**压紧填充序列**

在使用PyTorch处理数据时，一般是采用batch的形式同时处理多个样本序列，而每个batch中的样本序列是不等长的，导致RNN无法处理。所以，通常的做法是先将每个batch按照最长的序列进行padding处理等长的形式。

但padding操作会带来一个问题，那就是对于多数进行padding过的序列，会导致多了很多无用的字符。希望的是在最后一个有用的字符后就可以输出该序列的向量表示，而不是在很多padding字符后。

此时，可通过pack\_padded\_sequence将一个经过padding后的变长序列压紧，压紧后就不含padding的字符0了。

**torch.nn.utils.rnn.pack\_padded\_sequence(input,length,batch\_first,** **enforce\_sorted=True)**

**参数解释**

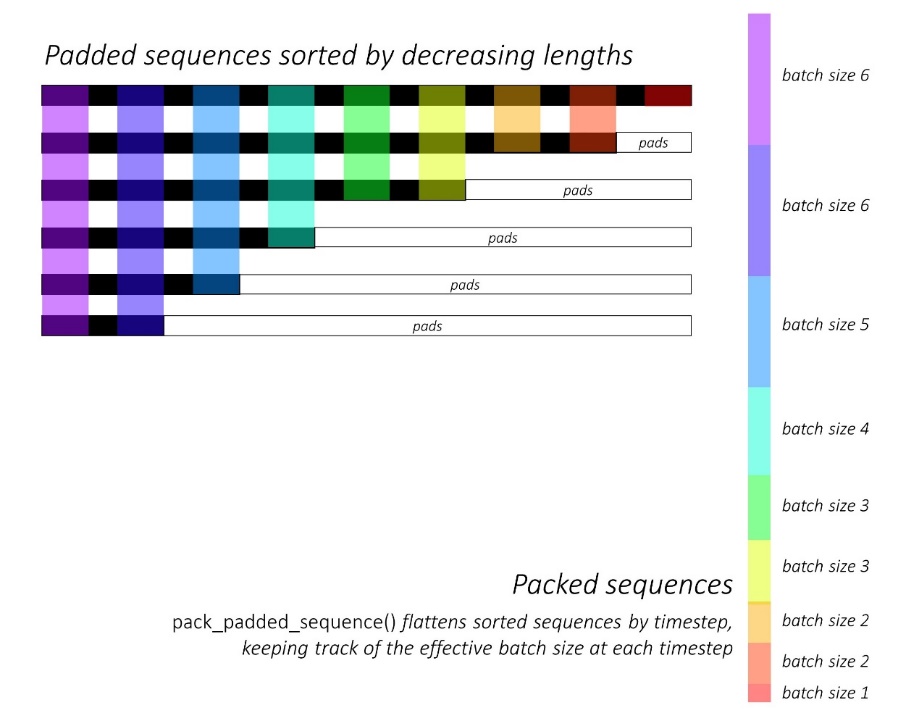
若batch\_first=True，则

input：batch\_size×length\_seq×input\_size的输入

length：与input相对应的batch\_size长度的列表，其中每个元素是批量中每个样本的实际序列长度

enforce\_sorted：是否按length将input强制降序排序，若强制排序，则input[0]应该最长，input[batch\_size-1]应该最短

**函数功能**



按照可变长度length压紧序列，返回一个PackSequence对象，按时间步拉平已按照实际长度排序的多个序列，并记录每个时间步的批大小

input = torch.tensor([[1,2,3,4,5],

[1,2,3,4,0],

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

[1,2,0,0,0]]).unsqueeze(2)

lengths = [5,4,3,2]

input\_packed = nn.utils.rnn.pack\_padded\_sequence(input, lengths, batch\_first=True, enforce\_sorted=False)

print(type(input\_packed))

print(input\_packed)

>>>

<class 'torch.nn.utils.rnn.PackedSequence'>

PackedSequence(data=tensor([[1],

[1],

[1],

[1],

[2],

[2],

[2],

[2],

[3],

[3],

[3],

[4],

[4],

[5]]), batch\_sizes=tensor([4, 4, 3, 2, 1]), sorted\_indices=tensor([0, 1, 2, 3]), unsorted\_indices=tensor([0, 1, 2, 3]))

用pack\_padded\_sequence处理输入，再输入RNN的暂时不写

*"""求损失"""*

pred = torch.randn(4,6,5)*#模拟批量中每个样本在各时间步上词典中各个词的概率*

lengths = torch.tensor([6,5,3,2])*#规定每个序列的长度，按照该长度截取序列中的靠前部分，即使序列中0改为其他数，结果仍然不变*

target = torch.tensor([[1,2,3,4,2,4],

[1,2,3,4,3,0],

[1,2,4,0,0,0],

[1,2,0,0,0,0]])

criterion = nn.CrossEntropyLoss()

pred\_packed = nn.utils.rnn.pack\_padded\_sequence(pred, lengths, batch\_first=True).data

target\_packed = nn.utils.rnn.pack\_padded\_sequence(target, lengths, batch\_first=True).data

loss = criterion(pred\_packed,target\_packed)

print(type(pred\_packed))

print(pred\_packed)

print(target\_packed)

print(loss)

>>>

tensor([[-1.8481e+00, 3.6931e-01, 1.5493e-01, 9.0750e-01, 8.3813e-01],

[ 3.2476e-01, -5.6642e-01, 8.4930e-01, 6.3233e-01, 2.6574e-01],

[-1.3479e+00, -3.8757e-01, -1.4394e+00, 1.1456e+00, -1.0220e+00],

[ 6.0105e-01, 1.4729e-04, 7.8888e-01, -1.7261e+00, -1.2590e+00],

[ 2.9313e-01, -1.2414e-01, 1.4014e+00, 1.3431e+00, 8.3515e-02],

[ 2.4648e+00, -8.6901e-01, -1.1452e-01, -7.1225e-01, -1.6640e-01],

[ 6.9017e-01, 3.2076e-01, 3.1110e-01, -5.5680e-02, 5.0748e-01],

[-1.0360e+00, -8.5917e-02, -1.4969e+00, -1.7011e+00, -9.3809e-01],

[ 4.6478e-01, 1.1325e+00, -9.5100e-02, 2.4038e-01, 2.7457e-01],

[ 7.9044e-01, 6.8425e-01, -6.9772e-01, 1.4286e+00, 1.7439e+00],

[-3.8869e-01, -9.2449e-01, -2.2165e-01, 8.8458e-01, -1.6050e+00],

[-1.3097e-01, 5.3221e-01, 4.4650e-01, 7.5885e-01, 2.9712e-01],

[-1.4244e+00, 1.2308e+00, 1.7454e+00, 4.2942e-01, 5.6410e-01],

[-5.5668e-01, 9.7513e-02, 6.4632e-01, -8.7660e-01, 2.9258e-01],

[-1.6388e-01, -1.3093e+00, -2.4074e-01, -3.3282e-01, 1.2814e+00],

[ 6.6134e-01, -1.1141e-01, 1.1809e+00, 7.8697e-01, -4.2150e-01]])

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

tensor(1.9566)

​