# Hierarchical Neural Networks for Document Classfication 论文复现

蔡令帆 2018104096

## 一、论文概况

九十年代以来，Internet 以惊人的速度发展起来，它容纳了海量的各种类型的数据和信息，包括文本、声音、图像等。文本数据与声音和图像数据相比，占用网络资源少，更容易上传和下载，这使得网络资源中的大部分是以文本（超文本）形式出现的。如何从这些浩瀚的文本中发现有价值的信息是信息处理的一大目标。基于机器学习的文本分类系统能够在给定的分类模型下，根据文本的内容自动对文本分门别类，从而更好地帮助人们组织文本、挖掘文本信息，因此得到日益广泛的关注，成为信息处理领域最重要的研究方向之一。

基于机器学习的文本分类系统首先通过在预先分类好的文本集上训练，建立一个判别规则或分类器，从而对未知类别的新样本进行自动归类。大量的结果表明它的分类精度比得上专家手工分类的结果，并且它的学习不需要专家干预，能适用于任何领域的学习，使得它成为目前文本分类的主流方法。

传统的机器学习方法对文本进行预处理之后一般采用基于词袋模型的 tf-idf 等向量表示方法，由于这种方法对文本的表示比较弱，得到的文本的表示也是高维稀疏的，所以需要进行降维处理，包括基于信息增益、互信息、$\chi 2$ 统计量特的征选择和对向量进行线性或非线性映射作特征提取。通过降维处理的文本就可以送入分类器进行分类，包括支持向量机、集成森林和贝叶斯分类器等常用的分类器，得到最后的分类结果。2013 年以来，神经网络在文本分类的任务上逐渐成为了主流方法之一。Word2vec 等预训练词向量模型为文本任务提供了一个具有很强语义信息的输入，而卷积神经网络和 LSTM/GRU 也已表现出了在序列任务上的聚合能力。前者可以像 n-gram 模型一样将前后的信息用一个滑动窗口整合在一起，而后者则是能够在长距离上捕获信息，对于前后顺序极其敏感的文本序列输入非常有

效。注意力机制的引入进一步推动了神经网络模型在文本任务以及自然语言处理上的发展。它不仅能够帮助模型更好地利用输入中的有效信息，而且还对神经网络模型的行为提供了一定的解释能力。此外一些基于图的模型（PTE、TextGCN）在一定程度上可以捕捉全局的词共现信息，在文本分类上也取得了不错的效果。

二、模型实现

本文介绍的“Hierarchical Neural Networks for Document Classfication”在传统注意力机制的基础上，考虑到了文本的内部结构：这篇论文将其与注意力机制结合，分别在词语级别上以及句子级别上做了注意力机制的建模，不仅提升了文本分类的效果，而且关注到了不同词语在不同类别上表征特性的变化。相较于直接使用 RNN 最后的输出，或是使用 Max-Pooling 或者 Average-Pooling 在 RNN 的所有输出上做聚合，使用注意力机制能够更好地利用不同时间点上 RNN 单元的输出，而且还为模型的行为提供了一定程度的解释。

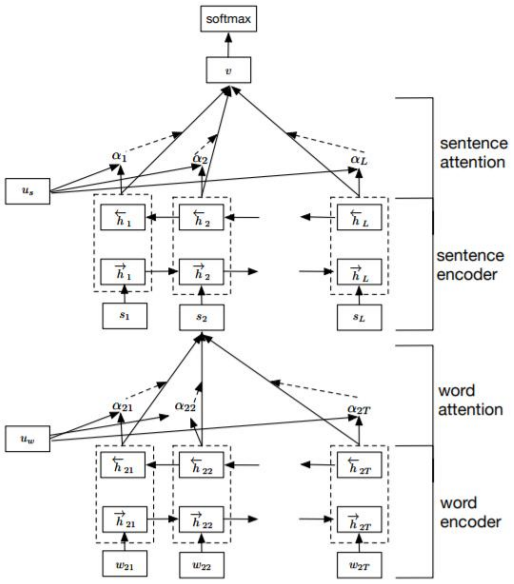
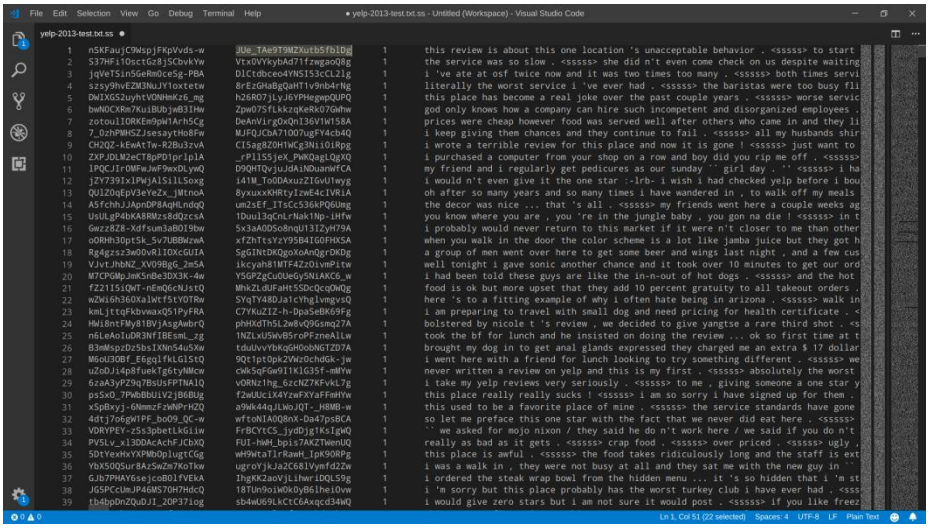



**Figure 2:** Hierarchical Attention Network.

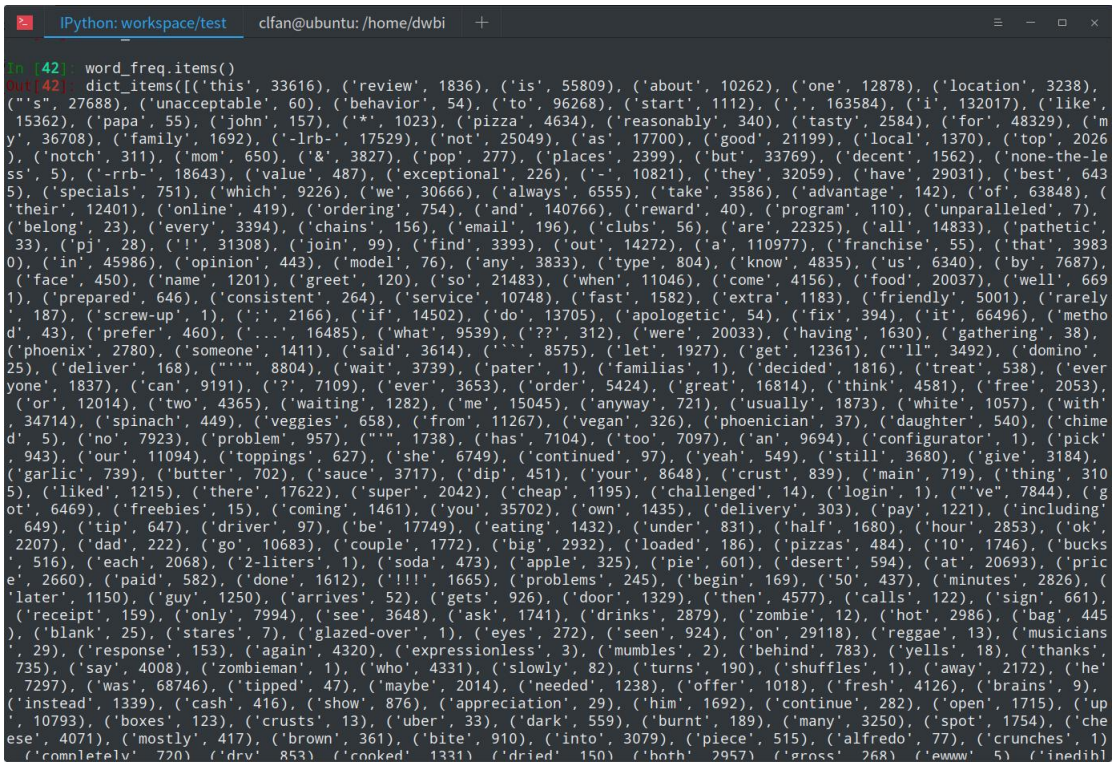
1、数据预处理

数据的原始格式：



取出数据的文本和对应的标记，并对文本数据做处理：按分隔符切割文本，形成列表，并根据训练集列表建立词频字典，根据词频字典去掉低频词（词频小于 5），并根据词频字典建立词-索引和索引-词字典，并根据词索引字典将文本表示成数字的列表，未在字典中的词都被填补成 1。

词频字典：



索引字典：

```
In [47]: word_to_index
Out[47]:
{'PAD': 0,
 'UNK': 1,
 'this': 2,
 'review': 3,
 'is': 4,
 'about': 5,
 'one': 6,
 'location': 7,
 "'s": 8,
 'unacceptable': 9,
 'behavior': 10,
 'to': 11,
 'start': 12,
 ',': 13,
 'i': 14,
 'like': 15,
 'papa': 16,
 'john': 17,
 '*': 18,
 'pizza': 19,
 'reasonably': 20,
 'tasty': 21,
 'for': 22,
 'my': 23,
 'family': 24,
 '-lrb-': 25,
 'not': 26,
 'as': 27,
 'good': 28,
 'local': 29,
 'top': 30,
 'notch': 31,
 'mom': 32,
 '&': 33,
 'pop': 34,
 'places': 35,
 'but': 36,
 'decent': 37,
 '-rrb-': 38,
 'value': 39,
 'exceptional': 40,
 '-': 41,
```

```
Loading vocabulary ...
Vocabulary size = 89724
```

原始文本数据：

```
In [50]: data
Out[50]:
       4                                                     6
0      1  this review is about this one location 's unac...
1      1  the service was so slow . <sssss> she did n't ...
2      1  i 've ate at osf twice now and it was two time...
3      1  literally the worst service i 've ever had . <...
4      1  this place has become a real joke over the pas...
5      1  god only knows how a company can hire such inc...
6      1  prices were cheap however food was served well...
7      1  i keep giving them chances and they continue t...
8      1  i wrote a terrible review for this place and n...
9      1  i purchased a computer from your shop on a row...
10     1  my friend and i regularly get pedicures as our...
11     1  i would n't even give it the one star :-lrb- i...
12     1  oh after so many years and so many times i hav...
13     1  the decor was nice ... that 's all . <sssss> m...
14     1  you know where you are , you 're in the jungle...
15     1  i probably would never return to this market i...
16     1  when you walk in the door the color scheme is ...
17     1  a group of men went over here to get some beer...
18     1  well tonight i gave sonic another chance and i...
19     1  i had been told these guys are like the in-n-o...
20     1  food is ok but more upset that they add 10 per...
21     1  here 's to a fitting example of why i often ha...
22     1  i am preparing to travel with small dog and ne...
23     1  bolstered by nicole t 's review , we decided t...
24     1  took the bf for lunch and he insisted on doing...
25     1  brought my dog in to get anal glands expressed...
```

处理后的文本（单个文本示例：前面为所属类别标记，后面为文本内容）：

```
In [202]: print(train_data[0])

(1, [[185, 820, 2620, 2621, 2622, 140, 2, 459, 1], [42, 64, 72, 516, 549, 204, 1, 1264, 42, 251, 41, 478, 27, 26, 18
2, 2389, 2623, 13, 1281, 225, 13, 1408, 13, 2427, 140, 2624, 41, 55, 410, 251, 147, 1, 51, 78, 902, 140, 52, 295, 26
25, 149, 315, 710, 72, 172, 619, 197, 2626, 1015, 1], [597, 979, 2627, 2628, 11, 295, 1970, 103, 133, 2629, 103, 149
, 1865, 658, 1164, 51, 874, 11, 284, 13, 1396, 42, 43, 315, 2630, 363, 1]])
```

加载预处理的 Glove 词向量，并根据词序列字典形成预训练词向量矩阵（200 维），查询不到的词的向量就随机初始化一个向量

```
Loading Glove pre-trained word embeddings ...
Total 400000 word vectors in glove.6B.200d.txt
Number of OOV words = 5266
```

2、模型构建：

模型的输入是一批处理之后的文本，和他们对应的标记，这一批文本数据都要经过词级别和句句子级别的补全，这里的每一篇文章被补全为[30，50]的矩阵，同时还有变量记录了每篇文章真正的句子数[一维列表]，和每个句子的单词数[二维矩阵]。

Embeeding 层：

```
def _init_embedding(self):
  with tf.variable_scope('embedding'):
    self.embedding_matrix = tf.get_variable(name='embedding_matrix',
                                            shape=[self.vocab_size, self.emb_size],
                                            initializer=tf.constant_initializer(self.pretrained_embs),
                                            dtype=tf.float32)
    self.embedded_inputs = tf.nn.embedding_lookup(self.embedding_matrix, self.docs)
```

对每篇文章的词找到对应的预训练向量，每篇文章则变成了[30,50,200]的三维张量。接下来要先经过 word 级别的编码处理。

Word encoding 层

```
def _init_word_encoder(self):
  with tf.variable_scope('word-encoder') as scope:
    word_inputs = tf.reshape(self.embedded_inputs, [-1, self.max_word_length, self.emb_size])
    word_lengths = tf.reshape(self.word_lengths, [-1])

    # word encoder
    cell_fw = rnn.GRUCell(self.cell_dim, name='cell_fw')
    cell_bw = rnn.GRUCell(self.cell_dim, name='cell_bw')

    init_state_fw = tf.tile(tf.get_variable('init_state_fw',
                                            shape=[1, self.cell_dim],
                                            initializer=tf.constant_initializer(0)),
                            multiples=[get_shape(word_inputs)[0], 1])
    init_state_bw  = tf.tile(tf.get_variable('init_state_bw',
                                            shape=[1, self.cell_dim],
                                            initializer=tf.constant_initializer(0)),
                            multiples=[get_shape(word_inputs)[0], 1])

    rnn_outputs, _ = bidirectional_rnn(cell_fw=cell_fw,
                                       cell_bw=cell_bw,
                                        inputs=word_inputs,
                                       input_lengths=word_lengths,
                                       initial_state_fw=init_state_fw,
                                       initial_state_bw=init_state_bw,
                                       scope=scope)

    word_outputs, word_att_weights = attention(inputs=rnn_outputs,
                                               att_dim=self.att_dim,
                                               sequence_lengths=word_lengths)
    self.word_outputs = tf.layers.dropout(word_outputs, self.dropout_rate, training=self.is_training)
```

由于对句子进行编码，可以先把当前 batch 调整成[batchsize×30，50, 200]，使用 GRUcell 作为基本单元，使用 bidirectional_dynamic_rnn 构建 word encode 层，将 batch 数据和记录每个句子实际长度的列表[batchsize×30]输入,经过双向 GRU 编码之后可以得到[batchsize*30, 50, 400]个隐状态向量，送入 attention 层进行加权平均处理。

```
def attention(inputs, att_dim, sequence_lengths, scope=None):
  assert len(inputs.get_shape()) == 3 and inputs.get_shape()[-1].value is not None

  with tf.variable_scope(scope or 'attention'):
    word_att_W = tf.get_variable(name='att_W', shape=[att_dim, 1])

    projection = tf.layers.dense(inputs, att_dim, tf.nn.tanh, name='projection')

    alpha = tf.matmul(tf.reshape(projection, shape=[-1, att_dim]), word_att_W)
    alpha = tf.reshape(alpha, shape=[-1, get_shape(inputs)[1]])
    alpha = masking(alpha, sequence_lengths, tf.constant(-1e15, dtype=tf.float32))
    alpha = tf.nn.softmax(alpha)

    outputs = tf.reduce_sum(inputs * tf.expand_dims(alpha, 2), axis=1)
    return outputs, alpha
```

这里的 attention 首先对每个隐状态向量经过一个全连接层得到一个隐含的表示[bachsize*30, 50, 100]，调整为[bachsize*30*50, 100]然后和一个 attention 矩阵[100,1] 相乘，经过调整得到每个隐状态向量的得分[bachsize*30, 50],根据每个句子真实长度的信息可以一个不等长的矩阵[bachsize*30，一行权重]，然后通过对原始的[bachsize*30，50，400]加权平均就可以得到这个 batch 里边所有句子的表示

[batchsize*30,400],调整为[batchsize ,30, 400]送到句子编码层处理。

Sentence Encoding 层

```
def _init_sent_encoder(self):
  with tf.variable_scope('sent-encoder') as scope:
    sent_inputs = tf.reshape(self.word_outputs, [-1, self.max_sent_length, 2 * self.cell_dim])

    # sentence encoder
    cell_fw = rnn.GRUCell(self.cell_dim, name='cell_fw')
    cell_bw = rnn.GRUCell(self.cell_dim, name='cell_bw')

    init_state_fw = tf.tile(tf.get_variable('init_state_fw',
                                            shape=[1, self.cell_dim],
                                            initializer=tf.constant_initializer(0)),
                            multiples=[get_shape(sent_inputs)[0], 1])
    init_state_bw = tf.tile(tf.get_variable('init_state_bw',
                                            shape=[1, self.cell_dim],
                                            initializer=tf.constant_initializer(0)),
                            multiples=[get_shape(sent_inputs)[0], 1])

    rnn_outputs, _ = bidirectional_rnn(cell_fw=cell_fw,
                                       cell_bw=cell_bw,
                                       inputs=sent_inputs,
                                       input_lengths=self.sent_lengths,
                                       initial_state_fw=init_state_fw,
                                       initial_state_bw=init_state_bw,
                                       scope=scope)

    sent_outputs, sent_att_weights = attention(inputs=rnn_outputs,
                                               att_dim=self.att_dim,
                                               sequence_lengths=self.sent_lengths)
    self.sent_outputs = tf.layers.dropout(sent_outputs, self.dropout_rate, training=self.is_training)
```

句子层的编码和词语层的处理基本一样，经过句子层的编码，每个文档表示成一个 800 维的向量，batch 被处理成[batchsize, 800]的矩阵，然后接全连连接层和 softmax 得到每个类别的概率，并根据 label 使用交叉熵计算损失。

```
def _init_classifier(self):
  with tf.variable_scope('classifier'):
    self.logits = tf.layers.dense(inputs=self.sent_outputs, units=self.num_classes, name='logits')
```

损失函数采用 softmax 回归交叉熵损失

```
def loss_fn(labels, logits):
  onehot_labels = tf.one_hot(labels, depth=FLAGS.num_classes)
  cross_entropy_loss = tf.losses.softmax_cross_entropy(onehot_labels=onehot_labels,
                                                       logits=logits)
  tf.summary.scalar('loss', cross_entropy_loss)
  return cross_entropy_loss
```

优化器采用 RMSPRO 优化器

```
  # Define optimizer
  global_step = tf.train.get_or_create_global_step()
  optimizer = tf.train.RMSPropOptimizer(FLAGS.learning_rate)
  train_op = optimizer.apply_gradients(zip(clipped_grads, trained_vars),
                                       name='train_op',
                                       global_step=global_step)
  return train_op, global_step
```

计算 batch accuracy 和 total accuracy

```
def eval_fn(labels, logits):
  predictions = tf.argmax(logits, axis=-1)
  correct_preds = tf.equal(predictions, tf.cast(labels, tf.int64))
  batch_acc = tf.reduce_mean(tf.cast(correct_preds, tf.float32))
  tf.summary.scalar('accuracy', batch_acc)

  total_acc, acc_update = tf.metrics.accuracy(labels, predictions, name='metrics/acc')
  metrics_vars = tf.get_collection(tf.GraphKeys.LOCAL_VARIABLES, scope="metrics")
  metrics_init = tf.variables_initializer(var_list=metrics_vars)
```

训练代码：

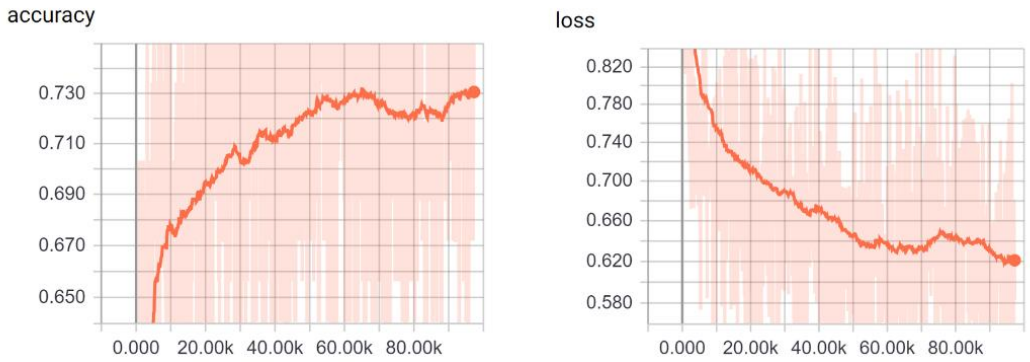
```
while epoch < FLAGS.num_epochs:
  epoch += 1
  print('\n{}> Epoch: {}'.format(datetime.now(), epoch))

  sess.run(metrics_init)
  for batch_docs, batch_labels in data_reader.read_train_set(FLAGS.batch_size, shuffle=True):
    _step, _, _loss, _acc, _ = sess.run([global_step, train_op, loss, batch_acc, acc_update],
                                        feed_dict=model.get_feed_dict(batch_docs, batch_labels, training=True))
    if _step % FLAGS.display_step == 0:
      _summary = sess.run(summary_op, feed_dict=model.get_feed_dict(batch_docs, batch_labels))
      train_writer.add_summary(_summary, global_step=_step)
  print('Training accuracy = {:.2f}'.format(sess.run(total_acc) * 100))

  sess.run(metrics_init)
  for batch_docs, batch_labels in data_reader.read_valid_set(test_batch_size):
    _loss, _acc, _  = sess.run([loss, batch_acc, acc_update], feed_dict=model.get_feed_dict(batch_docs, batch_labels
        ))
    valid_step += 1
    if valid_step % FLAGS.display_step == 0:
      _summary = sess.run(summary_op, feed_dict=model.get_feed_dict(batch_docs, batch_labels))
      valid_writer.add_summary(_summary, global_step=valid_step)
  print('Validation accuracy = {:.2f}'.format(sess.run(total_acc) * 100))
```

训练过程：

```
clfan@clfan-PC: ~/workspace/test    clfan@ubuntu: ~/hierarchical-attention-networks

2019-06-03 18:56:16.712174> Start training

2019-06-03 18:56:16.712251> Epoch: 1
Training: 100%|| 524/524 [05:03<00:00,  2.07it/s]
Training accuracy = 45.57
Validating: 100%|| 524/524 [00:28<00:00, 18.50it/s]
Validation accuracy = 58.49
Testing: 100%|| 524/524 [00:27<00:00, 13.03it/s]
Testing accuracy = 57.95
Best testing accuracy = 57.95

2019-06-03 19:02:17.227761> Epoch: 2
Training: 100%|| 524/524 [04:59<00:00,  2.00it/s]
Training accuracy = 59.02
Validating: 100%|| 524/524 [00:28<00:00, 18.69it/s]
Validation accuracy = 62.06
Testing: 100%|| 524/524 [00:27<00:00, 18.91it/s]
Testing accuracy = 61.93
Best testing accuracy = 61.93

2019-06-03 19:08:13.659501> Epoch: 3
Training: 100%|| 524/524 [05:00<00:00,  1.94it/s]
Training accuracy = 61.78
Validating: 100%|| 524/524 [00:27<00:00, 18.79it/s]
Validation accuracy = 62.61
Testing: 100%|| 524/524 [00:27<00:00, 18.86it/s]
Testing accuracy = 62.71
Best testing accuracy = 62.71

2019-06-03 19:14:10.057287> Epoch: 4
Training: 100%|| 524/524 [04:58<00:00,  1.90it/s]
Training accuracy = 63.31
Validating: 100%|| 524/524 [00:27<00:00, 11.01it/s]
Validation accuracy = 64.48
Testing: 100%|| 524/524 [00:27<00:00, 18.86it/s]
Testing accuracy = 64.29
Best testing accuracy = 64.29

2019-06-03 19:20:05.155041> Epoch: 5
Training: 100%|| 524/524 [04:58<00:00,  2.21it/s]
Training accuracy = 64.25
Validating: 100%|| 524/524 [00:27<00:00, 10.99it/s]
Validation accuracy = 65.12
Testing: 100%|| 524/524 [00:27<00:00, 10.96it/s]
```

```
2019-06-03 20:48:25.763341> Epoch: 20
Training: 100%|
Training accuracy = 72.84
Validating: 100%|
Validation accuracy = 64.82
Testing: 100%|
Testing accuracy = 65.07
Best testing accuracy = 65.07
2019-06-03 20:54:18.580234 Optimization Finished!
Best testing accuracy = 66.17
```

训练结果




三、实验总结

本次实验的主要是对这一经典文本分类模型的复现，代码全部由 tensorflow 代码实现，没有使用高阶的 API，仔细研究了网上别人的实现，做了部分修改使得代码在最新的 tensorflow 版本上运行不会出现警告。通过这次实验加深了对 tensorflow 的了解，可以实现一些基本神经网络算法，了解了对文本数据的预处理和灵活的批处理，对文本分类也有了进一步的了解。由于这次试验并不是做不同算法性能的比较，没有注重算法的效果，没有对数据，模型的参数等做严格的控制和调整。这个报告主要按照一个（批）示例的文本在代码中被处理直到和 label 比较计算 loss 的前向过程展开，一些代码的细节并没有展示。