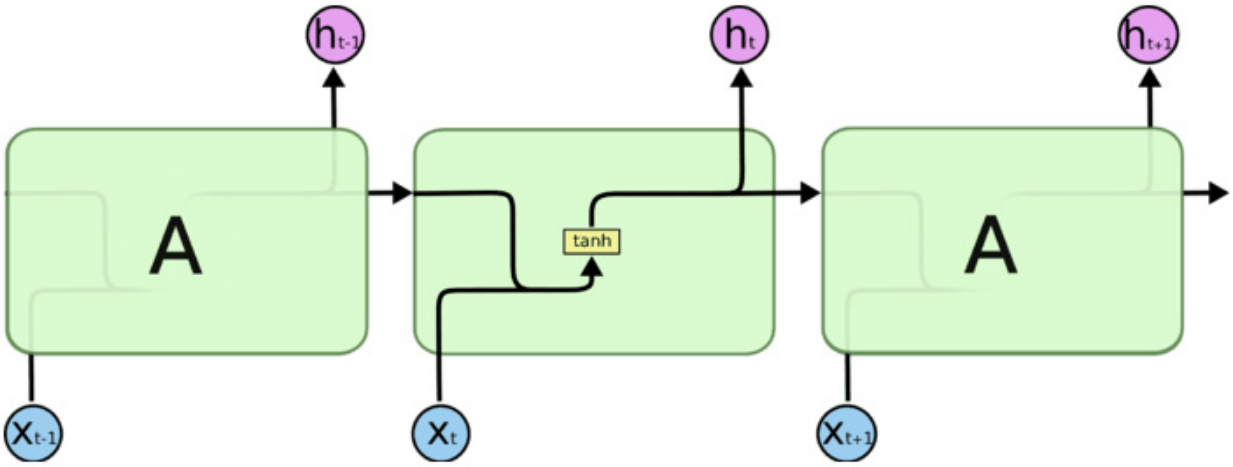
PyTorch 中的循环神经网络模块

前面我们讲了循环神经网络的基础知识和网络结构，下面我们教大家如何在 pytorch 下构建循环神经网络，因为 pytorch 的动态图机制，使得循环神经网络非常方便。

一般的 RNN



对于最简单的 RNN，我们可以使用下面两种方式去调用，分别是 `torch.nn.RNNCell()` 和 `torch.nn.RNN()`，这两种方式的区别在于 `RNNCell()` 只能接受序列中单步的输入，且必须传入隐藏状态，而 `RNN()` 可以接受一个序列的输入，默认会传入全 0 的隐藏状态，也可以自己申明隐藏状态传入。

`RNN()` 里面的参数有

input_size 表示输入 x_t 的特征维度

hidden_size 表示输出的特征维度

num_layers 表示网络的层数

nonlinearity 表示选用的非线性激活函数，默认是 'tanh'

bias 表示是否使用偏置，默认使用

batch_first 表示输入数据的形式，默认是 False，就是这样形式，(seq, batch, feature)，也就是将序列长度放在第一位，batch 放在第二位

dropout 表示是否在输出层应用 dropout

bidirectional 表示是否使用双向的 rnn，默认是 False

对于 `RNNCell()`，里面的参数就少很多，只有 input_size，hidden_size，bias 以及 nonlinearity

```
import torch
from torch.autograd import Variable
from torch import nn
```

```
# 定义一个单步的 rnn
rnn_single = nn.RNNCell(input_size=100, hidden_size=200)
```

```
# 访问其中的参数
rnn_single.weight_hh
```

```
Parameter containing:
1.00000e-02 *
 6.2260 -5.3805  3.5870  ...  -2.2162  6.2760  1.6760
-5.1878 -4.6751 -5.5926  ...  -1.8942  0.1589  1.0725
 3.3236 -3.2726  5.5399  ...   3.3193  0.2117  1.1730
          ...             ⋱              ...
 2.4032 -3.4415  5.1036  ...  -2.2035 -0.1900 -6.4016
 5.2031 -1.5793 -0.0623  ...   0.3424  6.9412  6.3707
-5.4495  4.5280  2.1774  ...   1.8767  2.4968  5.3403
[torch.FloatTensor of size 200x200]
```

```
# 构造一个序列，长为 6，batch 是 5，  特征是 100
x = Variable(torch.randn(6, 5, 100)) # 这是 rnn 的输入格式
```

```
# 定义初始的记忆状态
h_t = Variable(torch.zeros(5, 200))
```

```
# 传入 rnn
out = []
for i in range(6): # 通过循环 6 次作用在整个序列上
    h_t = rnn_single(x[i], h_t)
    out.append(h_t)
```

```
h_t
```

```
Variable containing:
 0.0136  0.3723  0.1704  ...   0.4306 -0.7909 -0.5306
-0.2681 -0.6261 -0.3926  ...   0.1752  0.5739 -0.2061
-0.4918 -0.7611  0.2787  ...   0.0854 -0.3899  0.0092
 0.6050  0.1852 -0.4261  ...  -0.7220  0.6809  0.1825
-0.6851  0.7273  0.5396  ...  -0.7969  0.6133 -0.0852
[torch.FloatTensor of size 5x200]
```

```
len(out)
```

```
6
```

```
out[0].shape # 每个输出的维度
```

```
torch.Size([5, 200])
```

可以看到经过了 rnn 之后，隐藏状态的值已经被改变了，因为网络记忆了序列中的信息，同时输出 6 个结果

下面我们看看直接使用 `RNN` 的情况

```
rnn_seq = nn.RNN(100, 200)
```

```
# 访问其中的参数
rnn_seq.weight_hh_l0
```

```
Parameter containing:
1.00000e-02 *
 1.0998 -1.5018 -1.4337  ...   3.8385 -0.8958 -1.6781
 5.3302 -5.4654  5.5568  ...   4.7399  5.4110  3.6170
 1.0788 -0.6620  5.7689  ...  -5.0747 -2.9066  0.6152
          ...             ⋱             ...
-5.6921  0.1843 -0.0803  ...  -4.5852  5.6194 -1.4734
 4.4306  6.9795 -1.5736  ...   3.4236 -0.3441  3.1397
 7.0349 -1.6120 -4.2840  ...  -5.5676  6.8897  6.1968
[torch.FloatTensor of size 200x200]
```

```
out, h_t = rnn_seq(x) # 使用默认的全 0 隐藏状态
```

```
h_t
```

```
Variable containing:
( 0 ,.,.) =
  0.2012  0.0517  0.0570  ...   0.2316  0.3615 -0.1247
  0.5307  0.4147  0.7881  ...  -0.4138 -0.1444  0.3602
  0.0882  0.4307  0.3939  ...   0.3244 -0.4629 -0.2315
  0.2868  0.7400  0.6534  ...   0.6631  0.2624 -0.0162
  0.0841  0.6274  0.1840  ...   0.5800  0.8780  0.4301
[torch.FloatTensor of size 1x5x200]
```

```
len(out)
```

```
6
```

这里的 h_t 是网络最后的隐藏状态，网络也输出了 6 个结果

```
# 自己定义初始的隐藏状态
h_0 = Variable(torch.randn(1, 5, 200))
```

这里的隐藏状态的大小有三个维度，分别是 (num_layers * num_direction, batch, hidden_size)

```
out, h_t = rnn_seq(x, h_0)
```

```
h_t
```

```
Variable containing:
( 0 ,.,.) =
  0.2091  0.0353  0.0625  ...   0.2340  0.3734 -0.1307
  0.5498  0.4221  0.7877  ...  -0.4143 -0.1209  0.3335
  0.0757  0.4204  0.3826  ...   0.3187 -0.4626 -0.2336
  0.3106  0.7355  0.6436  ...   0.6611  0.2587 -0.0338
  0.1025  0.6350  0.1943  ...   0.5720  0.8749  0.4525
[torch.FloatTensor of size 1x5x200]
```

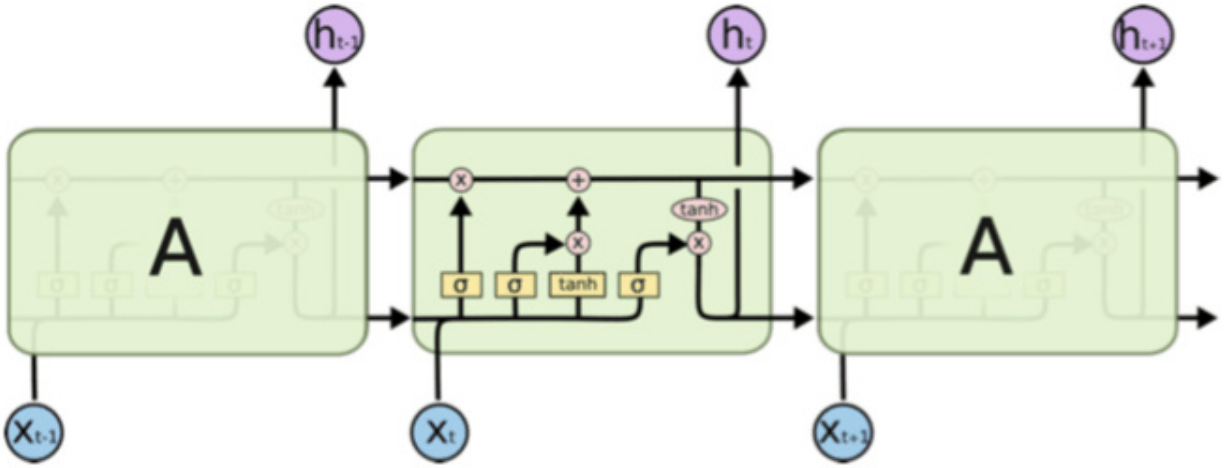
```
out.shape
```

```
torch.Size([6, 5, 200])
```

同时输出的结果也是 (seq, batch, feature)

一般情况下我们都是用 `nn.RNN()` 而不是 `nn.RNNCell()`，因为 `nn.RNN()` 能够避免我们手动写循环，非常方便，同时如果不特别说明，我们也会选择使用默认的全 0 初始化隐藏状态

LSTM



LSTM 和基本的 RNN 是一样的，他的参数也是相同的，同时他也有 `nn.LSTMCell()` 和 `nn.LSTM()` 两种形式，跟前面讲的都是相同的，我们就不再赘述了，下面直接举个小例子

```
lstm_seq = nn.LSTM(50, 100, num_layers=2) # 输入维度 100，输出 200，两层
```

```
lstm_seq.weight_hh_l0 # 第一层的 h_t 权重
```

```
Parameter containing:
1.00000e-02 *
 3.8420  5.7387  6.1351  ...   1.2680  0.9890  1.3037
-4.2301  6.8294 -4.8627  ...  -6.4147  4.3015  8.4103
 9.4411  5.0195  9.8620  ...  -1.6096  9.2516 -0.6941
          ...             ⋱             ...
 1.2930 -1.3300 -0.9311  ...  -6.0891 -0.7164  3.9578
 9.0435  2.4674  9.4107  ...  -3.3822 -3.9773 -3.0685
-4.2039 -8.2992 -3.3605  ...   2.2875  8.2163 -9.3277
[torch.FloatTensor of size 400x100]
```

小练习：想想为什么这个系数的大小是 (400, 100)

```
lstm_input = Variable(torch.randn(10, 3, 50)) # 序列 10，batch 是 3，输入维度 50
```

```
out, (h, c) = lstm_seq(lstm_input) # 使用默认的全 0 隐藏状态
```

注意这里 LSTM 输出的隐藏状态有两个，h 和 c，就是上图中的每个 cell 之间的两个箭头，这两个隐藏状态的大小都是相同的，(num_layers * direction, batch, feature)

```
h.shape # 两层，Batch 是 3，特征是 100
```

```
torch.Size([2, 3, 100])
```

```
c.shape
```

```
torch.Size([2, 3, 100])
```

```
out.shape
```

```
torch.Size([10, 3, 100])
```

我们可以不使用默认的隐藏状态，这是需要传入两个张量

```
h_init = Variable(torch.randn(2, 3, 100))
c_init = Variable(torch.randn(2, 3, 100))
```

```
out, (h, c) = lstm_seq(lstm_input, (h_init, c_init))
```

```
h.shape
```

```
torch.Size([2, 3, 100])
```

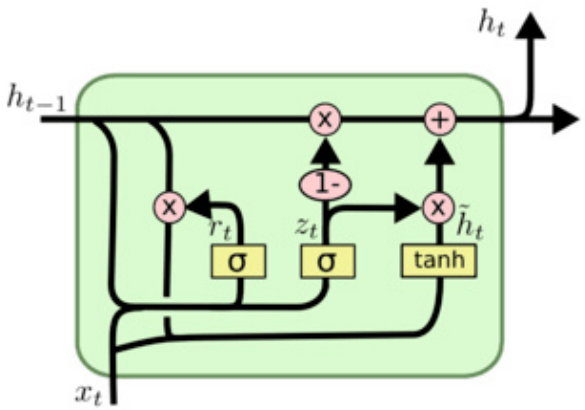
```
c.shape
```

```
torch.Size([2, 3, 100])
```

```
out.shape
```

```
torch.Size([10, 3, 100])
```

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU 和前面讲的这两个是同样的道理，就不再细说，还是演示一下例子

```
gru_seq = nn.GRU(10, 20)
gru_input = Variable(torch.randn(3, 32, 10))

out, h = gru_seq(gru_input)
```

```
gru_seq.weight_hh_l0
```

```
Parameter containing:
 0.0766 -0.0548 -0.2008  ...  -0.0250 -0.1819  0.1453
-0.1676  0.1622  0.0417  ...   0.1905 -0.0071 -0.1038
 0.0444 -0.1516  0.2194  ...  -0.0009  0.0771  0.0476
          ...             ⋱             ...
 0.1698 -0.1707  0.0340  ...  -0.1315  0.1278  0.0946
 0.1936  0.1369 -0.0694  ...  -0.0667  0.0429  0.1322
 0.0870 -0.1884  0.1732  ...  -0.1423 -0.1723  0.2147
[torch.FloatTensor of size 60x20]
```

```
h.shape
```

```
torch.Size([1, 32, 20])
```

```
out.shape
```

```
torch.Size([3, 32, 20])
```