Handling Missing Data in PyTorch's DataLoader and Why Do We Need torch.stack(data)

Introduction

In PyTorch, the `DataLoader` class is used to load data in batches. When dealing with datasets that contain missing data, it is important to handle these cases properly to avoid errors during batch processing. This note provides two code examples to demonstrate how to handle missing data and the impact on the output.

Code Example without Stacking Tensors

```
import torch
from torch.utils.data import DataLoader

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

class MissingData(torch.utils.data.Dataset):
   def __getitem__(self, id):
      return data[id], labels[id]

   def __len__(self):
      return len(data)

def collate_fun(batch):
   data, label = zip(*batch)
   label = torch.tensor(label)
```

```
   data = [d if d is not None else torch.zeros(2) for d in data]
   return data, label

dataset = MissingData()
dataloader = DataLoader(dataset, batch_size=2, shuffle=True,
   red↪ collate_fn=collate_fun)

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

Output:

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

In this example, the missing data is replaced with tensors of zeros, but the data tensors are not stacked. The output remains as a list of tensors.

Code Example with Stacking Tensors

```
import torch
from torch.utils.data import DataLoader

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

class MissingData(torch.utils.data.Dataset):
   def __getitem__(self, id):
      return data[id], labels[id]

   def __len__(self):
      return len(data)

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

dataset = MissingData()
dataloader = DataLoader(dataset, batch_size=2, shuffle=True,
   red↪ collate_fn=collate_fun)
```

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

Output:

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

In this example, the missing data is replaced with tensors of zeros and the data tensors are stacked using `torch.stack`. The output is a single tensor for the data in each batch.

Conclusion

The difference between the two code examples lies in how the data tensors are handled. In the first example, the data tensors are not stacked, resulting in a list of tensors. In the second example, the data tensors are stacked using `torch.stack`, resulting in a single tensor for each batch. Understanding this behavior is crucial for correctly handling missing data in neural network models.