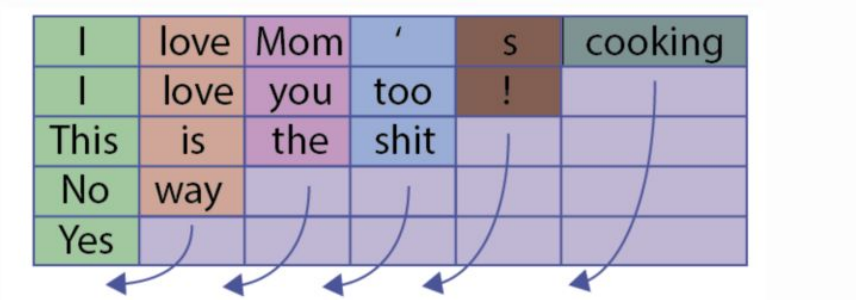
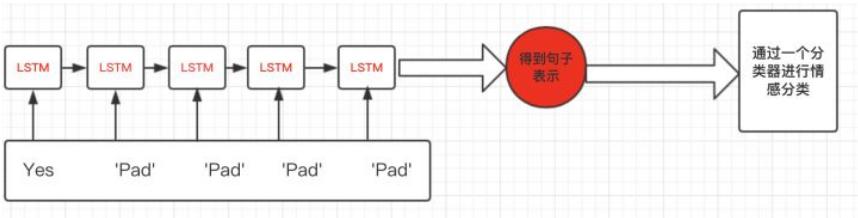
Pytorch中的RNN之pack_padded_sequence()和pad_packed_sequence()

为什么有pad和pack操作?

先看一个例子，这个batch中有5个sample

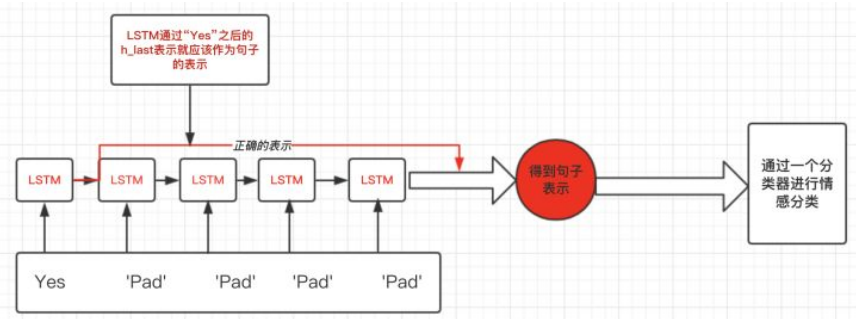


如果不用pack和pad操作会有一个问题，什么问题呢？比如上图，句子“Yes”只有一个单词，但是padding了多余的pad符号，这样会导致LSTM对它的表示通过了非常多无用的字符，这样得到的句子表示就会有误差，更直观的如下图：



那么我们正确的做法应该是怎么样呢?

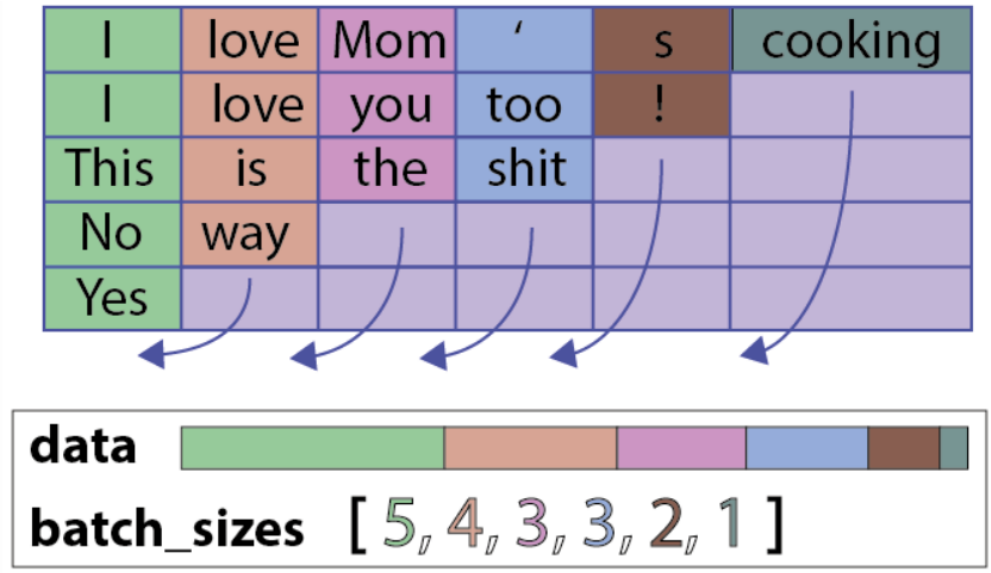
在上面这个例子，我们想要得到的表示仅仅是LSTM过完单词"Yes"之后的表示，而不是通过了多个无用的“Pad”得到的表示：如下图：



torch.nn.utils.rnn.pack_padded_sequence()

这里的pack，理解成压紧比较好。 将一个 填充过的变长序列 压紧。（填充时候，会有冗余，所以压紧一下）

其中pack的过程为：（注意pack的形式，不是按行压，而是按列压）



（下面方框内为PackedSequence对象，由data和batch_sizes组成）

pack之后，原来填充的 PAD（一般初始化为0）占位符被删掉了。

输入的形状可以是(T×B×*)。T是最长序列长度，B是batch size，*代表任意维度(可以是0)。如果batch_first=True的话，那么相应的 input size 就是 (B×T×*)。

Variable中保存的序列，应该按序列长度的长短排序，长的在前，短的在后。即input[:,0]代表的是最长的序列，input[:, B-1]保存的是最短的序列。

NOTE： 只要是维度大于等于2的input都可以作为这个函数的参数。你可以用它来打包labels，然后用RNN的输出和打包后的labels来计算loss。通过PackedSequence对象的.data属性可以获取 Variable。

参数说明:

- input (Variable) – 变长序列 被填充后的 batch
- lengths (list[int]) – Variable 中 每个序列的长度。
- batch_first (bool, optional) – 如果是True，input的形状应该是B*T*size。

返回值:

一个PackedSequence 对象。

torch.nn.utils.rnn.pad_packed_sequence()

填充packed_sequence。

上面提到的函数的功能是将一个填充后的变长序列压紧。 这个操作和

pack_padded_sequence()是相反的。把压紧的序列再填充回来。填充时会初始化为0。

返回的Varaible的值的size是 T×B×*, T 是最长序列的长度，B 是 batch_size,如果 batch_first=True,那么返回值是B×T×*。

Batch中的元素将会以它们长度的逆序排列。

参数说明:

- sequence (PackedSequence) – 将要被填充的 batch
- batch_first (bool, optional) – 如果为True，返回的数据的格式为 B×T×*。

返回值: 一个tuple，包含被填充后的序列，和batch中序列的长度列表

一个例子:

```
import torch
import torch.nn as nn
from torch.autograd import Variable
from torch.nn import utils as nn_utils
batch_size = 2
max_length = 3
hidden_size = 2
n_layers =1

tensor_in = torch.FloatTensor([[1, 2, 3], [1, 0, 0]]).resize_(2,3,1)
tensor_in = Variable( tensor_in ) #[batch, seq, feature], [2, 3, 1]
seq_lengths = [3,1] # list of integers holding information about the batch size at each sequence step

# pack it
pack = nn_utils.rnn.pack_padded_sequence(tensor_in, seq_lengths, batch_first=True)
print('packed:',pack)

# initialize
rnn = nn.RNN(1, hidden_size, n_layers, batch_first=True)
h0 = Variable(torch.randn(n_layers, batch_size, hidden_size))

#forward
out, _ = rnn(pack, h0)
print('out:',out)

# unpack
unpacked = nn_utils.rnn.pad_packed_sequence(out)
print('unpacked',unpacked)
```

输出： （这个输出结果能较为清楚地看到中间过程）

```
packed: PackedSequence(data=tensor([[1.],
        [1.],
        [2.],
        [3.]]), batch_sizes=tensor([2, 1, 1]))
out: PackedSequence(data=tensor([[-0.7841, -0.7122],
        [-0.7443, -0.8267],
        [-0.8814, -0.9341],
        [-0.9625, -0.9721]], grad_fn=<CatBackward>), batch_sizes=tensor([2, 1, 1]))
unpacked (tensor([[[-0.7841, -0.7122],
         [-0.7443, -0.8267]],

        [[-0.8814, -0.9341],
         [ 0.0000,  0.0000]],

        [[-0.9625, -0.9721],
         [ 0.0000,  0.0000]]], grad_fn=<CopySlices>), tensor([3, 1]))
```

此时PackedSequence对象输入RNN后，输出RNN的还是PackedSequence对象

（最后一个unpacked没有用batch_first，所以。。。）

参考：

https://www.cnblogs.com/lindaxin/p/8052043.html

https://pytorch.org/docs/stable/nn.html?highlight=pack_padded_sequence#torch.nn.utils.rnn.pack_padded_sequence

https://zhuanlan.zhihu.com/p/34418001?edition=yidianzixun&utm_source=yidianzixun&yidian_docid=0IVwLf60
