LSTM 做词性预测

下面我们用例子来简单的说明

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

我们使用下面简单的训练集

```
training_data = [("The dog ate the apple".split(),
                  ["DET", "NN", "V", "DET", "NN"]),
                 ("Everybody read that book".split(),
                  ["NN", "V", "DET", "NN"])]
```

接下来我们需要对单词和标签进行编码

```
word_to_idx = {}
tag_to_idx = {}
for context, tag in training_data:
    for word in context:
        if word.lower() not in word_to_idx:
            word_to_idx[word.lower()] = len(word_to_idx)
    for label in tag:
        if label.lower() not in tag_to_idx:
            tag_to_idx[label.lower()] = len(tag_to_idx)
```

```
word_to_idx
```

```
{'apple': 3,
 'ate': 2,
 'book': 7,
 'dog': 1,
 'everybody': 4,
 'read': 5,
 'that': 6,
 'the': 0}
```

```
tag_to_idx
```

```
{'det': 0, 'nn': 1, 'v': 2}
```

然后我们对字母进行编码

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
char_to_idx = {}
for i in range(len(alphabet)):
    char_to_idx[alphabet[i]] = i
```

```
char_to_idx
```

```
{'a': 0,
 'b': 1,
 'c': 2,
 'd': 3,
 'e': 4,
 'f': 5,
 'g': 6,
 'h': 7,
 'i': 8,
 'j': 9,
 'k': 10,
 'l': 11,
 'm': 12,
```

```
 'n': 13,
 'o': 14,
 'p': 15,
 'q': 16,
 'r': 17,
 's': 18,
 't': 19,
 'u': 20,
 'v': 21,
 'w': 22,
 'x': 23,
 'y': 24,
 'z': 25}
```

接着我们可以构建训练数据

```
def make_sequence(x, dic): # 字符编码
    idx = [dic[i.lower()] for i in x]
    idx = torch.LongTensor(idx)
    return idx
```

```
make_sequence('apple', char_to_idx)
```

```
  0
 15
 15
 11
  4
[torch.LongTensor of size 5]
```

```
training_data[1][0]
```

```
['Everybody', 'read', 'that', 'book']
```

```
make_sequence(training_data[1][0], word_to_idx)
```

```
 4
 5
 6
 7
[torch.LongTensor of size 4]
```

构建单个字符的 lstm 模型

```
class char_lstm(nn.Module):
    def __init__(self, n_char, char_dim, char_hidden):
        super(char_lstm, self).__init__()

        self.char_embed = nn.Embedding(n_char, char_dim)
        self.lstm = nn.LSTM(char_dim, char_hidden)

    def forward(self, x):
        x = self.char_embed(x)
        out, _ = self.lstm(x)
        return out[-1] # (batch, hidden)
```

构建词性分类的 lstm 模型

```
class lstm_tagger(nn.Module):
    def __init__(self, n_word, n_char, char_dim, word_dim,
                 char_hidden, word_hidden, n_tag):
        super(lstm_tagger, self).__init__()
        self.word_embed = nn.Embedding(n_word, word_dim)
        self.char_lstm = char_lstm(n_char, char_dim, char_hidden)
        self.word_lstm = nn.LSTM(word_dim + char_hidden, word_hidden)
        self.classify = nn.Linear(word_hidden, n_tag)

    def forward(self, x, word):
        char = []
        for w in word: # 对于每个单词做字符的 lstm
            char_list = make_sequence(w, char_to_idx)
```

```
            char_list = char_list.unsqueeze(1) # (seq, batch, feature) 满足 lstm 输入
条件
            char_infor = self.char_lstm(Variable(char_list)) # (batch, char_hidden)
            char.append(char_infor)
        char = torch.stack(char, dim=0) # (seq, batch, feature)

        x = self.word_embed(x) # (batch, seq, word_dim)
        x = x.permute(1, 0, 2) # 改变顺序
        x = torch.cat((x, char), dim=2) # 沿着特征通道将每个词的词嵌入和字符 lstm 输出的
结果拼接在一起
        x, _ = self.word_lstm(x)

        s, b, h = x.shape
        x = x.view(-1, h) # 重新 reshape 进行分类线性层
        out = self.classify(x)
        return out
```

```
net = lstm_tagger(len(word_to_idx), len(char_to_idx), 10, 100, 50, 128,
len(tag_to_idx))
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=1e-2)
```

```
# 开始训练
for e in range(300):
    train_loss = 0
    for word, tag in training_data:
        word_list = make_sequence(word, word_to_idx).unsqueeze(0) # 添加第一维 batch
        tag = make_sequence(tag, tag_to_idx)
        word_list = Variable(word_list)
        tag = Variable(tag)
        # 前向传播
        out = net(word_list, word)
        loss = criterion(out, tag)
        train_loss += loss.data[0]
        # 反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    if (e + 1) % 50 == 0:
        print('Epoch: {}, Loss: {:.5f}'.format(e + 1, train_loss /
len(training_data)))
```

```
Epoch: 50, Loss: 0.86690
Epoch: 100, Loss: 0.65471
Epoch: 150, Loss: 0.45582
Epoch: 200, Loss: 0.30351
Epoch: 250, Loss: 0.20446
Epoch: 300, Loss: 0.14376
```

最后我们可以看看预测的结果

```
net = net.eval()
```

```
test_sent = 'Everybody ate the apple'
test = make_sequence(test_sent.split(), word_to_idx).unsqueeze(0)
out = net(Variable(test), test_sent.split())
```

```
print(out)
```

```
Variable containing:
-1.2148  1.9048 -0.6570
-0.9272 -0.4441  1.4009
 1.6425 -0.7751 -1.1553
-0.6121  1.6036 -1.1280
[torch.FloatTensor of size 4x3]
```

```
print(tag_to_idx)
```

```
{'det': 0, 'nn': 1, 'v': 2}
```

最后可以得到上面的结果，因为最后一层的线性层没有使用 softmax，所以数值不太像一个概率，但是每一行数值最大的就表示属于该类，可以看到第一个单词 'Everybody' 属于 nn，第二个单词 'ate' 属于 v，第三个单词 'the' 属于det，第四个单词 'apple' 属于 nn，所以得到的这个预测结果是正确的