate Function to Handle Missing Data

The `handle_missing_collate` function processes a batch of data, replacing `None` entries with zero tensors.

```
def handle_missing_collate(batch):
    data, labels = zip(*batch)
    data = [d if d is not None else torch.zeros(2) for d in
       data]
    data = torch.stack(data)
    labels = torch.tensor(labels)
    return data, labels
```

4. DataLoader Initialization

The DataLoader is initialized with the custom dataset, batch size of 2, shuffling enabled, and using the custom collate function.

```
dataloader = DataLoader(dataset, batch_size=2, shuffle=True,
   collate_fn=handle_missing_collate)
```

5. Iterating Through DataLoader

This loop retrieves batches from the DataLoader and prints them.

```
for batch in dataloader:
    print(batch)
```

Internal Process Overview with Example

1. Dataset and DataLoader

The dataset contains:

- `data = [tensor([1, 2]), None, tensor([3, 4])]`
- `labels = tensor([0, 1, 0])`

2. Batch Formation

Suppose the DataLoader generates indices `[2, 0]` (since `shuffle=True`).

3. Calling __getitem__

The DataLoader calls:

```
dataset[2]  # returns (tensor([3, 4]), 0)
dataset[0]  # returns (tensor([1, 2]), 0)
```

Resulting batch:

```
batch = [(tensor([3, 4]), 0), (tensor([1, 2]), 0)]
```

4. Applying handle_missing_collate Function

The function processes the batch:

```
data, labels = zip(*batch)
# data = (tensor([3, 4]), tensor([1, 2]))
# labels = (0, 0)
data = [d if d is not None else torch.zeros(2) for d in data]
# data = [tensor([3, 4]), tensor([1, 2])]
data = torch.stack(data)
# data = tensor([[3, 4],
#                [1, 2]])
labels = torch.tensor(labels)
# labels = tensor([0, 0])
```

Final batch:

```
(tensor([[3, 4],
         [1, 2]]), tensor([0, 0]))
```

5. Example with Missing Data

Suppose the DataLoader generates indices `[1, 0]`. Calling __getitem__:

```
dataset[1]  # returns (None, 1)
dataset[0]  # returns (tensor([1, 2]), 0)
```

Resulting batch:

```
batch = [(None, 1), (tensor([1, 2]), 0)]
```

Applying handle_missing_collate:

```
data, labels = zip(*batch)
# data = (None, tensor([1, 2]))
# labels = (1, 0)
data = [d if d is not None else torch.zeros(2) for d in data]
# data = [tensor([0, 0]), tensor([1, 2])]
data = torch.stack(data)
# data = tensor([[0, 0],
#                [1, 2]])
labels = torch.tensor(labels)
# labels = tensor([1, 0])
```

Final batch:

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

Explanation of Zero Padding

The number of zeros padded depends entirely on the batch. For example, in the output:

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

In this batch: - The missing data (None) is replaced with a tensor of zeros of appropriate shape. - This ensures that all sequences in a batch have the same length, with the padding varying based on the sequences within that specific batch.