DISCPURSE PARSING PROJECT

NLP 课程大作业实验报告

郑凌寒 -- 5130309485

2016 年 5 月

目 录

1. Introduction

This is a project relating to shallow discourse parsing. Shallow discourse parsing is the task of parsing a piece of text into a set of discourse relations between two adjacent of non-adjacent discourse units. However, in this project, we concentrate on implicit discourse relations where a discourse connective is omitted.

In fact, this is a classification problem, the input is a set of two sentences, and the output is a prediction of the the relationship between these two sentences. To solve the bottleneck and increase the accuracy, convolutional neural networks (CNN) is a recommended method.

CNN utilize layers with convolving filters that are applied to local features(LeCun et al.,1998) At the beginning, CNN models were invented for computer vision but now it have been shown to be effective for NLP and achieved some excellent results in this field. Thus in my experiment, I designed a CNN model to solve the problem. The idea comes from the paper of *Convolutional Neural Networks for Sentence Classification* by Yoon Kim.

During my project, I have tried both simple full-connected models and CNN models, and I will compare them in the report.

2. Tools

All my codes are written in Python language. And I have used some tools to help finish the task:

word2vec :

Translating words into word vectors is the first step of this project. To initialize word vectors, I used the publicly available word2vec. Its documentation can be found at the following site: https://radimrehurek.com/gensim/models/word2vec.html.

In the paper of (Yoon Kim, 2014), the vectors that were trained on 100 billion words from Google News were directly used. These vectors have dimensionality of 300 and were trained using the continuous bag-of-words architecture. I also planned to use this model at first, if so, the code should be:

```
from gensim.models import Word2Vec
wordmodel = Word2Vec.load_word2vec_format('GoogelNews-vectors-negative300.bin',
                                          binary=Ture)
```

"GoogleNews-vectors-negative300.bin" can be downloaded directly online.

However, unfortunately, since the memory of my computer is only 4G, if I chose this wordmodel to initialize the words, I guess it would cost me a complete one month! Isn't it scary? Hence, I gave up after wasting about five days on processing the data with the GoogleNews model and turned to train my own word vector.

My vectors were trained on the words from the “train_pdtb.json” file and “dev_pdtb.json” file and the dimension is 50. The limited trainning set and the relatively small dimension may affect the final result. But I have no choice because of the limited memory and limited time.

The code to train the wordmodel can be found at the “trainWordModel.py”

The idea is that, first, I extracted words form the json file, as the following figure shows. Similar for “dev_pdtb.json” file.

```
ifile = open("train_pdtb.json",'r')

for eachLine in ifile:
    js = None
    try:
        js = json.loads(eachLine)
    except Exception,e:
        print "bad line"
        continue

    wordlist = js["Arg2"]["Word"]
    sentences.append(wordlist)

    wordlist = js["Arg1"]["Word"]
    sentences.append(wordlist)

ifile.close()
```

Fig 1. Get the words from json file

Next, use all these words to train a word model.

```
#------------ word to vector -------------

from gensim.models import Word2Vec
from gensim.models.word2vec import *

min_count = 2
size = 50
window = 5

model = Word2Vec(sentences, min_count=min_count, size=size, window=window)
model.save('MyWordmodel.model')
model.save_word2vec_format('MyWordModel.bin',binary=True)
```

Fig 2. Train the word model

Now, I get my own word model and I save it as a “bin” file, thus I can load it to use in the future. The loading code is the same as loading GoogleNews Model.

NumPy :

NumPy is a general-purpose array-processing package designed to efficiently manipulate large multi-dimensional arrays of arbitrary records without sacrificing too much speed for small multi-dimensional arrays. In order to use Keras to implement my CNN, I have to use numpy arrays to represent all of the words.

There are some basic knowledge of NumPy that I should pay attention to.

- First, how to create an array?

There are several ways to create arrays, one of the simplest is to use the array function. For example:

```
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
```

Notice, if we write the instruction as "a=np.array(2,3,4)", it is wrong!

We can also create a two-dimensional arrays as:

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

And the function zeros could create an array full of zeros.

- Second, how to know the information about an exist array?

While learning from others' codes and checking my own result, I always have to know some information about the arrays. Good luck, it is very easy because an ndarray object has recorded all of its important attributes:

```
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
```

- Third, how to merge two or more arrays together?

Because I have to put the words of an argument together, and I have to put all the train samples together, I should know how to merge them. There exists a function called "concatenate", it can do this. Thanks for its help, now I can translate the word vectors into the format as I wish.

More information can be found at:
https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

Keras

The documentation of Keras: http://keras.io/

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running in top of either TensorFlow or Theano. It supports both convolutional networks and recurrent networks, as well as combinations of the two.

1) Models

There are two models Keras support, Sequential and Graph. This website http://www.friskit.me/2015/06/29/keras-mnist/ introduced the basic techniques of both models. Here, I chose Graph model because I want to develop a more flexible model instead of a fully linear stack of layers.

The function to add an input layer is :

add_input (name, input_shape, dtype='float'):往model里边增加一输入层

参数：

- **name**: 输入层的唯一字符串标识
- **input_shape**: 整数元组，表示新增层的shape。例如**(10,)**表示10维向量；**(None, 128)**表示一个可变长度的128维向量；**(3, 32, 32)** 表示一张3通道(RGB)的32*32的图片。
- **dtype:**float或者int；输入数据的类型。

The function to add an output layer is :

add_output (name, input=None, inputs=[], merge_mode=' concat'):增加一个连接到input或inputs的输出层

参数：

- **name**: 输出层的唯一字符串标识
- **input**: 输出层要连接到的隐层的名字。只能是input或inputs中的一个。
- **inputs:**新增层要连接到的多个隐层的名字列表。
- **merge_mode:** “sum”或”concat”。在指定inputs时有效，将不同的输入合并起来。

The function to add a hidden layer is :

add_node (layer, name, input=None,inputs=[], merge_mode='concat'):增加一个连接到input或inputs的输出层（就是除去输入输出之外的隐层）

参数：

- **layer**: Layer实例(Layer后边会介绍到)
- **name:**隐层的唯一字符串标识
- **input**: 新增隐层要连接到的某隐层或输入层的名字。只能是input或inputs中的一个。
- **inputs:**新增隐层要连接到的多个隐层的名字列表。
- **merge_mode:** “sum”或”concat”。在指定inputs时有效，将不同的输入合并起来。

And the function to train a model is :

fit(data, batch_size=128, nb_epoch=100, verbose=1, validation_split=0,validation_data=None, shuffle=True, callbacks=[]):用于训练一个固定迭代次数的模型

Where “data” is a dict describing the input and output layer. For example, {‘input’:X_train, ‘output’:Y_train}; “batch_size” denotes number of samples per gradient update; “nb_epoch” denotes the number of epochs to train the model. All the arguments are explained in detail in the documentation. You can get to know them if you want. One thing to stress, there is a “show_accuracy” arguments in Sequential Model but there isn’t in Graph model, so if using Graph model, the accuracy should be calculated by ourselves.

Finally the function to evaluate a model is :

evalute(data, batch_size=128, verbose=1): 展示模型在验证数据上的效果

返回：误差率

参数：和**fit**函数中的参数基本一致，其中verbose取1或0，表示有进度条或没有

Similarly, the function doesn't return the accuracy, and as a result I choose to use predict function to predict the results and calculate the accuracy on my own.

2) Layers

There are many different layers that Keras can support, such as Core Layers, Convolutional Layers, Recurrent Layers, Embedding Layers, Noise Layers and so on. Because I used CNN to finish my project, I only introduce Convolutional Layers now.

/* Convolution1D */

```
keras.layers.convolutional.Convolution1D(nb_filter,filter_length,
        init='uniform', activation='linear', weights=None,
        border_mode='valid', subsample_length=1,
        W_regularizer=None, b_regularizer=None, W_constraint=None,
        b_constraint=None, input_dim=None, input_length=None)
```

This convolution operator can filter neighborhoods of one-dimensional inputs. Among the arguments, "nb_filter" is number of convolution kernels to use, which is also the dimensionality of the output; "filter_length" is the extension of each filter; "activation" denotes the name of activation function to use.

In the convolution layer, the filter is applied to each possible window to produce a feature map. And then, a max-over-time pooling operation should be applied over the feature map in order to take the maximum value as the feature corresponding to this particular filter. Thus, we need add a max pooling layer here.

/* MaxPooling1d */

```
layers.convolutional.MaxPooling1D(pool_length=2, stride=None, ignore_border=True)
```

This is the max pooling operation for temporal data. The argument "pool_length" is the factor by which to down scale. Usually, if the length input is L, the window size is S, and the argument "border_mode" of Convolution1D is "valid", which means there is no need to pad, then pool_length should be L-S+1.

Now, by using Convolution1D and MaxPooling1D function, we can construct the convolution layer and thus get the features.

There is one other layer that I should mention, that is Dense Layer. Actually, it is the most simple layer, which is just a regular fully connected NN layer. But it is useful.

```
keras.layers.core.Dense(output_dim,init='glorot_uniform', activation='linear', weights=None
W_regularizer=None, b_regularizer=None, activity_regularizer=None,
W_constraint=None, b_constraint=None, input_dim=None)
```

The output_dim is the dimension of the output, and the activation sets the

activation function of this layer. According some related websites, usually, the number of nodes in this layer should be twice of the input nodes.

3. Implementation

From what has been talked above, now we cam get the word vector for every word, and I know how to construct a CNN, but the point is how to use the data I have? How to organize them and how to process them? In this section, I will introduce my implementation in detail.

Get train set :

First, I want to get a train set having the architecture as the following figure shows :

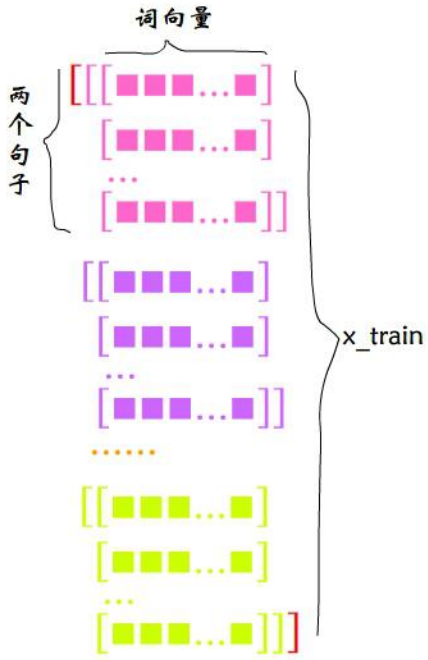

Fig 3. The architecture of train set

In this figure, every line represents a word vector which has a dimensionality of 50 in my project. Then a group of word vectors consist of a sentence. Here, different colors represent different sentences, including Arg1 and Arg2. All of them should have the same length in order to fit this architecture. Thus, I should pad zero vectors for short sentences and cut a part off from long sentences. And all of the sentences consist of the complete x_train set.

The source code to get x_train set and y_train set is “getTrainXY.py” file, and the source code to get x_test set and y_test set is “getTestXY.py” file. They are almost the same except that when we get train set, we have to know how many classes there

are and what are they, but when we get test set, we already know how many classes there are and we only have to judge which class it belongs to.

After almost one-day running, finally I got the x_train, y_train as following :

```
>>> x_train
array([[[ 0.8062973 ,  1.63731444, -0.54872191, ...,  0.74839604,
          0.75063396, -0.29271334],
        [ 0.09086199,  0.10592488, -0.04710583, ...,  0.03629664,
          0.02580731, -0.01674536],
        [ 0.32953757,  0.56096691, -0.25425592, ...,  0.28300747,
          0.16209285, -0.15828386],
        ...,
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ]],

       [[ 2.75225806, -0.01015351, -0.37119713, ...,  0.54151946,
         -0.1157146 ,  0.77481228],
        [ 2.06042624,  0.76162553, -1.2607832 , ...,  1.48364723,
          0.75065148,  0.38467622],
        [ 1.62229216,  0.04677755, -1.64512157, ..., -0.8180232 ,
          0.67519784,  1.44502056],
        ...,
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ]],

       [[ 0.86830902,  1.03667903, -0.05919283, ...,  0.54845679,
          0.37461528,  0.2683875 ],
        [ 1.14916778,  1.030761  , -0.84908098, ..., -0.31338203,
          0.68134284,  0.82331514],
        [ 0.60044044,  0.66388249, -0.61530626, ...,  0.20638497,
          0.32288525,  0.14850354],
        ...,
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ]],

       ...,
```

Fig 4. x_train

```
>>> y_train
array([ 0.,  1.,  0., ...,  5.,  5.,  5.])
```

Fig 5. y_train

The max length of Arg1 is 189 and the max length of Arg2 is 408. Considering that most sentences are actually shorter, I finally set every Argument is fixed to be 180-length and as a result the length of the whole sentence is 360.

Besides, there are total 101classes and y_train uses [0,10] to represent the class.

By the way, the 11 classes are :

[0] Comparison.Contrast (1653);
[1] Expansion. Conjunction(2968);
[2] Expansion.Instantiation (1176);
[3] Expansion. Restatement (2569);
[4] Temporal.Asynchronous (582);
[5] Contingency.Cause (3423);
[6] Expansion. List (345);
[7] Comparison.Concession (195);
[8] Expansion.Alternative (159);
[9] Temporal. Synchrony (213);
[10] Contingency.Pragmatic cause (68).

Hence, the x_train.shape is (13551L, 360L, 50L) and the y_train.shape is (13551,).

```
(13351L, 360L, 50L) (13351L,)
>>>
```

Train the model :

Next, I can finish the most important step in this project -- to train the model.

Actually, before I read the paper[2], I have tried to train a really simple full-connected mode. The idea is borrowed from the classification task of Mnist handwritten digit database.

First, I imitated the code for mnist project online to change the shape of input data as following.

```
X_train = x_train.reshape(x_train.shape[0],x_train.shape[1]* x_train.shape[2])
Y_train = (numpy.arange(11) == y_train[:,None]).astype(int)
```

After reshaping, X_train.shape=(13351L, 36000L), where 36000 = 180 * 50 and Y_train.shape=(13351L, 11L) .

Then, I set up a model with 36000 input nodes, 10000 hidden layer nodes and 10 output nodes:

```
model = Sequential()
model.add(Dense(500, input_dim=36000, init='uniform')) # 输入层，36000
model.add(Activation('tanh')) # 激活函数是tanh
model.add(Dropout(0.5)) # 采用50%的dropout

model.add(Dense(10000, init='uniform')) # 隐层节点10000个
model.add(Activation('tanh'))
model.add(Dropout(0.5))

# 输出结果是11个类别，所以维度是11
model.add(Dense(11, init='uniform'))
model.add(Activation('softmax')) # 最后一层用softmax

# 设定学习率（lr）等参数
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,nesterov=True)
```

```
# 使用交叉熵作为loss函数，就是熟知的log损失函数
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=500, nb_epoch=100,shuffle=True,
          verbose=1, show_accuracy=True, validation_split=0.3)
```

Fig 6. Set up a simple fully connected network

The Finally, I used the acm_scorer to test the F1 accuracy of this model on "dev_pdtb.json" data file. The result is as Fig 7. The model is saved as "Yoyo_sequential_arch" and the weights is saved as "Yoyo_sequential_weights.h5"

```
D:\Yoyofile\NLP\model_scorer\acm_scorer>python scorer.py result.json dev_pdtb.json

================================================
Evaluation for non-explicit discourse relations only (Implicit, EntRel, AltLex)
Sense classification--------------
*Micro-Average           precision 0.4536      recall 0.3489   F1 0.3944
Comparison.Contrast      precision 0.3333      recall 0.3571   F1 0.3448
Expansion.Alternative    precision 1.0000      recall 0.0000   F1 0.0000
Expansion.Conjunction    precision 0.6667      recall 0.3771   F1 0.4818
Expansion.Instantiation  precision 0.2973      recall 0.3929   F1 0.3385
Expansion.Restatement    precision 0.4167      recall 0.2941   F1 0.3448
Temporal.Synchrony       precision 0.0000      recall 0.0000   F1 0.0000
Overall parser performance --------------
Precision 0.4536 Recall 0.3489 F1 0.3944
```

Fig 7. The accuracy of my first simple sequential model

And then I changed the hidden layer to 36000 nodes, getting a new model "Yoyo_sequential_arch_v2" with "Yoyo_sequential_weights_v2.h5"

```
D:\Yoyofile\NLP\model_scorer\acm_scorer>python scorer.py result_2.json dev_pdtb.json

================================================
Evaluation for non-explicit discourse relations only (Implicit, EntRel, AltLex)
Sense classification--------------
*Micro-Average           precision 0.4220      recall 0.3190   F1 0.3634
Comparison.Concession    precision 0.0000      recall 0.0000   F1 0.0000
Comparison.Contrast      precision 0.3934      recall 0.3582   F1 0.3750
Expansion.Alternative    precision 0.0000      recall 0.0000   F1 0.0000
Expansion.Conjunction    precision 0.5481      recall 0.3585   F1 0.4335
Expansion.Instantiation  precision 0.3333      recall 0.3889   F1 0.3590
Expansion.Restatement    precision 0.4068      recall 0.2243   F1 0.2892
Temporal.Synchrony       precision 0.0000      recall 0.0000   F1 0.0000
Overall parser performance --------------
Precision 0.4220 Recall 0.3190 F1 0.3634
```

Fig 8. The accuracy of my second simple sequential model

In addition, when there are 20000 hidden layer nodes, F1 is 0.3823; when there are 10000 hidden layer nodes and dropout is 0.25, F1 is 0.4080. Actually, the accuracy is not bad, even very close to the baseline. Thus I am looking forward to seeing the performance of a CNN model.

Now I will begin to introduce the CNN model.

Figure 9 shows a draft of my ideal network. Here, I set nb_filter as 2 for simple, but it is not enough in practice.

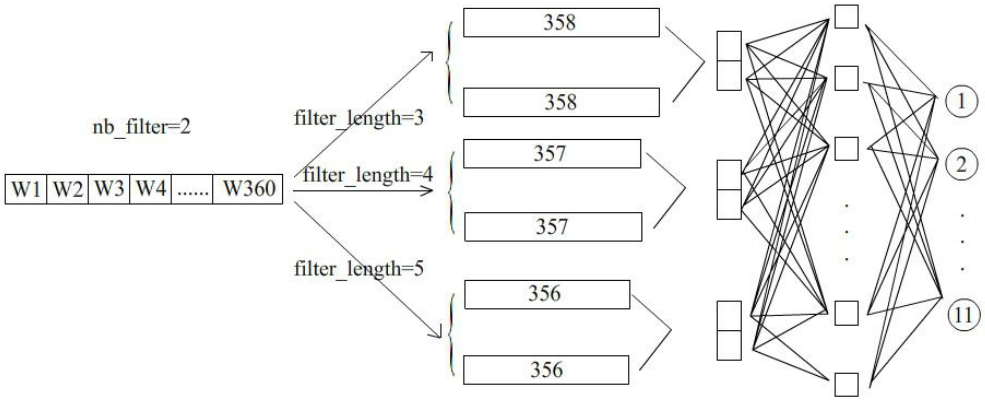

Fig 9. A draft of the network I design

In this network, the input is a vector which contains 360 word vectors, i.e. one train sample. Then the model uses multiple filters (with varying window sizes to obtain multiple features). Then, these filters produces different outputs. For example, when filter_length equals to 3, and now the nb_filter equals 2, we can get 2 vectors, each of them has a length of 358 (360-3+1). Apply max pooling layer on these vectors, we can extract 2 features. In general, suppose nb_filter=n, and we have k different filter_lengths, then, there will be n*k features in total. Next, I have developed a hidden layer, which is a dense layer. Finally, the output layer is 11 nodes represents 11 different classes.

The source code of this part is in “trainModel.py” file.

To get a better model, I have to spend a lot of time trying and trying again, looking forward to getting more suitable parameters.

There are some different results with different parameters I got during the project :

- nb_filter = 250, filter_lengths = [2,3,4], hidden layer nodes = 1800

```
D:\Yoyofile\NLP\model_scorer\acm_scorer>python scorer.py result_CNN_1.json dev_pdtb.json

================================================
Evaluation for non-explicit discourse relations only (Implicit, EntRel, AltLex)
Sense classification--------------
*Micro-Average            precision 0.3770      recall 0.2778   F1 0.3199
Comparison.Contrast       precision 0.4107      recall 0.1933   F1 0.2629
Expansion.Conjunction     precision 0.5514      recall 0.3598   F1 0.4354
Expansion.Instantiation   precision 0.1613      recall 0.2941   F1 0.2083
Expansion.Restatement     precision 0.1379      recall 0.1905   F1 0.1600
Overall parser performance --------------
Precision 0.3770 Recall 0.2778 F1 0.3199
```

Fig 10. The result of first CNN model

- nb_filter = 300, filter_lengths = [2,3,4], hidden layer nodes = 1800, nb_epoch = 30 => F1 = 0.3933

- nb_filter = 300, filter_lengths = [2,3,4], hidden layer nodes = 1000, nb_epoch = 30 => F1 = 0.3625

- nb_filter = 360, filter_lengths = [2,3,4], hidden layer nodes = 1800, nb_epoch = 50 => F1 = 0.2785

- nb_filter = 360, filter_lengths = [2,3,4], hidden layer nodes = 2000, nb_epoch = 50 => F1 = 0.3823

Due to the limited time, it is a pity that I still cannot get a model that can get the F1 value better than 46% on dev_pdtb.json.

Ironically, I spend too much time on training these CNN models but they are not only time-costed but also low-accurate. If forget about the risk that I will not pass the course, I will say that I don't regret my choice because I learn a lot during this experience, including the knowledge about word vector, various of tools and CNN models. However, considering that I may get a low score even though I spent two months and tried very hard, I still feel very upset.

4. Something to say

开始写报告的时候，考虑到这是一门英文 PPT 的课程，所以选择了英文写报告，现在看来，是我太作，辛苦了自己，更辛苦了看报告的助教学长，在这里说一句“非常抱歉”！所以最后我还是用中文写一点感想。

从五月开始着手完成这个 project 到现在，两个月的时间，总体感觉收获颇丰。从一开始的茫然无措毫无头绪，到后来经助教指导，开始学习 word2vec 词向量的使用，学习 keras 工具的使用，学习 CNN 的概念，学习 CNN 如何应用到 NLP 相关的领域，到最后出了至少可以使用的代码，觉得还是小有成就感。在这个过程当中，非常非常感谢王浩学长的耐心指导与帮助，给我这个一无所知的小白很多有实际用途的指导和鼓励，给我指明了方向，也向我介绍了很多有用的知识！

非常遗憾和可惜的是，我最后训练的 model 也没有在 dev 上跑出大于 40%的结果。全连接的网络每次只用跑半天，尚且能够达到 39%，但是当我花费了将近一个月的时间不停调参训练 CNN 模型，一次跑起来就是一周半周，却没能够取得理想的结果。对于这样的失败，我目前想到了几个可能可以改进的方法，首先

是调整数据的不平衡性，由于 train 中各个类型的样本数量差别较大，所以可能会影响模型训练；其次，参数还需要再调整，比如 nb_filter、hidden layer 的结点数等等，但是这实在太过耗时，又有点拼运气，我现在已经没时间和精力继续完成了。

可能是这个 project 拖得太长，到尝试到 17 周，还是只看到 38%的 F1 值时，感觉甚是疲惫，并且有点麻木，觉得达不达标都已经无所谓了。Anyway，现在说这些也没什么意义，就写到这里吧。

最后，致谢部分。首先必须再次感谢助教学长，真的很感谢你的帮助；然后感谢父母和男朋友，在我感到沮丧时安慰我鼓励我；最后感谢电脑，两个月里一直被一个蠢货“虐待”，总是在不停跑程序，都没有休息过，辛苦了。

以上，深夜写报告的胡言乱语。

2016.6.20

5. Reference

LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.