



同濟大學
TONGJI UNIVERSITY

CRF 自动分词实验报告

姓 名：

学 号：

所在院系：电子与信息工程学院

学科专业：计算机科学与技术

指导教师：卫志华 苗夺谦

二〇二二年三月

目录

第 1 章 CRF 分词算法	1
1.0.1 CRF 基本概念	1
1.0.2 算法构建思路	1
1.0.3 CRF 学习算法	3
1.0.4 CRF 预测算法	5
第 2 章 模型结构与细节优化	6
2.0.1 bilstm 时间序列优化	6
2.0.2 未登录词处理	7
2.0.3 非等长句子处理	8
第 3 章 模型运行截图与结果	9
3.0.1 训练语料库	9
3.0.2 训练过程	9
3.0.3 测试过程	9
第 4 章 未来提升与反思	13
附录 A Appendix	14
A.0.1 BiLSTM_CRF.py	14
A.0.2 data_process.py	17
A.0.3 training.py	19
A.0.4 predict.py	20
A.0.5 likelihood.py	21
A.0.6 parameters.py	24

第1章 CRF分词算法

1.0.1 CRF基本概念

CRF(更准确地说是Linear-chain CRF)是最大熵模型的sequence扩展、HMM的conditional求解。CRF假设标注序列Y在给定观察序列X的条件下，Y构成的图为一个MRF，即可表示成图：

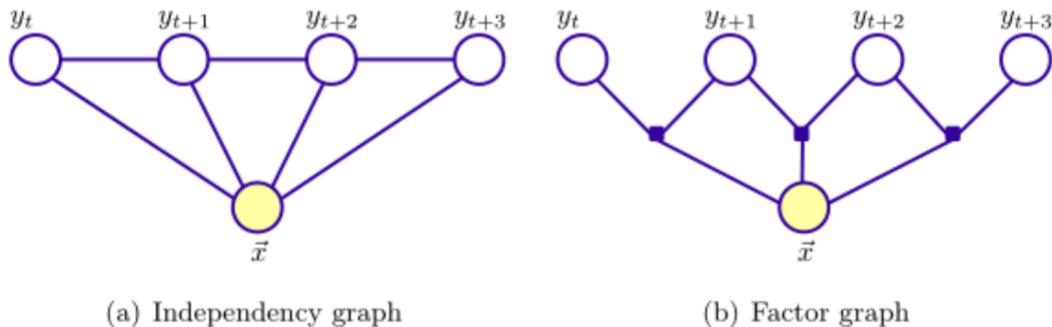


Figure 6: A Linear Chain Conditional Random Field

图 1.1 线性链条件随机场

最大熵模型与马尔可夫随机场(Markov Random Field, MRF)所对应factor graph都满足这样的因子分解：

$$P(Y | X) = \frac{\prod_j \Psi_j(\vec{x}, \vec{y})}{Z(\vec{x})}$$

同时可以根据Linear-chain CRF对随机场所作的Markov假设改写为：

$$P(Y | X) = \frac{1}{Z(X)} \exp \left(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, X, i) + \sum_{i,l} \mu_l s_l(y_i, x, i) \right)$$

其中， t_k, s_l 分别为转移特征与状态特征，而 λ_k, μ_l 为对应权重，即为学习目标。 $Z(X)$ 为规范化因子，其中，求和是在所有可能的输出序列上进行的，即遍历 Y 的全排列，在这里共有 4^L 种可能， L 表示句子长度。

1.0.2 算法构建思路

在线性链CRF参数化形式中，未知的只有特征与对应参数(或称为权重)，而参数是在学习问题中关注的，因此主要问题在于为中文分词任务构建特征。

这里，我借鉴了 CRF++ 的思路，首先，明确特征函数的含义：它描述是否满足指定特征，满足返回 1 否则返回 0；再来看特征模板：

```

1 # Unigram      --> 构造状态特征的模板
2 U00:%x[-2,0]
3 U01:%x[-1,0]
4 U02:%x[0,0]
5 U03:%x[1,0]
6 U04:%x[2,0]
7 U05:%x[-2,0]/%x[-1,0]/%x[0,0]
8 U06:%x[-1,0]/%x[0,0]/%x[1,0]
9 U07:%x[0,0]/%x[1,0]/%x[2,0]
10 U08:%x[-1,0]/%x[0,0]
11 U09:%x[0,0]/%x[1,0]
12
13 # Bigram      --> 构造转移特征的模板
14 B

```

我们的特征共有两种类型，即状态特征与转移特征：状态特征函数

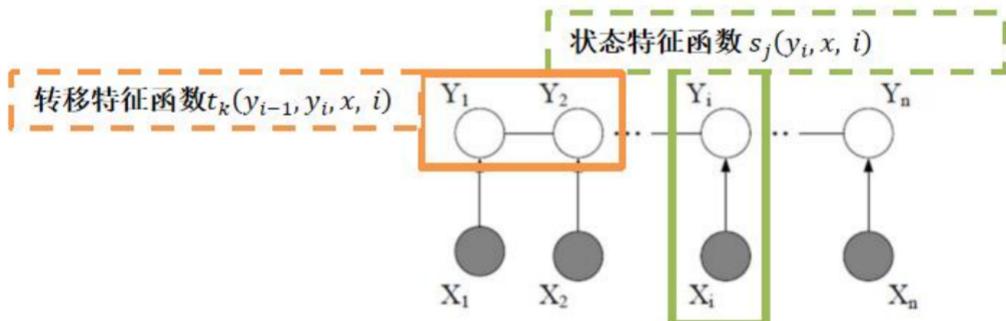


图 1.2 状态特征与转移特征

$s_i(y_i, X, i)$ ，它的输入为当前状态、整个观测序列以及当前位置。上面特征模板中 U01 ~ U09 用于生成状态特征。以 U00:%x[-2,0] 为例，-2 代表取观测序列中相对当前位置的倒数第 2 个，0 在此处无意义。所以，用 U00 ~ U09 做模板在我爱我的祖国。第 3 个位置下可构建的状态特征为：

```

1 我
2 爱
3 我    <-- 当前处于此位置
4 的
5 祖
6 国
7 # 下面为生成的状态特征
8 U00:我
9 U01:爱
10 U02:我
11 U03:的

```

```

12 U04:祖
13 U05:我/爱/我
14 U06:爱/我/的
15 U07:我/的/祖
16 U08:爱/我
17 U09:我/的

```

注：每行代表4个状态特征，因为这里有4标签（BMES）。所以，这里一共建产生40个状态特征。

注：请注意，这只是在第3个位置下产生的。转移特征 $t_k(y_{i-1}, y_i, X, i)$ ，它的输入为上一状态、当前状态、整个观测序列以及当前位置。上面特征模板中，B表示生成所有的转移特征。因此一共可以生成16种特征，即：

```

1 BB --> 表示从上一个状态 B 转移到当前状态 B
2 BM
3 BE
4 BS
5 MB
6 MM
7 ME
8 MS
9 EB
10 EM
11 EE
12 ES
13 SB
14 SM
15 SE
16 SS

```

因此，本实验中，每一个中文字对应四种状态特征之一：

```

1 TypeDict = {
2     'B': 0,
3     'M': 1,
4     'E': 2,
5     'S': 3
6 }

```

并将输入字与状态特征一一映射。

1.0.3 CRF学习算法

线性链CRF参数化形式中，未知项为特征函数和对应权重，特征函数构建如上，CRF训练目标即为更新此类参数。基本思路为，根据已知训练数据集，获取经验概率分布 $\tilde{P}(X, Y)$ ，而求得条件分布，可以通过极大化训练数据的对数似然函数来求模型函数，通过迭代方式，不断优化对数似然函数改变量下界，实现极大化。算法描述如下：

输入: 特征函数 t_1, t_2, \dots, t_{K_1} , s_1, s_2, \dots, s_{K_2} ; 经验分布 $\tilde{P}(x, y)$;

输出: 参数估计值 \hat{w} ; 模型 $P_{\hat{w}}$.

(1) 对所有 $k \in \{1, 2, \dots, K\}$, 取初值 $w_k = 0$

(2) 对每一 $k \in \{1, 2, \dots, K\}$:

(a) 当 $k = 1, 2, \dots, K_1$ 时, 令 δ_k 是方程

$$\sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k T(x, y)) = E_{\tilde{P}}[t_k]$$

的解;

当 $k = K_1 + l$, $l = 1, 2, \dots, K_2$ 时, 令 δ_{K_1+l} 是方程

$$\sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} T(x, y)) = E_{\tilde{P}}[s_l]$$

的解, 式中 $T(x, y)$ 由式 (11.38) 给出.

(b) 更新 w_k 值: $w_k \leftarrow w_k + \delta_k$

(3) 如果不是所有 w_k 都收敛, 重复步骤(2).

图 1.3 条件随机场模型迭代尺度法

1. 算法输入为: 转移特征、状态特征和经验分布。其中 batch size 对应每次训练使用句子个数, 实际训练参数放置于 parameter.py 中。

以下为 CRF 对数似然函数的实现:

```

1 def crf_log_likelihood(inputs,
2                         tag_indices,
3                         sequence_lengths,
4                         transition_params=None):
5     # log_likelihood是对数似然函数, transition_params是转移概率矩阵
6     # crf_log_likelihood{inputs:[batch_size,max_seq_length,num_tags
7     ],
8     # tag_indices:[batchsize,max_seq_length],
9     # sequence_lengths:[real_seq_length]
10    # transition_params: A [num_tags, num_tags] transition matrix
11    # log_likelihood: A scalar containing the log-likelihood of the
12    given sequence of tag indices.
13    num_tags = inputs.get_shape()[2].value
14
15
16    if transition_params is None:
17        transition_params = vs.get_variable("transitions", [num_tags, num_tags])
18
19    sequence_scores = crf_sequence_score(inputs, tag_indices, sequence_lengths
20                                         ,
21                                         transition_params)
22    log_norm = crf_log_norm(inputs, sequence_lengths, transition_params)
23
24    log_likelihood = sequence_scores - log_norm

```

```
21     return log_likelihood, transition_params
```

1.0.4 CRF预测算法

现在已经有条件随机场 $P(Y|X)$ 以及输入序列（句子），要求条件概率最大的输出序列（标记序列） Y^* ，当然，遍历每一种类型组合也不是不可以，经典的做法是动态规划求解最优路径。此处便用的是著名的维特比算法。

```
1 def viterbi_decode(score, transition_params):
2     trellis = np.zeros_like(score)
3     backpointers = np.zeros_like(score, dtype=np.int32)
4     trellis[0] = score[0]
5
6     for t in range(1, score.shape[0]):
7         v = np.expand_dims(trellis[t - 1], 1) + transition_params
8         trellis[t] = score[t] + np.max(v, 0)
9         backpointers[t] = np.argmax(v, 0)
10
11    viterbi = [np.argmax(trellis[-1])]
12    for bp in reversed(backpointers[1:]):
13        viterbi.append(bp[viterbi[-1]])
14    viterbi.reverse()
15
16    viterbi_score = np.max(trellis[-1])
17    return viterbi, viterbi_score
```

第 2 章 模型结构与细节优化

2.0.1 bilstm 时间序列优化

在序列标注任务（中文分词 CWS，词性标注 POS，命名实体识别 NER 等）中，目前主流的深度学习框架是 BiLSTM+CRF。其中 BiLSTM 融合两组学习方向相反（一个按句子顺序，一个按句子逆序）的 LSTM 层，能够在理论上实现当前词即包含历史信息、又包含未来信息，更有利对当前词进行标注。BiLSTM 在时间上的展开图如下所示。

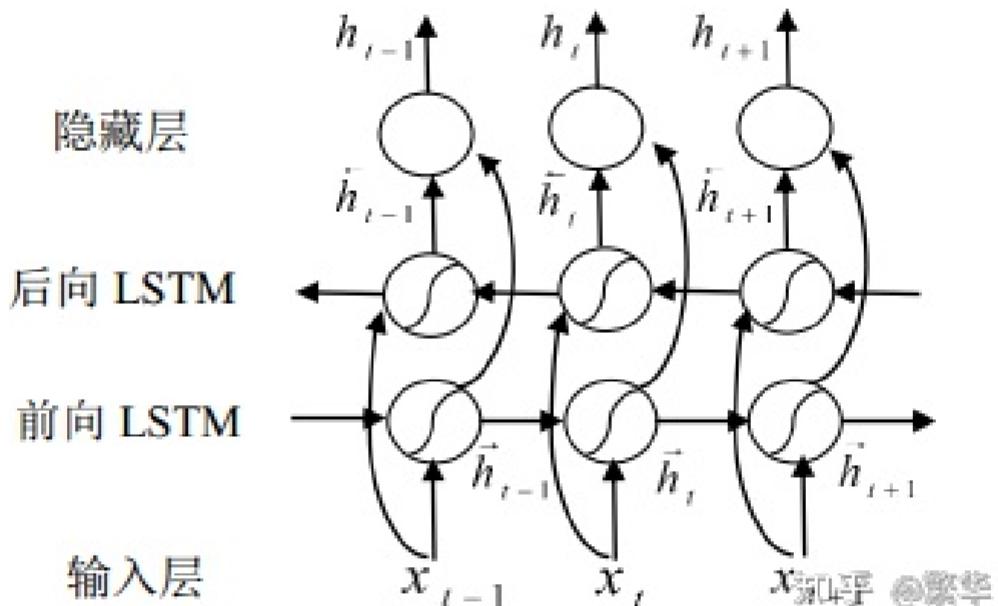


图 2.1 BiLSTM 在时间上展开

若输入句子由 120 个词组成，每个词由 100 维的词向量表示，则模型对应的输入是 $(120, 100)$ ，经过 BiLSTM 后隐层向量变为 $T1 (120, 128)$ ，其中 128 为模型中 BiLSTM 的输出维度。如果不使用 CRF 层，则可以在模型最后加上一个全连接层用于分类。设分词任务的目标标签为 B (Begin)、M (Middle)、E (End)、S (Single)，则模型最终输出维度为 $(120, 4)$ 的向量。对于每个词对应的 4 个浮点值，分别表示对应 BMES 的概率，最后取概率大的标签作为预测 label。通过大量的已标注数据和模型不断迭代优化，从而获得较好模型。

模型通过下述公式计算最优标注序列，A 矩阵是标签转移概率，P 矩阵是 BiLSTM 的预测结果：

$$s(X, y) = \sum_{i=0}^n A_{y_i, y_{i+1}} + \sum_{i=1}^n A_{P_i, y_i}$$

而对于每个序列 y 优化对数损失函数，调整 A 矩阵的值，这里不操作赘述。
相关代码如下：

```

1 with tf.name_scope('Cell'):
2     cell_fw = tf.nn.rnn_cell.LSTMCell(
3         para.hidden_dim, state_is_tuple=True)
4     Cell_fw = tf.contrib.rnn.DropoutWrapper(cell_fw, self.keep_pro)
5
6     cell_bw = tf.nn.rnn_cell.LSTMCell(
7         para.hidden_dim, state_is_tuple=True)
8     Cell_bw = tf.contrib.rnn.DropoutWrapper(cell_bw, self.keep_pro)
9
10    with tf.name_scope('biLSTM'):
11        outputs, _ = tf.nn.bidirectional_dynamic_rnn(cell_fw=Cell_fw,
12            cell_bw=Cell_bw, inputs=self.embedding,
13                                         sequence_length=
14                                         self.seq_length, dtype=tf.float32)
15        outputs = tf.concat(outputs, 2)

```

2.0.2 未登录词处理

这里参考了 Eliyahu Kiperwasser and Yoav Goldberg. 2016b 的做法，将所有未登录词登记为 <'UNK'>, UNK 词向量随机初始化，需要人在训练集中补充 UNK 标记字，采取词频法，根据 $z/(z + f(w))$ ，其中， $f(w)$ 为该词词频，当 $f(w) > 2$ 时应用此公式，通常 z 取 0.8375。对应代码：

```

1 def sequence2id(filename):
2     """
3     将文字与标签，转换为数字
4     """
5     content2id, label2id = [], []
6     _, content, label = read_file(filename)
7     with open('./data/word2id.pkl', 'rb') as fr:
8         key_dict = pickle.load(fr)
9     for i in label:
10        label2id.append([TypeDict[ii] for ii in i])
11
12    for j in content:
13        w = []
14        for key in j:
15            if key not in key_dict:
16                key = '<UNK>'
17            w.append(key_dict[key])
18        content2id.append(w)
19    return content2id, label2id

```

2.0.3 非等长句子处理

对于长短不一的句子，需要预先填充处理为长度一致的向量，记录原始长度，最终对 logit 结果再进行截断调用维特比算法，返回最优标注序列：

```

1 def process(x_batch):
2     """
3         计算一个batch中content最大长度，并且处理成等长
4     """
5     len_seq = []
6     max_len = max(map(lambda x: len(x), x_batch)) # 计算一个batch中最长长度
7     for i in x_batch:
8         len_seq.append(len(i)) # 原始长度
9
10    x_pad = kr.preprocessing.sequence.pad_sequences( # 处理成等长
11        x_batch, max_len, padding='post', truncating='post') # 默认填充0 --
12        key_dict['<PAD>']
13
14    return x_pad, len_seq

1 def predict(self, sess, x_batch):
2     seq_pad, seq_length = process(x_batch)
3     logits, transition_params = sess.run([self.logits, self.
4 transition_params], feed_dict={self.input_x: seq_pad,
5
6             self.seq_length: seq_length,
7
8             self.keep_pro: 1.0})
9     label_ = []
10    for logit, length in zip(logits, seq_length):
11        # logit 每个子句的输出值, length子句的真实长度, logit[:length] 的
12        # 真实输出值
13        # 调用维特比算法求最优标注序列
14        viterbi_seq, _ = viterbi_decode(logit[:length],
15        transition_params)
16        label_.append(viterbi_seq)
17
18    return label_

```

第3章 模型运行截图与结果

3.0.1 训练语料库

人民日报新闻语料库，内容示意如下：

```

data > WordSeg.txt
1 1986年，十亿中华儿女踏上新的征程。
2 过去的一年，是全国各族人民在中国共产党领导下，
3 在建设有中国特色的社会主义道路上，坚持改革、开放，团结奋斗、胜利前进的一年。
4 城乡经济体制改革向纵深稳步推进，对外开放迈出了新的步伐，工农业生产和其他各项建设事业全
5 政治上安定团结，端正党风和社会风气的工作取得了新的进展，社会主义民主和法制建设不断加强。
6 在党的十二届六中全会通过的《关于社会主义精神文明建设指导方针的决议》指引下，我国两个文
7 从党的十一届三中全会实现伟大历史转折到现在，我国政治安定团结，经济稳定、持续、协调发展
8 在十年动乱之后，取得这样一个大好局面是不容易的。
9 这是因为我们党在新时期制定的一系列路线、方针、政策正确，改革、开放符合全国人民的利益
10 坚持马列主义、毛泽东思想，坚持人民民主专政，坚持社会主义道路。
11 四项基本原则是我们立国的根本，是中国革命历史发展的必然结论。
12 中国没有共产党的领导，不搞社会主义，是没有前途的。
13 在改革、开放的新时期，同样是如此。
14 我们搞改革，无论是搞经济体制改革还是搞政治体制改革，都要在党的领导下进行，都是为了完
15 我们搞开放，无论是引进外资，引进外国的先进技术、先进管理经验，都是为了发展社会生产力
16 也是为了批判地吸取和概括各门科学发展的最新成果，而决不是意味着可以放弃马列主义、毛泽
17 中国今天所需要的民主，只能是社会主义民主或称人民民主，而不是资产阶级的个人主义的目
18 特别值得注意的是，在我国，剥削阶级虽然已经消灭，但是阶级斗争还将在一定范围内长期存在。
19 我们决不可以忘记对极少数敌视和破坏我国社会主义制度的敌对分子进行斗争，
20 任何时候，决不可放下人民民主专政这个武器。
21 坚持四项基本原则，
22 就要旗帜鲜明地反对资产阶级自由化。
23 近几年来，思想文化界有一些人借改革、开放之机，
24 发表各种背离四项基本原则的言论，使资产阶级自由化的思潮在一些地方泛滥开来，
25 而我们的一些同志表现软弱，不敢理直气壮地起来斗争，
26 这种情况再也不能继续下去了。
27 如果继续听任资产阶级自由化的思潮泛滥，党就失去了凝聚力和战斗力，怎么能成为全国人民的
28 中国又会成为一盘散沙，那还有什么希望？
29 党的十二届六中全会决议明确指出：“搞资产阶级自由化，即否定社会主义制度、主张资本主义制度
30 1987年摆在全党和全国各族人民面前的任务是很艰巨的。
31 我们要使经济持续、稳定地发展，要在精神文明建设方面办实事，要把经济体制改革继续推向前
32 还要为进行政治体制改革做调查、做准备。
33 让我们更加旗帜鲜明地反对资产阶级自由化思潮、坚持四项基本原则，搞好改革、开放，以更加士
34 迎接党的十三大的召开！
35 和平力量的增长超过了战争因素的增长，
36 维护世界和平的事业是大有希望的。
37 他说：“据林业部门估计，
38 缅甸约有野生象4000至5000只。”

```

图 3.1 人民日报新闻语料库

3.0.2 训练过程

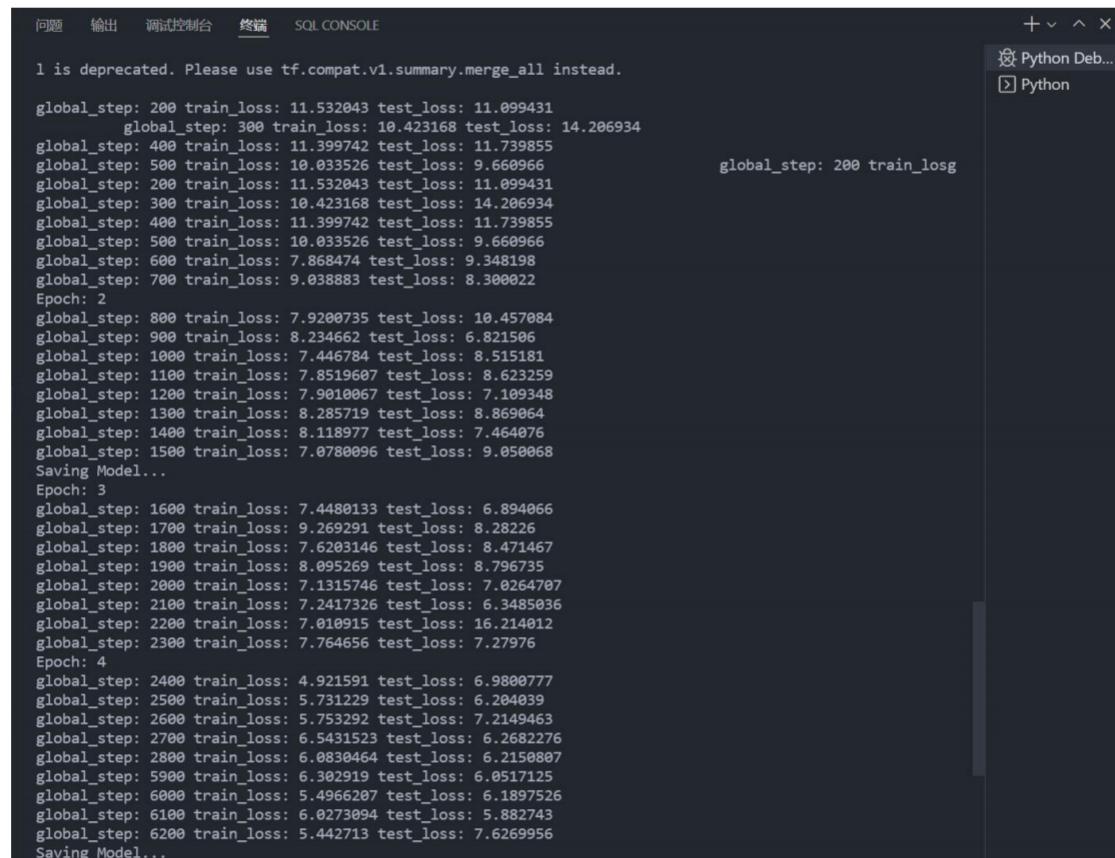
运行 training.py 即可，要求环境安装 tensorflow2.0+，以及 python3.7+，运行截图如下：

训练过程中保存的 checkpoint 以及 tensorboard：

3.0.3 测试过程

运行 predict.py 即可，要求同上，运行截图如下：

原测试文本：



The screenshot shows a Jupyter Notebook interface with a terminal tab open. The terminal output displays a sequence of training logs. The logs include epoch numbers (1, 2, 3, 4) and global step counts. For each epoch, multiple global steps are shown, each with a train loss and a test loss. The logs also mention saving models at certain intervals. A warning message at the top of the log states: "1 is deprecated. Please use tf.compat.v1.summary.merge_all instead."

```
1 is deprecated. Please use tf.compat.v1.summary.merge_all instead.

global_step: 200 train_loss: 11.532043 test_loss: 11.099431
global_step: 300 train_loss: 10.423168 test_loss: 14.206934
global_step: 400 train_loss: 11.399742 test_loss: 11.739855
global_step: 500 train_loss: 10.033526 test_loss: 9.660966
global_step: 200 train_loss: 11.532043 test_loss: 11.099431
global_step: 300 train_loss: 10.423168 test_loss: 14.206934
global_step: 400 train_loss: 11.399742 test_loss: 11.739855
global_step: 500 train_loss: 10.033526 test_loss: 9.660966
global_step: 600 train_loss: 7.868474 test_loss: 9.348198
global_step: 700 train_loss: 9.038883 test_loss: 8.300022
Epoch: 2
global_step: 800 train_loss: 7.9200735 test_loss: 10.457084
global_step: 900 train_loss: 8.234662 test_loss: 6.821586
global_step: 1000 train_loss: 7.446784 test_loss: 8.515181
global_step: 1100 train_loss: 7.8519607 test_loss: 8.623259
global_step: 1200 train_loss: 7.9010067 test_loss: 7.109348
global_step: 1300 train_loss: 8.285719 test_loss: 8.869064
global_step: 1400 train_loss: 8.118977 test_loss: 7.464076
global_step: 1500 train_loss: 7.0780096 test_loss: 9.050068
Saving Model...
Epoch: 3
global_step: 1600 train_loss: 7.4480133 test_loss: 6.894066
global_step: 1700 train_loss: 9.269291 test_loss: 8.28226
global_step: 1800 train_loss: 7.6203146 test_loss: 8.471467
global_step: 1900 train_loss: 8.095269 test_loss: 8.796735
global_step: 2000 train_loss: 7.1315746 test_loss: 7.0264707
global_step: 2100 train_loss: 7.2417326 test_loss: 6.3485036
global_step: 2200 train_loss: 7.010915 test_loss: 16.214012
global_step: 2300 train_loss: 7.764656 test_loss: 7.27976
Epoch: 4
global_step: 2400 train_loss: 4.921591 test_loss: 6.9800777
global_step: 2500 train_loss: 5.731229 test_loss: 6.204039
global_step: 2600 train_loss: 5.753292 test_loss: 7.2149463
global_step: 2700 train_loss: 6.5431523 test_loss: 6.2682276
global_step: 2800 train_loss: 6.0830464 test_loss: 6.2150807
global_step: 5900 train_loss: 6.302919 test_loss: 6.0517125
global_step: 6000 train_loss: 5.4966207 test_loss: 6.1897526
global_step: 6100 train_loss: 6.0273094 test_loss: 5.882743
global_step: 6200 train_loss: 5.442713 test_loss: 7.6269956
Saving Model...
```

图 3.2 训练截图

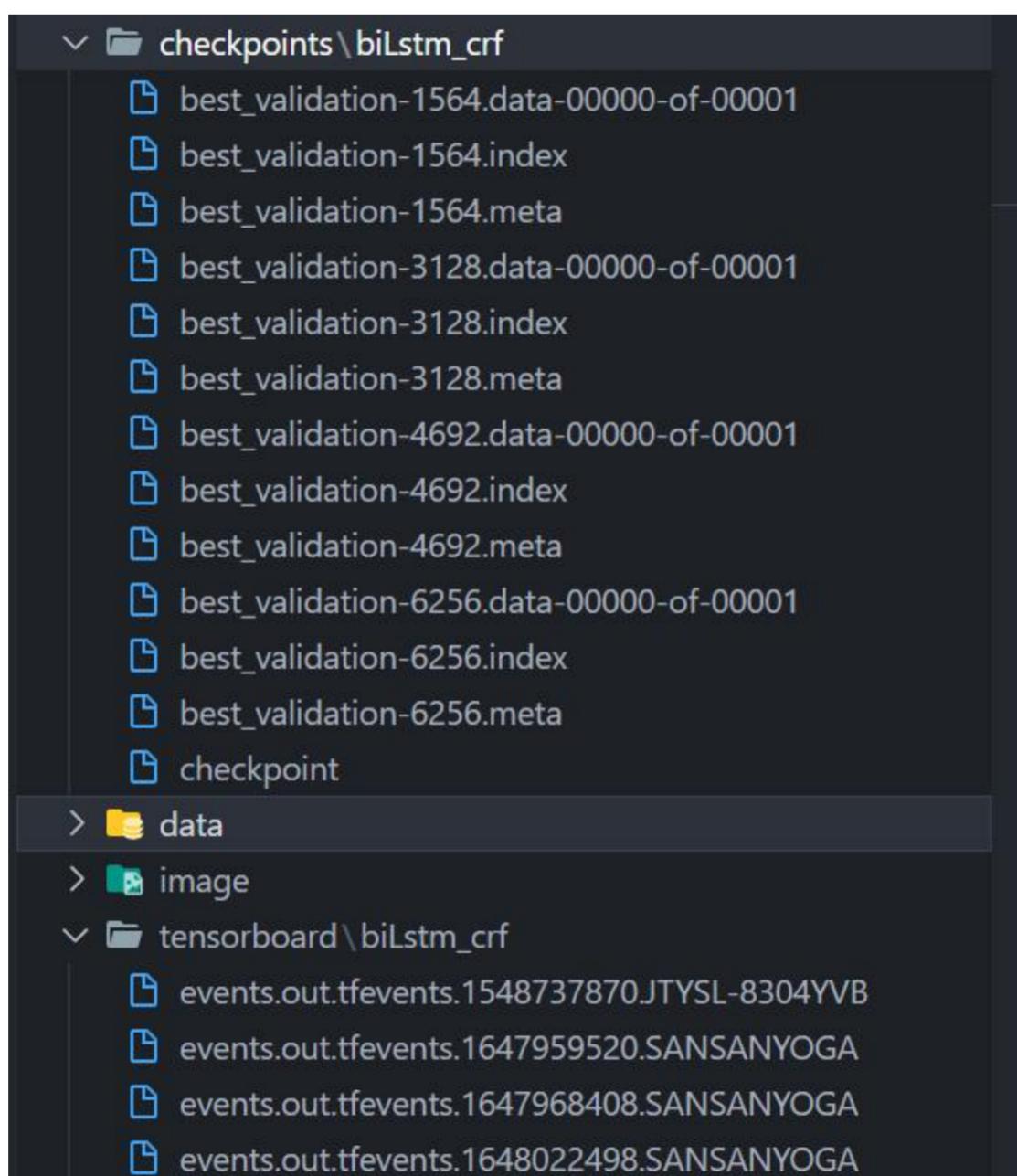


图 3.3 checkpoint 以及 tensorboard

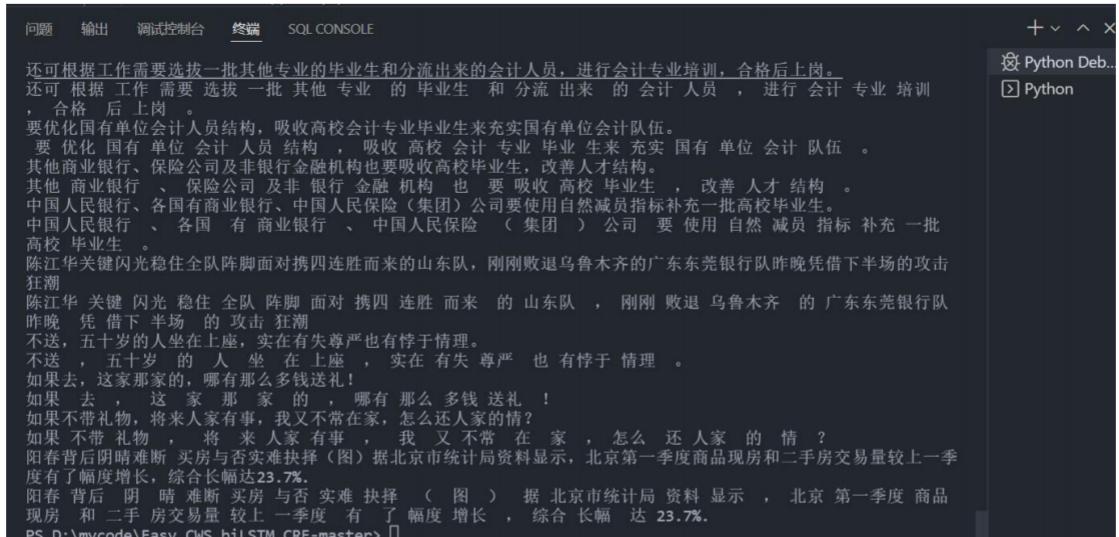


图 3.4 测试截图

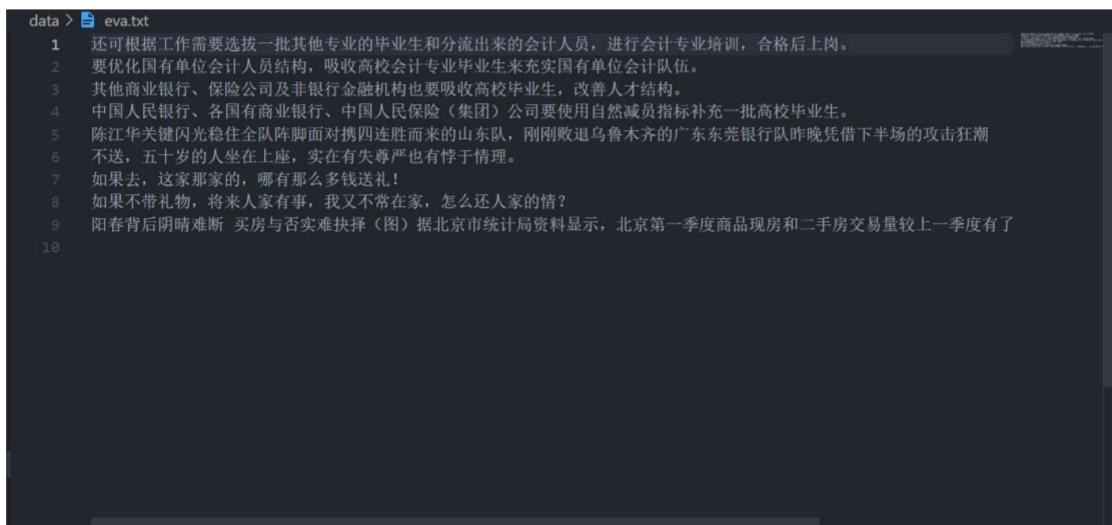


图 3.5 测试文本

第4章 未来提升与反思

本次实验准备时间有限，并且在学习马尔可夫模型、条件随机场模型上面花费了较多精力，甚至为此重温了概率论的相关知识。因此本次实验仍有很多不完善的地方，第一，未完成针对歧义词的处理，第二，未登录词可以进一步优化处理，目前是选择全部登记为 UNK 标识，但是根据测试结果不难发现，未登录词构成了识别错误的主要部分，后续的改进思路可以有：

1) 首先对全部预料集（测试机 + 训练集）进行 W2v，得到每个词的词向量，在实际预测时，若出现未登录词，则利用已经训练的词向量，将该未登录词替换为一个在词向量距离上和它最近的已登录词。但还是有问题，主要原因是，在迁移学习时，再次训练的语料数量有限，无法对大量的用于替换未登录词的已登录词进行训练。相比而言，之前只要遇到未登录词就将其换为 <UNK> 标签就不存在这种问题，因为只要在训练集中出现未登录词，就可以对该标签进行训练。

2) 将未登录词进行分型，如 1 个字的未登录词设置标签 A，2 个字的未登录词设置标签 B，以此类推这种方法是直接设置单一标签与方案 1 方法的折中方法。

这些想法可以留在后续补充。总而言之，这此我参考了很多前辈的写法以及 CRF++ 官方思路，这提升了我的信息获取和自学能力，同时，我也熟练了 tensorflow、keras 等机器学习框架的使用，获益匪浅。

附录 A Appendix

A.0.1 BiLSTM_CRF.py

```

1 from parameters import Parameters as para
2 from data_process import batch_iter, process
3 import tensorflow as tf
4 from likelihood import crf_log_likelihood
5 import numpy as np
6
7
8 def viterbi_decode(score, transition_params):
9     trellis = np.zeros_like(score)
10    backpointers = np.zeros_like(score, dtype=np.int32)
11    trellis[0] = score[0]
12
13    for t in range(1, score.shape[0]):
14        v = np.expand_dims(trellis[t - 1], 1) + transition_params
15        trellis[t] = score[t] + np.max(v, 0)
16        backpointers[t] = np.argmax(v, 0)
17
18    viterbi = [np.argmax(trellis[-1])]
19    for bp in reversed(backpointers[1:]):
20        viterbi.append(bp[viterbi[-1]])
21    viterbi.reverse()
22
23    viterbi_score = np.max(trellis[-1])
24    return viterbi, viterbi_score
25
26
27 class biLstm_crf(object):
28
29     def __init__(self):
30         self.input_x = tf.Variable([[0, 0]], dtype=tf.int32, shape=[None, None],
31                                   name='input_x')
32         self.input_y = tf.Variable([[0, 0]], dtype=tf.int32, shape=[None, None],
33                                   name='input_y')
34         self.seq_length = tf.Variable([0], dtype=tf.int32, shape=[None], name=
35                                       'seq_length')
36         self.keep_pro = tf.Variable([0], dtype=tf.float32, name='drop_out')
37         self.global_step = tf.Variable(0, trainable=False, name='global_step')
38
39         self.bils_crf()
40
41     def bils_crf(self):
42
43         with tf.device('/cpu:0'), tf.name_scope('embedding'):
44             embedding = tf.Variable(tf.truncated_normal(

```

```

41         [para.vocab_size, para.embedding_size], -0.25, 0.25), name='embedding')
42         embedding_input = tf.nn.embedding_lookup(embedding, self.input_x)
43         self.embedding = tf.nn.dropout(
44             embedding_input, keep_prob=self.keep_pro)
45
46     with tf.name_scope('Cell'):
47         cell_fw = tf.nn.rnn_cell.LSTMCell(
48             para.hidden_dim, state_is_tuple=True)
49         Cell_fw = tf.contrib.rnn.DropoutWrapper(cell_fw, self.keep_pro)
50
51         cell_bw = tf.nn.rnn_cell.LSTMCell(
52             para.hidden_dim, state_is_tuple=True)
53         Cell_bw = tf.contrib.rnn.DropoutWrapper(cell_bw, self.keep_pro)
54
55     with tf.name_scope('biLSTM'):
56         outputs, _ = tf.nn.bidirectional_dynamic_rnn(cell_fw=Cell_fw,
57 cell_bw=Cell_bw, inputs=self.embedding,
58                                         sequence_length=
59                                         self.seq_length, dtype=tf.float32)
60         outputs = tf.concat(outputs, 2)
61
62     with tf.name_scope('output'):
63         s = tf.shape(outputs)
64         output = tf.reshape(outputs, [-1, 2*para.hidden_dim])
65         output = tf.layers.dense(output, para.num_tags)
66         output = tf.contrib.layers.dropout(output, self.keep_pro)
67         self.logits = tf.reshape(output, [-1, s[1], para.num_tags])
68
69     with tf.name_scope('crf'):
70         self.log_likelihood, self.transition_params = crf_log_likelihood
71         (inputs=self.logits, tag_indices=self.input_y,
72
73         sequence_lengths=self.seq_length)
74         # log_likelihood是对数似然函数，transition_params是转移概率矩阵
75         # crf_log_likelihood{inputs:[batch_size,max_seq_length,num_tags
76         ],
77         # tag_indices:[batchsize,max_seq_length],
78         # sequence_lengths:[real_seq_length]
79         # transition_params: A [num_tags, num_tags] transition matrix
80         # log_likelihood: A scalar containing the log-likelihood of the
81         given sequence of tag indices.
82
83     with tf.name_scope('loss'):
84         self.loss = tf.reduce_mean(-self.log_likelihood) # 最大似然取
85         负，使用梯度下降
86
87     with tf.name_scope('optimizer'):
88         # 退化学习率 learning_rate = lr*(0.9**((global_step/10);staircase
89         =True表示每decay_steps更新梯度

```

```

82         # learning_rate = tf.train.exponential_decay(self.config.lr,
83         global_step=self.global_step,
84             # decay_steps=10, decay_rate=self.config.lr_decay, staircase=
85             True)
86         # optimizer = tf.train.AdamOptimizer(learning_rate)
87         # self.optimizer = optimizer.minimize(self.loss, global_step=
88         self.global_step) #global_step 自动+1
89         # no.2
90         optimizer = tf.train.AdamOptimizer(para.learning_rate)
91         gradients, variables = zip(
92             *optimizer.compute_gradients(self.loss)) # 计算变量梯度, 得
93         到梯度值, 变量
94         gradients, _ = tf.clip_by_global_norm(gradients, para.clip)
95         # 对g进行l2正则化计算, 比较其与clip的值, 如果l2后的值更大, 让梯
96         度*(clip/l2_g), 得到新梯度
97         self.optimizer = optimizer.apply_gradients(
98             zip(gradients, variables), global_step=self.global_step)
99         # global_step 自动+1
100
101     def feed_data(self, x_batch, y_batch, seq_length, keep_pro):
102         feed_dict = {self.input_x: x_batch,
103                     self.input_y: y_batch,
104                     self.seq_length: seq_length,
105                     self.keep_pro: keep_pro}
106         return feed_dict
107
108     def test(self, sess, x, y):
109         batch_test = batch_iter(x, y, batch_size=para.batch_size)
110         for x_batch, y_batch in batch_test:
111             x_batch, seq_length_x = process(x_batch)
112             y_batch, seq_length_y = process(y_batch)
113             feed_dict = self.feed_data(x_batch, y_batch, seq_length_x, 1.0)
114             loss = sess.run(self.loss, feed_dict=feed_dict)
115             return loss
116
117     def predict(self, sess, x_batch):
118         seq_pad, seq_length = process(x_batch)
119         logits, transition_params = sess.run([self.logits, self.
120         transition_params], feed_dict={self.input_x: seq_pad,
121
122                         self.seq_length: seq_length,
123
124                         self.keep_pro: 1.0})
125         label_ = []
126         for logit, length in zip(logits, seq_length):
127             # logit 每个子句的输出值, length子句的真实长度, logit[:length] 的
128             真实输出值
129             # 调用维特比算法求最优标注序列
130             viterbi_seq, _ = viterbi_decode(logit[:length],
131             transition_params)
132             label_.append(viterbi_seq)

```

```
123     return label_
```

A.0.2 data_process.py

```

1 # -*- coding:utf-8 -*-
2 import re
3 import keras as kr
4 import pickle
5 import numpy as np
6 from parameters import Parameters as para
7
8 TypeDict = {'B': 0, 'M': 1, 'E': 2, 'S': 3}
9
10
11 def convert_word_seq(sentence):
12     '''句子转换为字序列'''
13     sentence = ''.join(sentence.split(' '))  # 去除'
14     return [s for s in sentence]  # 返回字序列
15
16
17 def convert_BMES_seq(sentence):
18     '''句子转换为BMES序列'''
19     sentence = re.sub('  ', ' ', sentence)  # 处理2个空格
20     L = sentence.split(' ')
21     BMES_L = []
22     for l in L:
23         if len(l) == 1:
24             BMES_L.append('S')
25         elif len(l) == 2:
26             BMES_L.append('B')
27             BMES_L.append('E')
28         else:
29             BMES_L.append('B')
30             n_M = len(l)-2
31             for i in range(n_M):
32                 BMES_L.append('M')
33             BMES_L.append('E')
34     return BMES_L
35
36
37 def read_file(filename):
38     word, content, label = [], [], []
39     with open(filename, 'r', encoding='utf-8-sig') as f:
40         for l in f:
41             l = l.strip('\n').strip(' ')
42             word_seq = convert_word_seq(l)
43             BMES_seq = convert_BMES_seq(l)
44             word.extend(word_seq)
45             content.append(word_seq)
46             label.append(BMES_seq)
```

```
47     return word, content, label
48
49
50 def word_dict(filename):
51     """
52     从训练集获取所有字符的字典
53     """
54     word, _, _ = read_file(filename)
55     word = set(word)
56
57     key_dict = {}
58     key_dict['<PAD>'] = 0    # 填充符
59     key_dict['<UNK>'] = 1    # 低频词或未在词表中的词(未登录词)
60
61     j = 2
62     for w in word:
63         key_dict[w] = j
64         j += 1
65     with open('./data/word2id.pkl', 'wb') as fw:    # 将建立的字典 保存
66         pickle.dump(key_dict, fw)
67     return key_dict
68
69
70 def sequence2id(filename):
71     """
72     将文字与标签, 转换为数字
73     """
74     content2id, label2id = [], []
75     _, content, label = read_file(filename)
76     with open('./data/word2id.pkl', 'rb') as fr:
77         key_dict = pickle.load(fr)
78     for i in label:
79         label2id.append([TypeDict[ii] for ii in i])
80
81     for j in content:
82         w = []
83         for key in j:
84             if key not in key_dict:
85                 key = '<UNK>'
86             w.append(key_dict[key])
87         content2id.append(w)
88     return content2id, label2id
89
90
91 def batch_iter(content, label, batch_size=para.batch_size):
92     Len = len(content)
93     x = np.array(content)
94     y = np.array(label)
95     num_batch = int((Len-1) / batch_size) + 1    # 批数
96     indices = np.random.permutation(Len)    # [0,Len) 数字随机排序
97     # 混洗,打乱顺序
```

```

98     x_shuffle = x[indices]
99     y_shuffle = y[indices]
100    for i in range(num_batch):
101        start = i * batch_size
102        end = min((i+1) * batch_size, Len)
103        yield x_shuffle[start:end], y_shuffle[start:end]
104
105
106 def process(x_batch):
107     """
108         计算一个batch中content最大长度，并且处理成等长
109     """
110     len_seq = []
111     max_len = max(map(lambda x: len(x), x_batch)) # 计算一个batch中最长长度
112     for i in x_batch:
113         len_seq.append(len(i)) # 原始长度
114
115     x_pad = kr.preprocessing.sequence.pad_sequences( # 处理成等长
116         x_batch, max_len, padding='post', truncating='post') # 默认填充0 --
117         key_dict['<PAD>']
118
119     return x_pad, len_seq

```

A.0.3 training.py

```

1 import os
2 from parameters import Parameters as para
3 from biLSTM_CRF import biLstm_crf
4 import tensorflow as tf
5 from data_process import sequence2id, process, batch_iter
6
7 def train():
8     tensorboard_dir = './tensorboard'
9     save_dir = './checkpoints'
10    if not os.path.exists(tensorboard_dir):
11        os.makedirs(tensorboard_dir)
12    if not os.path.exists(save_dir):
13        os.makedirs(save_dir)
14    save_path = os.path.join(save_dir, 'best_validation')
15
16
17    tf.summary.scalar('loss', model.loss)
18    merged_summary = tf.summary.merge_all()
19    writer = tf.summary.FileWriter(tensorboard_dir)
20    saver = tf.train.Saver()
21    session = tf.Session()
22    session.run(tf.global_variables_initializer())
23    writer.add_graph(session.graph)
24
25    content_train, label_train = sequence2id(para.train)

```

```

26     content_test, label_test = sequence2id(para.test)
27
28     for epoch in range(para.epochs):
29         print('Epoch:', epoch+1)
30         num_batchs = int((len(content_train) - 1) / para.batch_size) + 1 # 轮数
31         batch_train = batch_iter(content_train, label_train)
32         for x_batch, y_batch in batch_train:
33             x_batch, seq_leng_x = process(x_batch)
34             y_batch, seq_leng_y = process(y_batch)
35             feed_dict = model.feed_data(x_batch, y_batch, seq_leng_x, para.
36             keep_pro)
37             _, global_step, loss, tain_summary = session.run([model.
38             optimizer, model.global_step, model.loss, merged_summary],
39             feed_dict=feed_dict)
40             if global_step % 100 == 0:
41                 test_loss = model.test(session, content_test, label_test)
42                 print('global_step:', global_step, 'train_loss:', loss, 'test_loss:', test_loss)
43             if global_step % (2*num_batchs) == 0:# 两个epoch保存一个
44                 print('Saving Model...')
45                 saver.save(session, save_path=save_path, global_step=
46                 global_step)
47                 para.learning_rate *= para.lr
48 if __name__ == '__main__':
49     para = para
50     model = biLstm_crf()
51     train()

```

A.0.4 predict.py

```

1 # -*- coding:utf-8 -*-
2 import pickle
3 from data_process import convert_word_seq, process, batch_iter, sequence2id,
4     read_file
5 from parameters import Parameters as para
6 from biLSTM_CRF import biLstm_crf
7 import tensorflow as tf
8 import numpy as np
9
10 def cutting(sentence, label_line):
11     word_cut = ''
12     wordlist = convert_word_seq(sentence)
13     # TypeDict = {'B': 0, 'M': 1, 'E': 2, 'S': 3}
14     for i in range(len(label_line)):

```

```

15     if label_line[i] == 2:
16         word_cut += wordlist[i]
17         word_cut += ' '
18     elif label_line[i] == 3:
19         word_cut += ' '
20         word_cut += wordlist[i]
21         word_cut += ' '
22     else:
23         word_cut += wordlist[i]
24
25
26
27 def val():
28     label = []
29     session = tf.Session()
30     session.run(tf.global_variables_initializer())
31     latest_save_path = tf.train.latest_checkpoint('./checkpoints')
32     saver = tf.train.Saver()
33     saver.restore(sess=session, save_path=latest_save_path)
34
35     content, _ = sequence2id(para.eva)
36     pre_label = model.predict(session, content)
37     label.extend(pre_label)
38
39     return label
40
41
42 if __name__ == '__main__':
43     para = para
44
45     model = biLstm_crf()
46
47     label = val()
48     with open(para.eva, 'r', encoding='utf-8-sig') as f:
49         sentences = [line.strip('\n') for line in f]
50
51     for i in range(len(sentences)):
52         sentence_cut = cutting(sentences[i], label[i])
53         print(sentences[i])
54         print(sentence_cut)

```

A.0.5 likelihood.py

```

1 import numpy as np
2 from tensorflow.python.framework import dtypes
3 from tensorflow.python.ops import array_ops
4 from tensorflow.python.ops import math_ops
5 from tensorflow.python.ops import rnn
6 from tensorflow.python.ops import rnn_cell
7 from tensorflow.python.ops import variable_scope as vs

```

```

8
9 __all__ = [
10     "crf_sequence_score", "crf_log_norm", "crf_log_likelihood",
11     "crf_unary_score", "crf_binary_score", "CrfForwardRnnCell"
12 ]
13
14 def _lengths_to_masks(lengths, max_length):
15     tiled_ranges = array_ops.tile(
16         array_ops.expand_dims(math_ops.range(max_length), 0),
17         [array_ops.shape(lengths)[0], 1])
18     lengths = array_ops.expand_dims(lengths, 1)
19     masks = math_ops.to_float(
20         math_ops.to_int64(tiled_ranges) < math_ops.to_int64(lengths))
21     return masks
22
23
24 def crf_sequence_score(inputs, tag_indices, sequence_lengths,
25                        transition_params):
26     unary_scores = crf_unary_score(tag_indices, sequence_lengths, inputs)
27     binary_scores = crf_binary_score(tag_indices, sequence_lengths,
28                                     transition_params)
29     sequence_scores = unary_scores + binary_scores
30     return sequence_scores
31
32
33 def crf_log_norm(inputs, sequence_lengths, transition_params):
34     first_input = array_ops.slice(inputs, [0, 0, 0], [-1, 1, -1])
35     first_input = array_ops.squeeze(first_input, [1])
36     rest_of_input = array_ops.slice(inputs, [0, 1, 0], [-1, -1, -1])
37
38     forward_cell = CrfForwardRnnCell(transition_params)
39     _, alphas = rnn.dynamic_rnn(
40         cell=forward_cell,
41         inputs=rest_of_input,
42         sequence_length=sequence_lengths - 1,
43         initial_state=first_input,
44         dtype=dtypes.float32)
45     log_norm = math_ops.reduce_logsumexp(alphas, [1])
46     return log_norm
47
48
49 def crf_log_likelihood(inputs,
50                       tag_indices,
51                       sequence_lengths,
52                       transition_params=None):
53     # log_likelihood是对数似然函数, transition_params是转移概率矩阵
54     # crf_log_likelihood{inputs:[batch_size,max_seq_length,num_tags
55     ],
56     # tag_indices:[batchsize,max_seq_length],
57     # sequence_lengths:[real_seq_length]
58     # transition_params: A [num_tags, num_tags] transition matrix

```

```

58         # log_likelihood: A scalar containing the log-likelihood of the
59         # given sequence of tag indices.
60         num_tags = inputs.get_shape()[2].value
61
62     if transition_params is None:
63         transition_params = vs.get_variable("transitions", [num_tags, num_tags])
64
65     sequence_scores = crf_sequence_score(inputs, tag_indices, sequence_lengths
66                                         ,
67                                         transition_params)
68
69     log_norm = crf_log_norm(inputs, sequence_lengths, transition_params)
70
71     log_likelihood = sequence_scores - log_norm
72     return log_likelihood, transition_params
73
74
75
76
77 def crf_unary_score(tag_indices, sequence_lengths, inputs):
78     batch_size = array_ops.shape(inputs)[0]
79     max_seq_len = array_ops.shape(inputs)[1]
80     num_tags = array_ops.shape(inputs)[2]
81
82     flattened_inputs = array_ops.reshape(inputs, [-1])
83
84     offsets = array_ops.expand_dims(
85         math_ops.range(batch_size) * max_seq_len * num_tags, 1)
86     offsets += array_ops.expand_dims(math_ops.range(max_seq_len) * num_tags,
87                                     0)
88     flattened_tag_indices = array_ops.reshape(offsets + tag_indices, [-1])
89
90     unary_scores = array_ops.reshape(
91         array_ops.gather(flattened_inputs, flattened_tag_indices),
92         [batch_size, max_seq_len])
93
94     masks = _lengths_to_masks(sequence_lengths, array_ops.shape(tag_indices)
95                               [1])
96
97     unary_scores = math_ops.reduce_sum(unary_scores * masks, 1)
98     return unary_scores
99
100
101
102 def crf_binary_score(tag_indices, sequence_lengths, transition_params):
103     num_tags = transition_params.get_shape()[0]
104     num_transitions = array_ops.shape(tag_indices)[1] - 1
105
106     start_tag_indices = array_ops.slice(tag_indices, [0, 0],
107                                         [-1, num_transitions])
108     end_tag_indices = array_ops.slice(tag_indices, [0, 1], [-1,
109                                         num_transitions])
110
111     flattened_transition_indices = start_tag_indices * num_tags +
112     end_tag_indices

```

```

103 flattened_transition_params = array_ops.reshape(transition_params, [-1])
104
105 binary_scores = array_ops.gather(flattened_transition_params,
106                                     flattened_transition_indices)
107
108 masks = _lengths_to_masks(sequence_lengths, array_ops.shape(tag_indices)
109                           [1])
110 truncated_masks = array_ops.slice(masks, [0, 1], [-1, -1])
111 binary_scores = math_ops.reduce_sum(binary_scores * truncated_masks, 1)
112 return binary_scores
113
114 class CrfForwardRnnCell(rnn_cell.RNNCell):
115     def __init__(self, transition_params):
116         self._transition_params = array_ops.expand_dims(transition_params, 0)
117         self._num_tags = transition_params.get_shape()[0].value
118
119     @property
120     def state_size(self):
121         return self._num_tags
122
123     @property
124     def output_size(self):
125         return self._num_tags
126
127     def __call__(self, inputs, state, scope=None):
128         state = array_ops.expand_dims(state, 2)
129         transition_scores = state + self._transition_params
130         new_alphas = inputs + math_ops.reduce_logsumexp(transition_scores, [1])
131         return new_alphas, new_alphas

```

A.0.6 parameters.py

```

1 # -*- coding:utf-8 -*-
2 class Parameters(object):
3     embedding_size = 100
4     vocab_size = 4000
5     batch_size = 64
6     hidden_dim = 128
7
8     learning_rate = 0.006
9     clip = 5.0
10    lr = 0.8
11
12    keep_pro = 0.5
13    num_tags = 4
14    epochs = 8
15
16    train = './data/WordSeg.txt'
17    test = './data/test.txt'
18    eva = './data/eva.txt'

```