Assignment 4 报告

1. Dataset

这次文本分类，我延用了 Lab2 的数据生成器，选用了 sklearn 中的 20 newsgroups dataset(https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html)。不同的是，lab2 中仅选取了其中的 4 个分类。为了后续训练的方便，我使用了大数据集，即 20 个分类的所有数据。

在数据处理的时候，我使用了 fastNLP 的 Dataset，Instance，Vocabulary 等类型来对数据做预处理。这些类型经过了非常好的封装，经过预处理后，原来的 sklearn 的 Dataset 类被处理成了 fastNLP 的 Dataset 类，这为后续使用 torch 进行训练提供了方便。

2. CNN & RNN

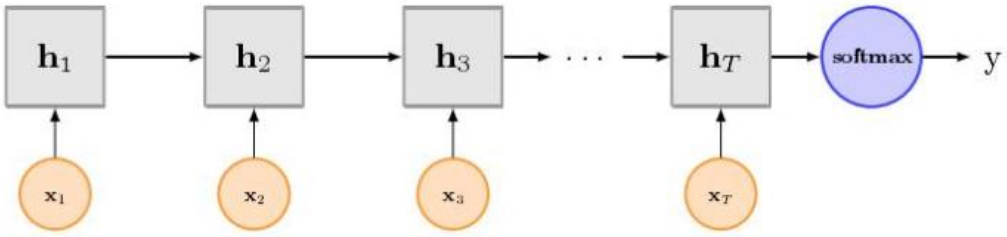
我使用了 fastNLP 和 pytorch 作为工具来实现两个神经网络，其中，在接入神经网络前，我将每一个字典里的单词 embedding 成一个 128 维的向量，128 作为神经网络的输入维度。神经网络的输出结果是一个长度维 20 的向量，分别代表每一类的结果。

在训练中，由于显存的限制，我取 batch 为(rnn: 16， cnn: 20)，这样做是因为 rnn 对显存的占用相对于 cnn 更高。batch 的实现使用了经过封装的 fastNLP.BucketSampler 类型。两个模型均使用了 softmax 交叉熵作为目标函数，即对应 pytorch 中的 CrossEntropyLoss。

测试时，使用了 fastNLP 的 AccuracyMetric 模块。因为文本长度不全一样，在处理时，对于每一个 batch，将 batch 之内的文本调节至了相同长度（扩大到

最长文本所对应的长度），这样方便模型内部的矩阵运算。

1） RNN



RNN 是在时序上深度的神经网络，擅长捕捉长的序列信息。对应到文本分类中，从某种意义上来说它更为擅长捕捉变长的 N-gram 信息。

本次 RNN 实现时选取了 fastNLP 内置的 LSTM 模块，以 embedding 之后的 128 维向量作为输入，隐藏层的输入和输出维度均为 128。之后，经过一个线性层得到 20 维的输出，即每个类别的概率。

Result:

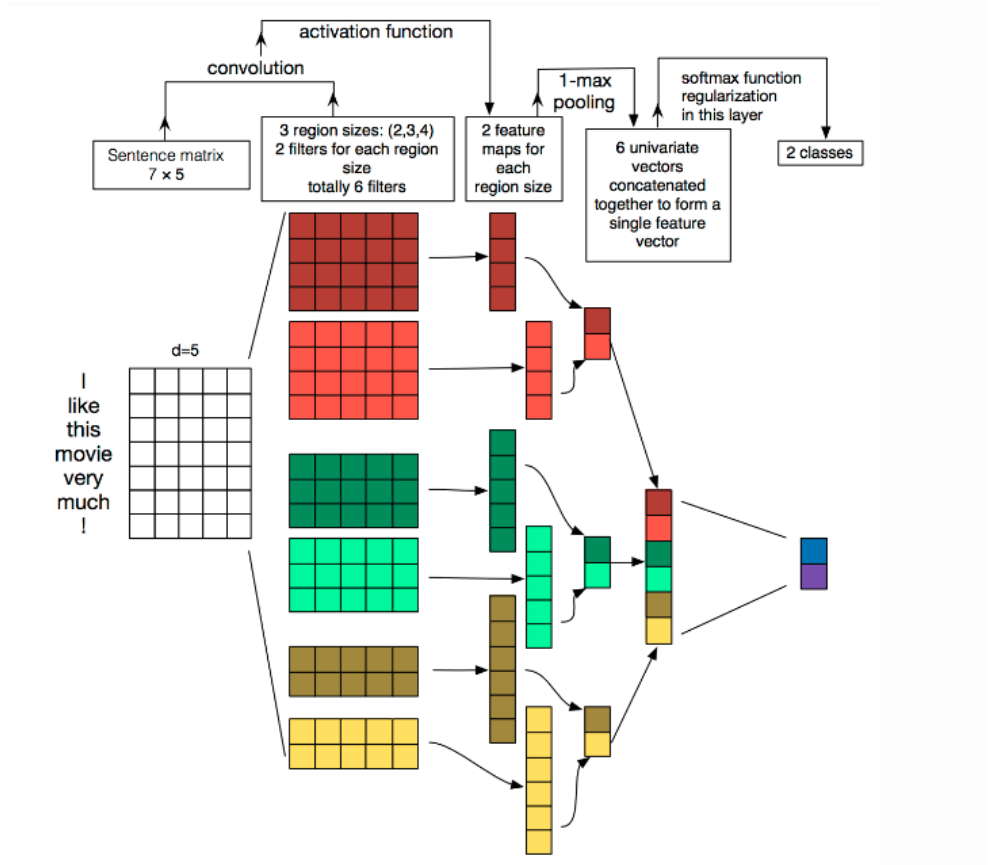
```
[info] Epoch 19 Iteration 697 Loss : 0.000591
[info] Epoch 19 Iteration 698 Loss : 0.000129
[info] Epoch 19 Iteration 699 Loss : 0.000200
[info] Epoch 19 Iteration 700 Loss : 0.000498
[info] Epoch 19 Iteration 701 Loss : 0.000296
[info] Epoch 19 Iteration 702 Loss : 0.000658
[info] Epoch 19 Iteration 703 Loss : 0.000095
[info] Epoch 19 Iteration 704 Loss : 0.000140
[info] Epoch 19 Iteration 705 Loss : 0.000151
[info] Epoch 19 Iteration 706 Loss : 0.000151
[info] Epoch 19 Iteration 707 Loss : 0.000057
```

```
[tester]
AccuracyMetric: acc=0.706851
```

在 20 个分类中能达到 71%的准确率

2） CNN

利用 CNN 处理文本分类问题的流程大致如下图：



可以看出，网络主要经过 3 层：

1. Convolution 层：这一层以 embedding 之后的向量作为输入，经过 3 个不同 Kernel size 的一位卷积，每个 Kernel size 有两个输出 channel
2. Maxpolling 层：这是一个 1-Max polling 层，这样不同长度的句子经过 pooling 层后都能变成定长的表示
3. Softmax 层：最后是一个全连接的 softmax 层，得出每个类别的概率

CNN 的实现同样依赖了 fastNLP 中的一些封装模块，其中 conv 层的 Kernel size=(3, 4, 5)。同时为了防止过拟合，在前向传播时使用了 Dropout，超参数 dropout = 0.1.

Result:

```
[info] Epoch 19 Iteration 555 Loss : 0.154102
[info] Epoch 19 Iteration 556 Loss : 0.066908
[info] Epoch 19 Iteration 557 Loss : 0.013489
[info] Epoch 19 Iteration 558 Loss : 0.233094
[info] Epoch 19 Iteration 559 Loss : 0.039188
[info] Epoch 19 Iteration 560 Loss : 0.250348
[info] Epoch 19 Iteration 561 Loss : 0.141806
[info] Epoch 19 Iteration 562 Loss : 0.289667
[info] Epoch 19 Iteration 563 Loss : 0.034465
[info] Epoch 19 Iteration 564 Loss : 0.014332
[info] Epoch 19 Iteration 565 Loss : 0.015232
```

```
[tester]
AccuracyMetric: acc=0.541954
```

准确率为 54.1%。

之所以没有 rnn 高，分析是因为文本长度比较长，而 rnn (尤其是这次选用的 lstm 网络)对长序列的记忆有着优势，准确率相对更高。

3. About fastNLP

在数据处理中，fastNLP 的 Instance，Vocabulary 等模块都非常便利，省下了狠多的代码复杂度；同时，fastNLP 的数据类型能够直接支持 torch 的运算，让开发者不用再利用 numpy 进行中间变换；并且类似这次的文本分类测试，fastNLP 能够提供完全封装好的函数。这是这个工具非常优秀的一点。

个人使用起来比较在意的一点在于 fastNLP 的文档有些过于简洁，希望在用例时能够提供用例的输入输出。这会让开发者更快更地好了解这个工具。

此外，如果可以，希望 fastNLP 能够封装一些诸如 loss 画图之类的功能，这样能够更有利于直观的了解训练进度。