Creating Masks for Padded Sequences in PyTorch

Introduction

In this document, we aim to demonstrate how to handle variable-length sequences in PyTorch by padding them to the same length and creating corresponding masks. This is particularly useful for tasks such as sequence modeling where inputs of different lengths need to be processed in batches. We will walk through the process with a step-by-step example.

Code Example

```
import torch
from torch.nn.utils.rnn import pad_sequence

# Sample dataset with variable-length sequences
data = [torch.tensor([1, 2]), torch.tensor([3, 4, 5]), torch.
    red↪ tensor([6])]
labels = torch.tensor([0, 1, 0])

# Pad sequences to the same length
padded_data = pad_sequence(data, batch_first=True, padding_value
    red↪ =0)
# padded_data:
# tensor([[1, 2, 0],
# [3, 4, 5],
# [6, 0, 0]])

# Lengths of the original sequences
lengths = [2, 3, 1]
```

```
# Create masks
masks = torch.zeros_like(padded_data).float()
for i, length in enumerate(lengths):
   masks[i, :length] = 1

print("Padded␣Data:")
print(padded_data)
print("Masks:")
print(masks)
```

Output

```
Padded Data:
tensor([[1, 2, 0],
      [3, 4, 5],
      [6, 0, 0]])
Masks:
tensor([[1., 1., 0.],
      [1., 1., 1.],
      [1., 0., 0.]])
```

This code will output the padded sequences and their corresponding masks, where valid elements are marked with 1 and padding elements are marked with 0.

Explanation

Let's use the following data and lengths to illustrate:

```
data = pad_sequence([torch.tensor([1, 2]), torch.tensor([3, 4,
   red↪ 5]), torch.tensor([6])], batch_first=True, padding_value
   red↪ =0)
# Padded data tensor:
# tensor([[1, 2, 0],
# [3, 4, 5],
# [6, 0, 0]])

lengths = [2, 3, 1]
```

Now, let's go through the mask creation step by step.

Initial Mask Tensor

```
masks = torch.zeros_like(data).float()
# masks will be:
# tensor([[0., 0., 0.],
# [0., 0., 0.],
# [0., 0., 0.]])
```

Setting Mask Values

```
for i, length in enumerate(lengths):
   masks[i, :length] = 1
```

Iteration 1 (i=0, length=2)

```
masks[0, :2] = 1
# masks will be:
# tensor([[1., 1., 0.],
# [0., 0., 0.],
# [0., 0., 0.]])
```

Iteration 2 (i=1, length=3)

```
masks[1, :3] = 1
# masks will be:
# tensor([[1., 1., 0.],
# [1., 1., 1.],
# [0., 0., 0.]])
```

Iteration 3 (i=2, length=1)

```
masks[2, :1] = 1
# masks will be:
# tensor([[1., 1., 0.],
# [1., 1., 1.],
# [1., 0., 0.]])
```

Final Masks

The final mask tensor indicates which elements in the padded data are valid (1) and which are padding (0):

- The first sequence is [1, 2] with a length of 2, so its mask is [1, 1, 0]. - The second sequence is [3, 4, 5] with a length of 3, so its mask is [1, 1, 1]. - The third sequence is [6] with a length of 1, so its mask is [1, 0, 0].

Full Example

```
import torch
from torch.nn.utils.rnn import pad_sequence

data = [torch.tensor([1, 2]), torch.tensor([3, 4, 5]), torch.
    red↪ tensor([6])]
labels = torch.tensor([0, 1, 0])

padded_data = pad_sequence(data, batch_first=True, padding_value
    red↪ =0)
# padded_data:
# tensor([[1, 2, 0],
# [3, 4, 5],
# [6, 0, 0]])

lengths = [2, 3, 1]

masks = torch.zeros_like(padded_data).float()
for i, length in enumerate(lengths):
   masks[i, :length] = 1

print("Padded␣Data:")
print(padded_data)
print("Masks:")
print(masks)
```

This code will output:

```
Padded Data:
tensor([[1, 2, 0],
      [3, 4, 5],
      [6, 0, 0]])
Masks:
tensor([[1., 1., 0.],
        [1., 1., 1.],
        [1., 0., 0.]])
```

This shows the padded sequences and their corresponding masks, where valid elements are marked with 1 and padding elements are marked with 0.