SHANGHAI UNIVERSITY

2019-2020学年夏季学期

《智能系统联合大作业(0869A006)》课程报告

学 号： 18120255

学 生： 姚施越

指导教师： 王欣芝

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平时学习成绩  20% | 项目成绩  20% | 比赛成绩  40% | 报告成绩  20% | 总成绩 |
|  |  |  |  |  |

计算机工程与科学学院

2020年 7 月 5 日

《智能系统联合大作业(0869A006)》课程报告

**一、模型设计**

**（一）试题说明**

**1、任务描述**

对于每个用户问题“QID”，有对应文本形式的文本集合D =“Brand”，“Collection”，“Problem”，“Conversation”，要求阅读理解系统自动对D进行分析，输出相应的报告文本“Report”，其中包含摘要与推理。目标是“Report”可以正确、完整、简洁、清晰、连贯地对D中的信息作归纳总结与推理。

**2、数据说明**

汽车大师提供了11万条来自于真实场景的问答数据；每条数据都包含一次问答服务的整体体验—问题描述、追问追答、结构化。

**（二）项目简介**

本项目使用AiStudio平台，基于PaddlePaddle框架，完整实现了问答摘要的具体流程，包括预处理、模型建立、训练、预测等等。对数据进行预处理之后，通过实现一个Seq2Seq的网络结构，根据Problem和Conversation，以生成式的方法，生成Report。由于使用到了python2及paddle.v2的相关api，在比赛版本下无法运行，故自建项目以实现。

项目环境：Python2.7

预加载项目框架：PaddlePaddle1.5.1（后在项目内安装v2版本）

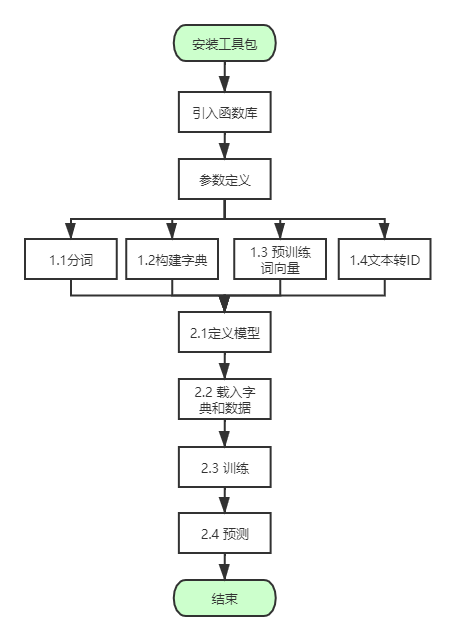


图1-1 项目流程图

**（三）预处理**

**1、分词**

使用jieba中文分词组件，进行分词分句，并把结果以csv文件存储，以便进行后续操作。

**2、统计词典**

统计经过分词的训练集中各词词频，将Problem, Conversation, Report三列数据作为构建语料，词频大于等于五的所有词构成词典，同时词典中包含start\_token，end\_token，unk\_token这三个特殊词。最后，保存为txt文件。

**3、预训练词向量**

使用gensim库中的Word2Vec模型，将数据集中的所有Problem，Conversation和Report集合起来训练skip-gram 词向量，以txt文件保存，使后续训练能更快地收敛。

**4、数据集向量化**

将分词后的数据集向量化，并把最终开发集中的部分数据加入训练集中，增大训练集规模。

**（四）主体模型**

**1、概念引入**

这部分代码参考了PaddlePaddle机器翻译模型的官方文档，NMT（神经机器翻译）模型使用神经网络将源语言转换为目标语言，即端到端神经机器翻译。

**（1）神经机器翻译**

神经机器翻译由Nal Kalchbrenner和Phil Blunsom于2013年提出。该模型使用卷积神经网络将给定的源文本编码为连续向量，然后使用循环神经网络作为解码器将状态向量转换为目标语言。NMT是一种使用深度学习神经网络在自然语言之间进行映射的方法，它使用连接编码器和解码器的状态向量来描述语义等价。

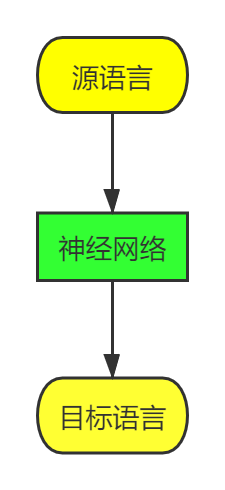


图1-2 NMT模型

**（2）Encoder-Decoder结构**

Encoder-Decoder结构能够将具有任意长度的源序列转换为具有任意长度的另一个目标序列。在编码阶段，它将整个源序列编码为一个向量。并且在解码阶段，它通过最大化预测的序列概率来解码整个目标序列。编码和解码过程通常由RNN实现。

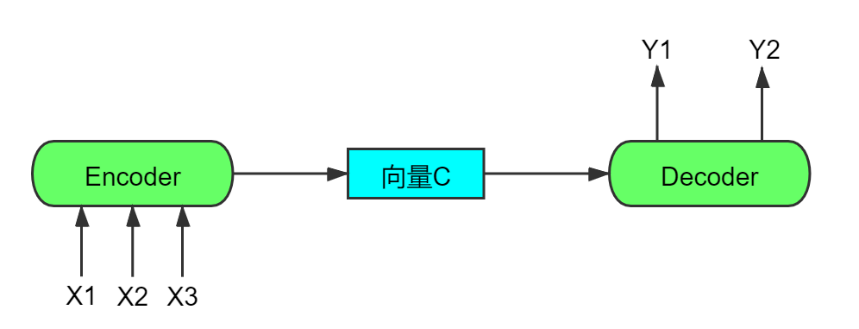


图1-3 Encoder-Decoder结构

说明：

（1）不论输入和输出的长度是什么，中间向量C的长度都是固定的

（2）根据不同的任务可以选择不同的编码器和解码器（可以是一个RNN，通常是LSTM或者GRU）

**（3）门控循环单元（GRU）**

我们上文提到编码器和解码器常选用RNN（循环神经网络），GRU（Gate Recurrent Unit）便是其中的一种。和LSTM（Long-Short Term Memory）一样，是为了解决长期记忆和反向传播中的梯度等问题而提出来的。

相比LSTM，使用GRU能够达到相当的效果，并且相比之下更容易进行训练，能够很大程度上提高训练效率，因此很多时候会更倾向于使用GRU。

有一个当前的输入，和上一个节点传递下来的隐状态（hidden state），这个隐状态包含了之前节点的相关信息。结合和，GRU会得到当前隐藏节点的输出和传递给下一个节点的隐状态。其输入输出结构如下图1-4所示。

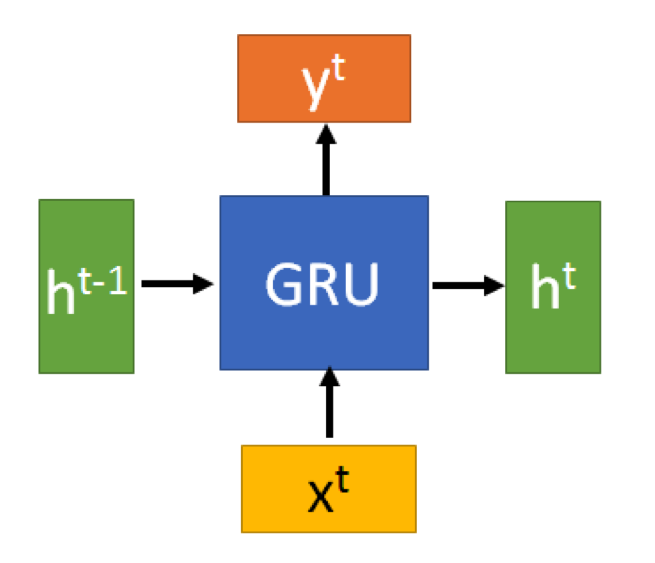


图1-4 GRU的输入输出结构

下图1-5展示了GRU的内部结构。其中，⊙表示Hadamard Product，也就是操作矩阵中对应的元素相乘，因此要求两个相乘矩阵是同型的。⊕则代表进行矩阵加法操作。

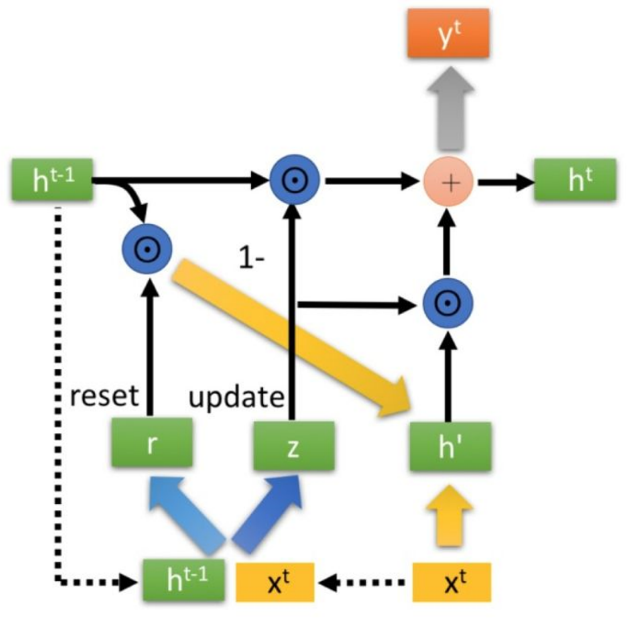


图1-5 GRU的内部结构

更新的表达式：

结合图例和表达式，可以发现，对于传递进来的维度信息，我们会进行选择性遗忘，则遗忘了多少权重，我们就会使用包含当前输入的中所对应的权重进行弥补，以此来保持一种“恒定”状态。

**（4）Seq2Seq框架**

Seq2Seq（全称Sequence to Sequence）不特指具体方法，只要满足输入序列、输出序列的目的，都可以统称为 Seq2Seq 模型。而 Seq2Seq 使用的具体方法基本都属于上文提到的Encoder-Decoder 模型的范畴。

注意机制（Attention Mechanism），作为Seq2Seq中的重要组成部分，注意机制最早由Bahdanau等人于2014年提出，该机制存在的目的是为了解决RNN中只支持固定长度输入的瓶颈。在该机制环境下，Seq2Seq中的编码器被替换为一个双向循环网络（bidirectional RNN）。

Attention模型的特点是Encoder不再将整个输入序列编码为固定长度，而是编码成一个向量的序列。引入了Attention的Encoder-Decoder模型如下图：

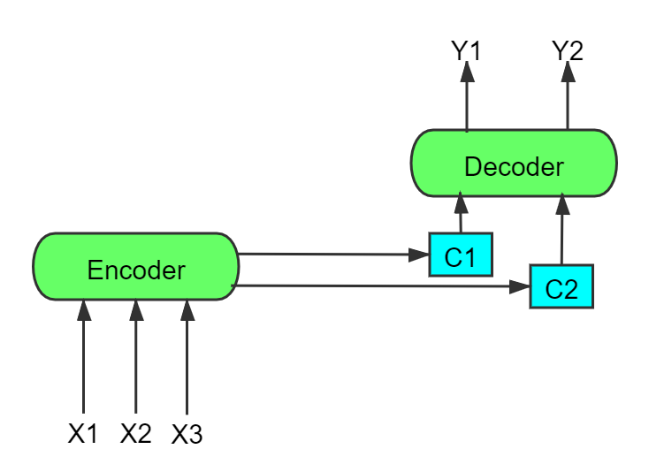
****

图1-6 引入Attention的Encoder-Decoder结构

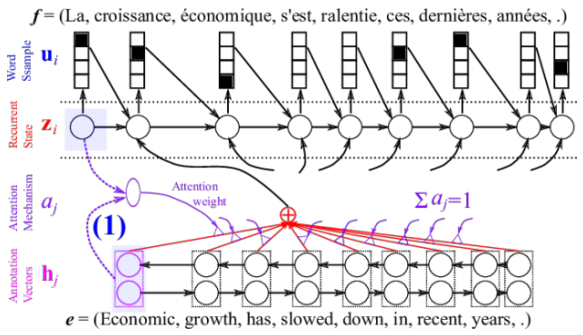


图1-7 完整的神经机器翻译模型（来源：http://statmt.org/mtma16/uploads/mtma16-neural.pdf）

至此，实现了引入双向GRU并使用Attention机制的Encoder-Decoder结构，完整的神经机器翻译模型便搭建完成了，如图1-7所示。

**2、模型定义**

参考PaddlePaddle机器翻译模型的官方文档，利用paddle.v2版本，实现了完整的基于Attention的Seq2Seq模型。在网络模型中尽可能共享参数，例如problem和conversation使用同一个encoder，embedding矩阵则是为encoder和decoder共用。具体的代码不在此展示，可在源代码部分查看。

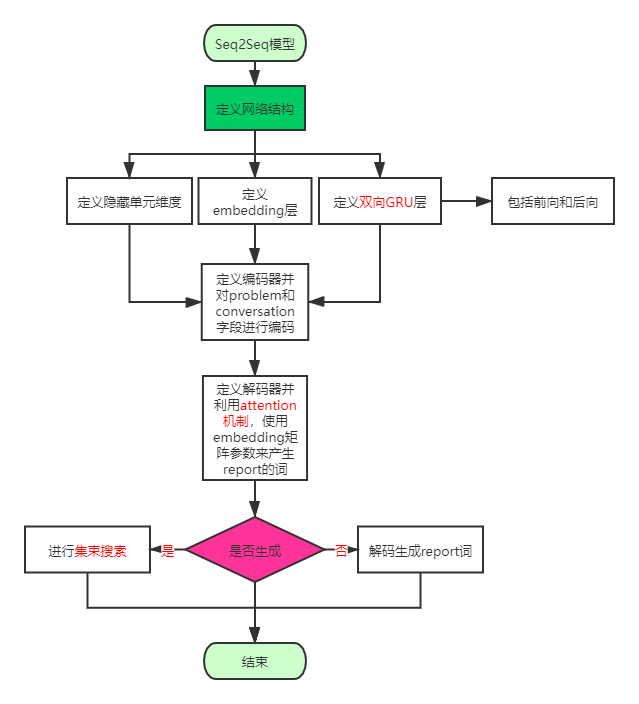
****

图1-7 Seq2Seq模型实现流程

**3、训练和预测函数定义**

这一部分主要由主函数和预测函数构成，具体可在源代码部分查看。

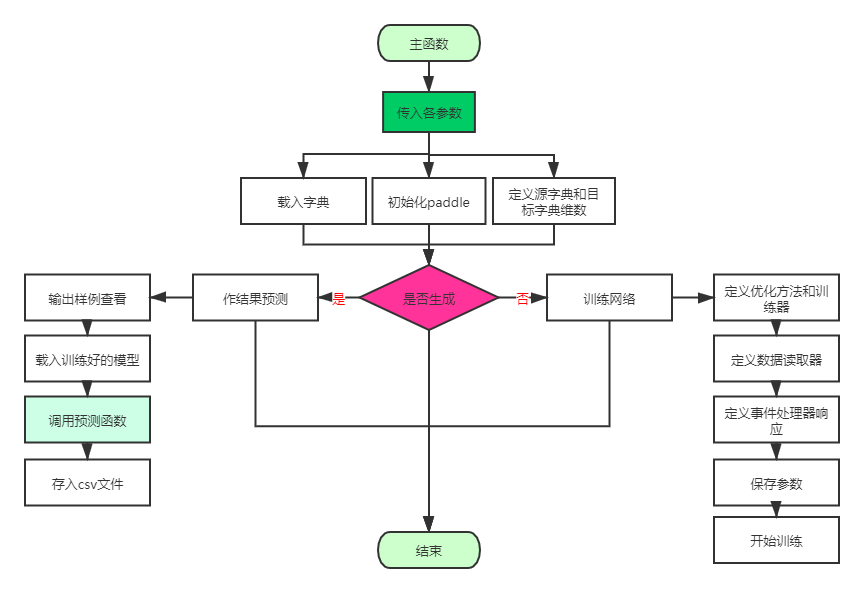


图1-8 训练和预测部分实现流程

**4、载入字典和数据**

载入字典和数据，并定义数据读取器。

**5、进行训练**

调用此前定义的主函数进行训练，通过更改传入的参数来实现调参过程。

训练包括两批次，第一批次是初始训练开始，默认学习率是5e-4；第二批次则载入第一批次模型继续训练，在这里可以手动调节学习率。具体的参数设计和训练过程见二、实验结果。

**6、测试集预测**

使用两批次训练结束的模型，对测试集的problem字段进行训练，合理预测答案，最后生成csv文件用于上传。

**二、实验结果**

**包括参数设计，训练过程，统计计算结果、示例（好的与差的）**

**（一）参数设计**

经历了漫长的调参过程，并通过提交比赛查验分数，我基本得出了两批次训练最优的参数。将其中的一些记录下来，如下表1所示。

表1 训练参数设计意义及最优数值

|  |  |  |
| --- | --- | --- |
| 名称 | 意义 | 最优数值 |
| lr（第一批次） | 第一批次训练过程的学习率 | 5e-4 |
| n\_epoch（第一批次） | 第一批次训练过程的轮数 | 15左右 |
| batch\_size（第一批次） | 第一批次训练过程的单次训练样本 | 32 |
| lr（第二批次） | 第二批次训练过程的学习率 | 1e-5 |
| n\_epoch（第二批次） | 第二批次训练过程的轮数 | 大于30 |
| batch\_size（第二批次） | 第二批次训练过程的单次训练样本 | 16 |

其中，学习率按照经验选定，第一批次学习率较高，使得loss下降较快；第二批次载入现有模型训练，设置一个非常小的初始学习率并手动调节学习率。至于训练轮数，通过多次训练和提交的分数来进行选定，经实验发现第一批次在达到15轮之后对分数的贡献不大；第二批次则分别以3轮，8轮，15轮进行实验，分数也由50分左右提升至87分，最终达到92分。在这过程中，调整了batch\_size的大小，由于使用到的是GPU环境，算力是足够的，因此将其初始值设置地较小为32，之后尝试减半即16，分数也是得到了提升。由于运行时长的限制，轮数没有设得很大，最优分数的提交参数为上表1中的各值。

**（二）训练过程**

训练过程将主要以图例形式呈现，预处理部分不在此展示，以训练模型和提交情况为主，记录了完整的调参过程。

**1、训练范例1**

表2 训练范例1参数及数值

|  |  |
| --- | --- |
| 名称 | 数值 |
| lr（第一批次） | 5e-4 |
| n\_epoch（第一批次） | 7 |
| batch\_size（第一批次） | 24 |
| lr（第二批次） | 1e-5 |
| n\_epoch（第二批次） | 2 |
| batch\_size（第二批次） | 32 |



图2-1 训练范例1第一次训练



图2-2 训练范例1第二次训练



图2-3 训练范例1预测结果



图2-4 训练范例1提交分数

**2、训练范例2**

表3 训练范例2参数及数值

|  |  |
| --- | --- |
| 名称 | 数值 |
| lr（第一批次） | 5e-4 |
| n\_epoch（第一批次） | 15 |
| batch\_size（第一批次） | 32 |
| lr（第二批次） | 1e-5 |
| n\_epoch（第二批次） | 8 |
| batch\_size（第二批次） | 16 |

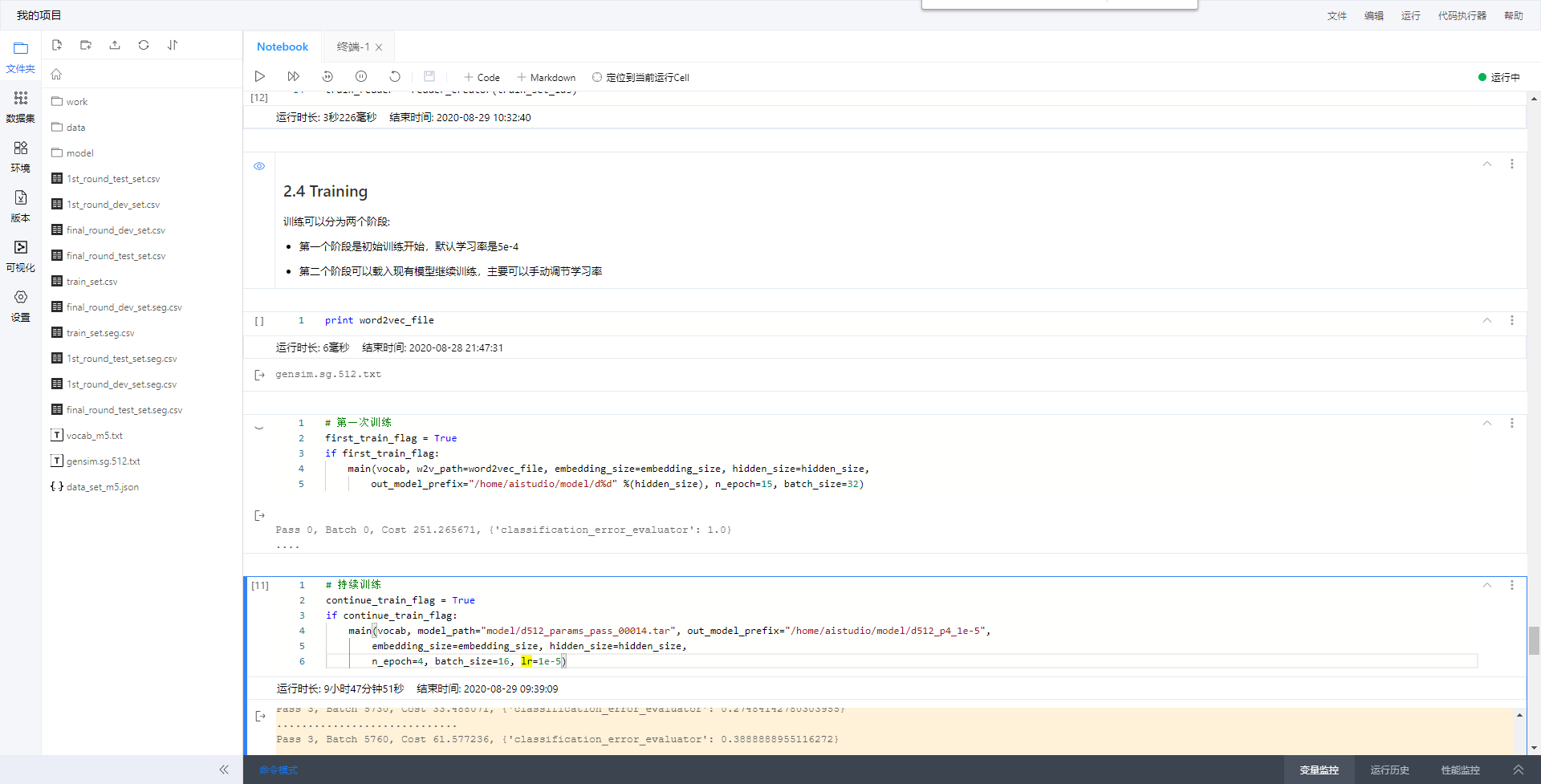


图2-5 训练范例2第一次训练

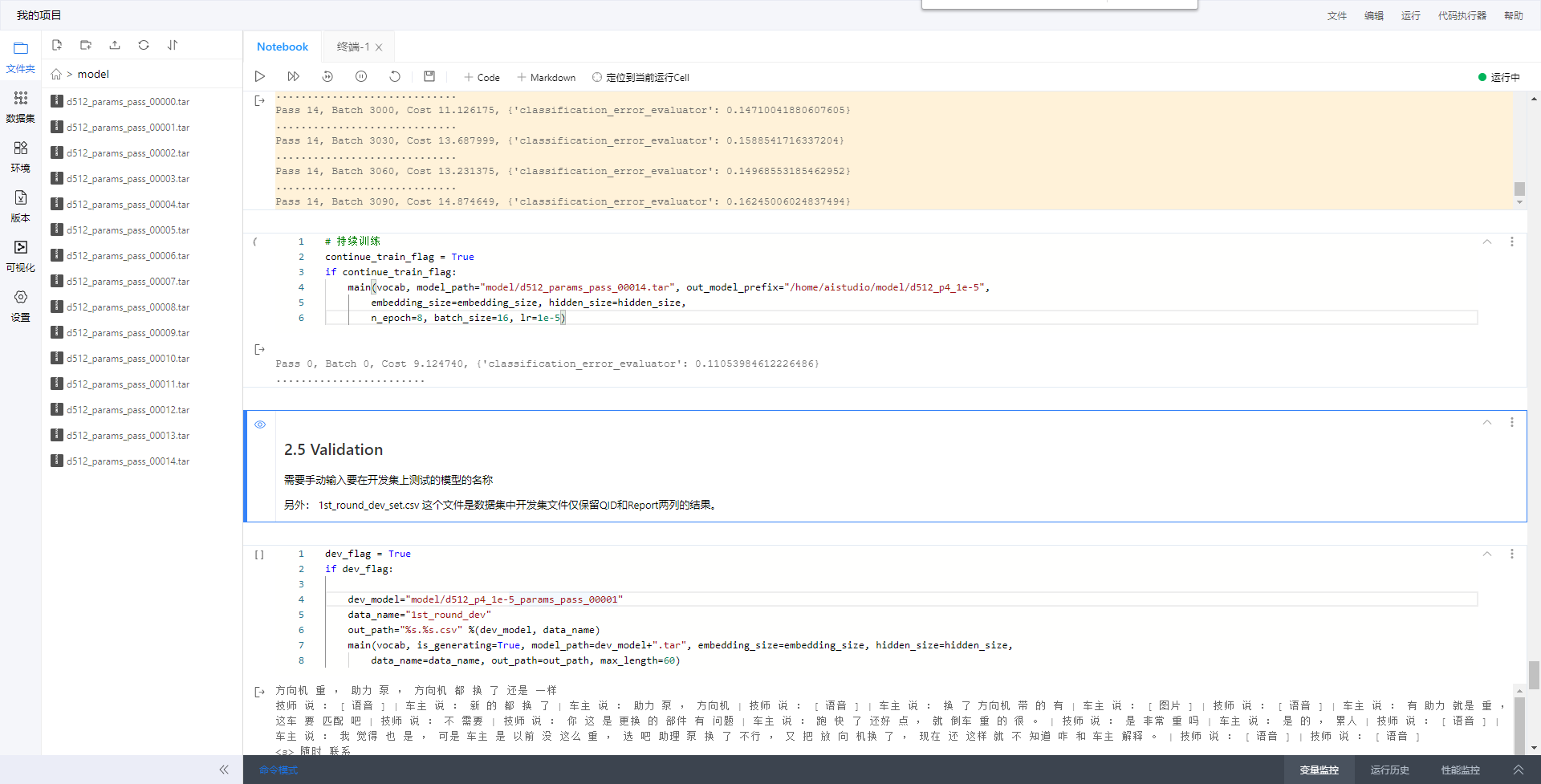


图2-6 训练范例2第二次训练

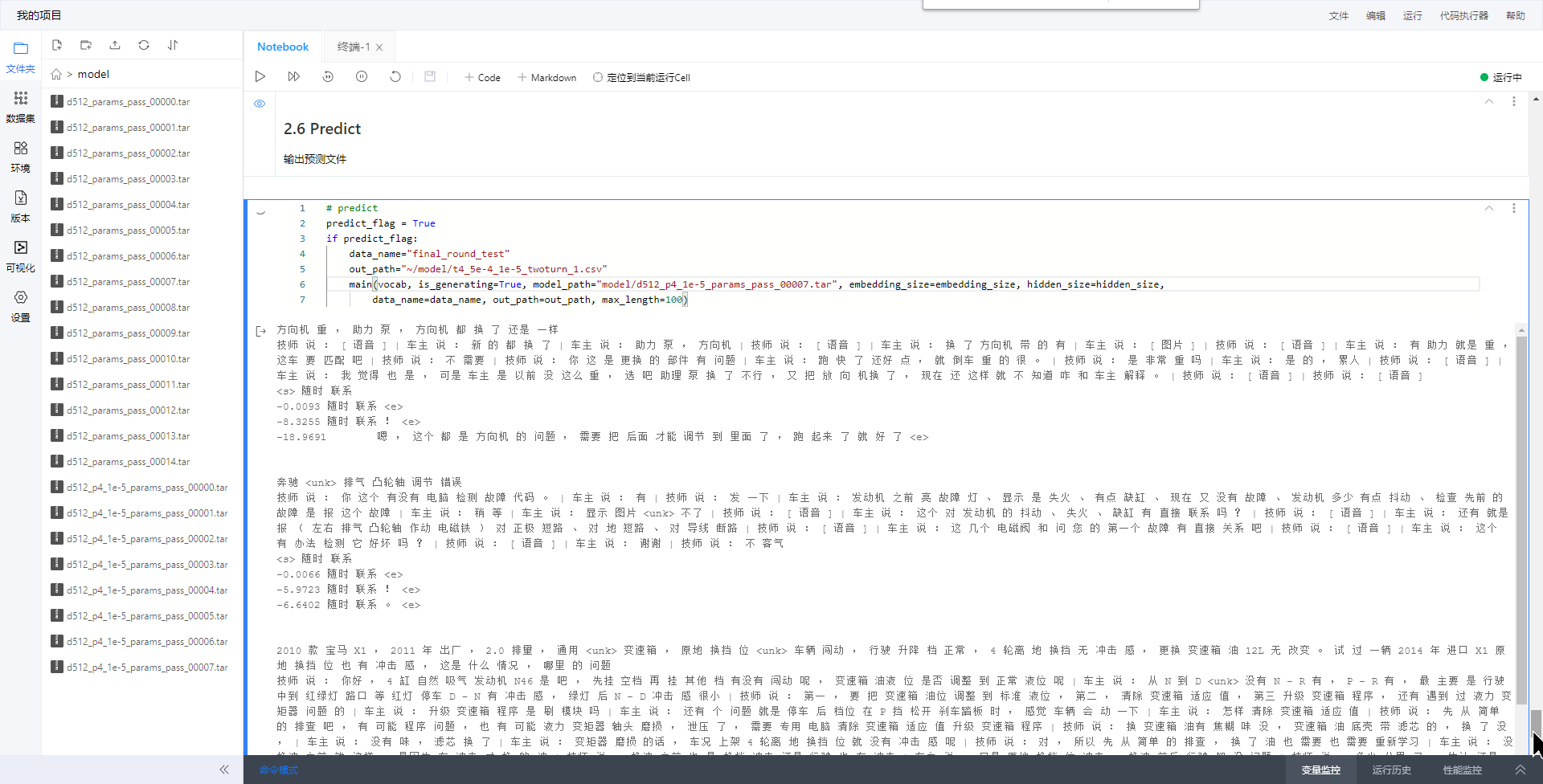


图2-7 训练范例2两批次运行之后保存的模型

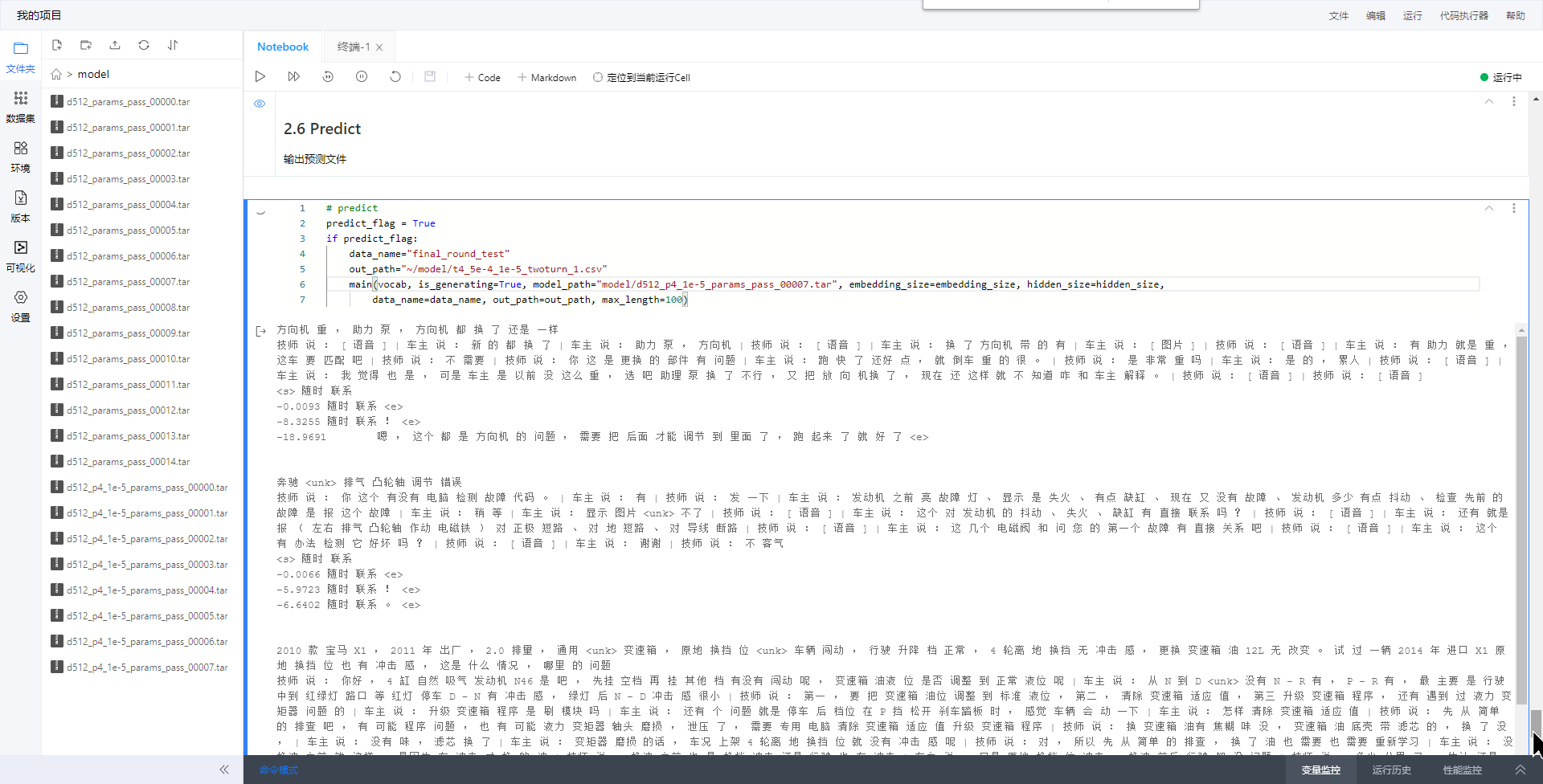


图2-8 训练范例2预测结果



图2-9 训练范例2提交分数

之后，对学习率及batch\_size先不加以修改，只是增加训练轮数， 查看其效果。



图2-10 训练范例2修改为15轮后的分数



图2-11 训练范例2修改为30轮后的分数

**3、训练范例3**

表4 训练范例3参数及数值

|  |  |
| --- | --- |
| 名称 | 数值 |
| lr（第一批次） | 5e-4 |
| n\_epoch（第一批次） | 15 |
| batch\_size（第一批次） | 32 |
| lr（第二批次） | 1e-5 |
| n\_epoch（第二批次） | 30 |
| batch\_size（第二批次） | 32 |

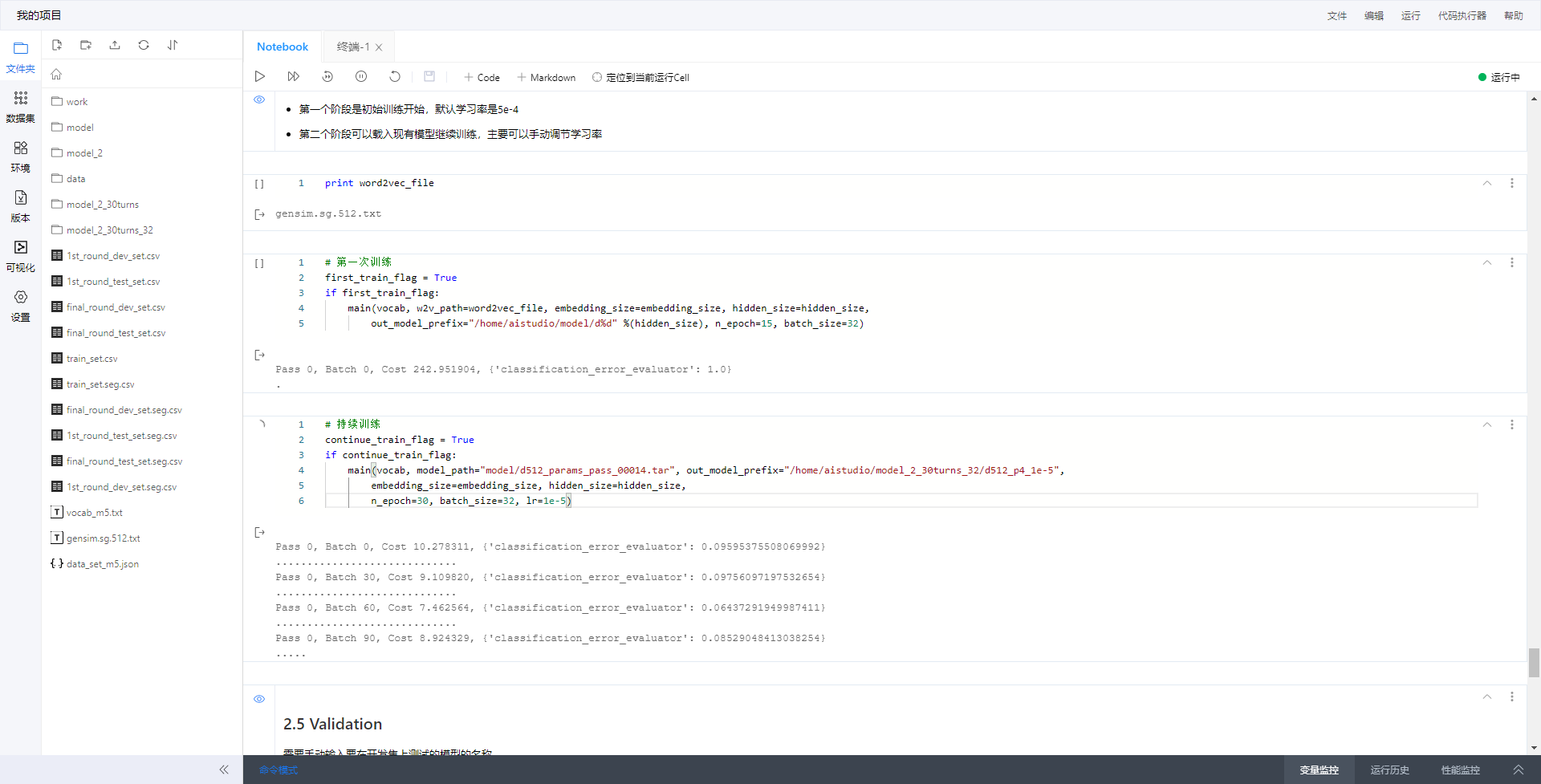


图2-12 训练范例3第一次训练

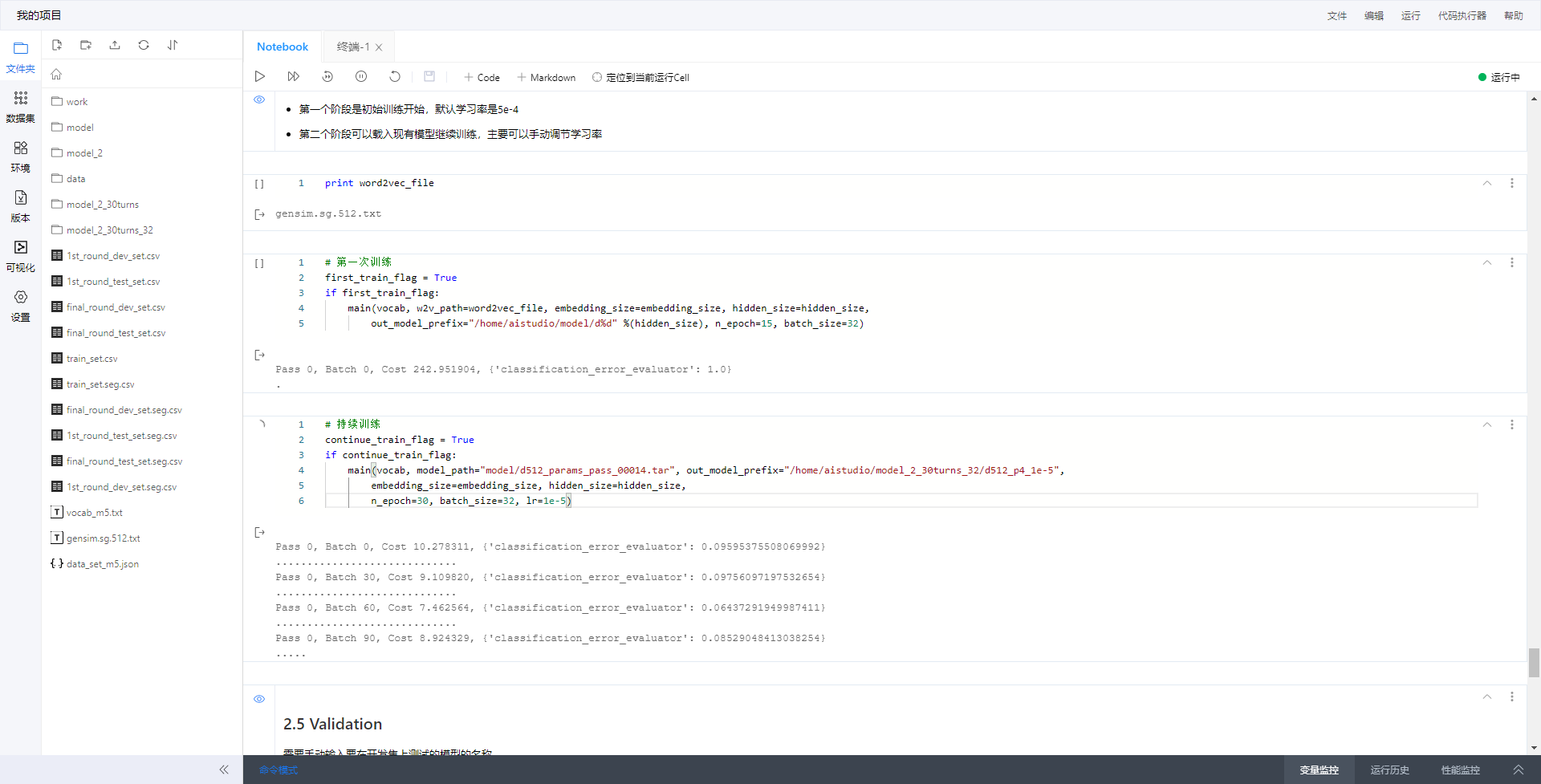


图2-13 训练范例3第二次训练

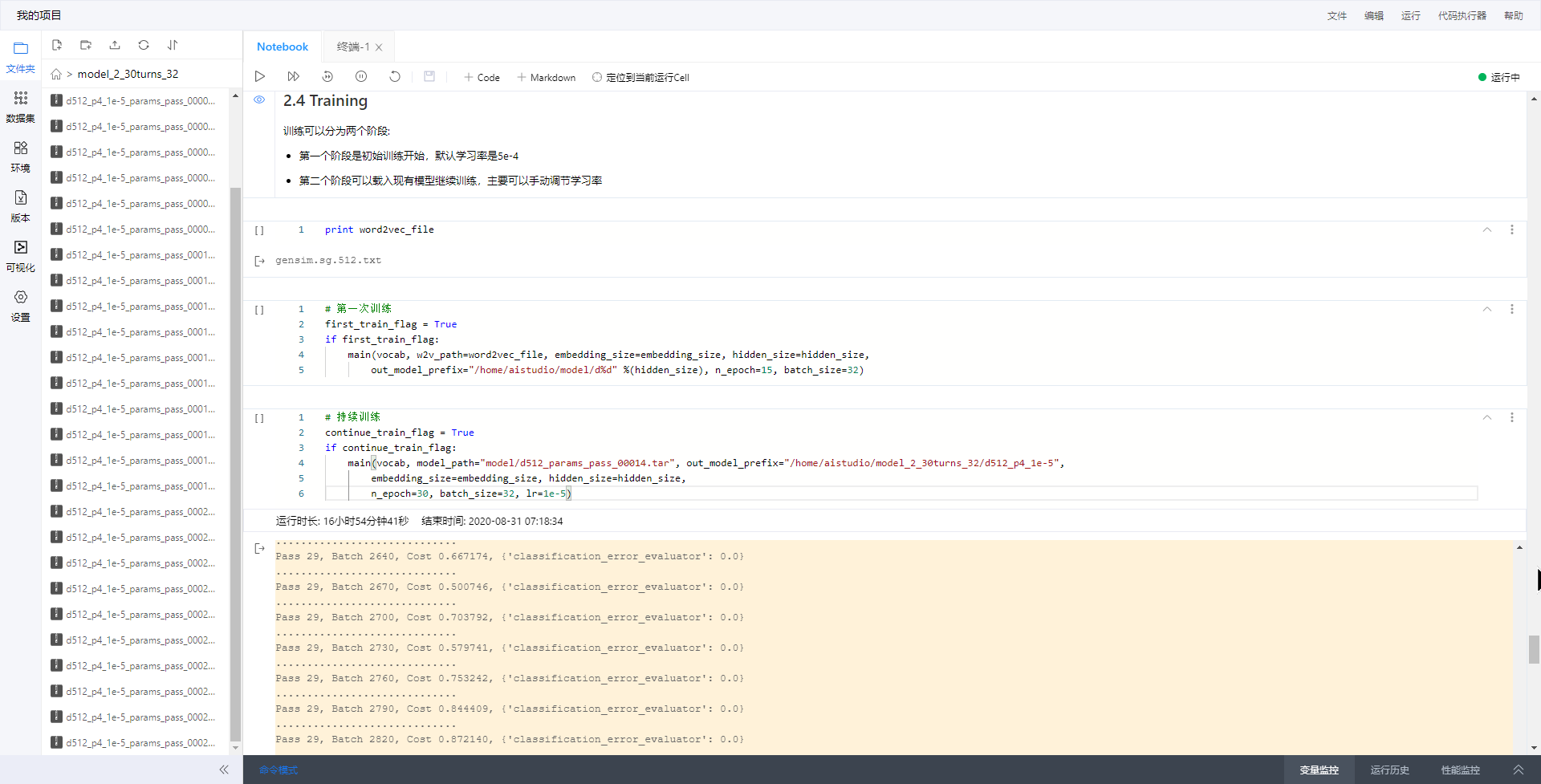


图2-14 训练范例3两批次运行之后保存的模型

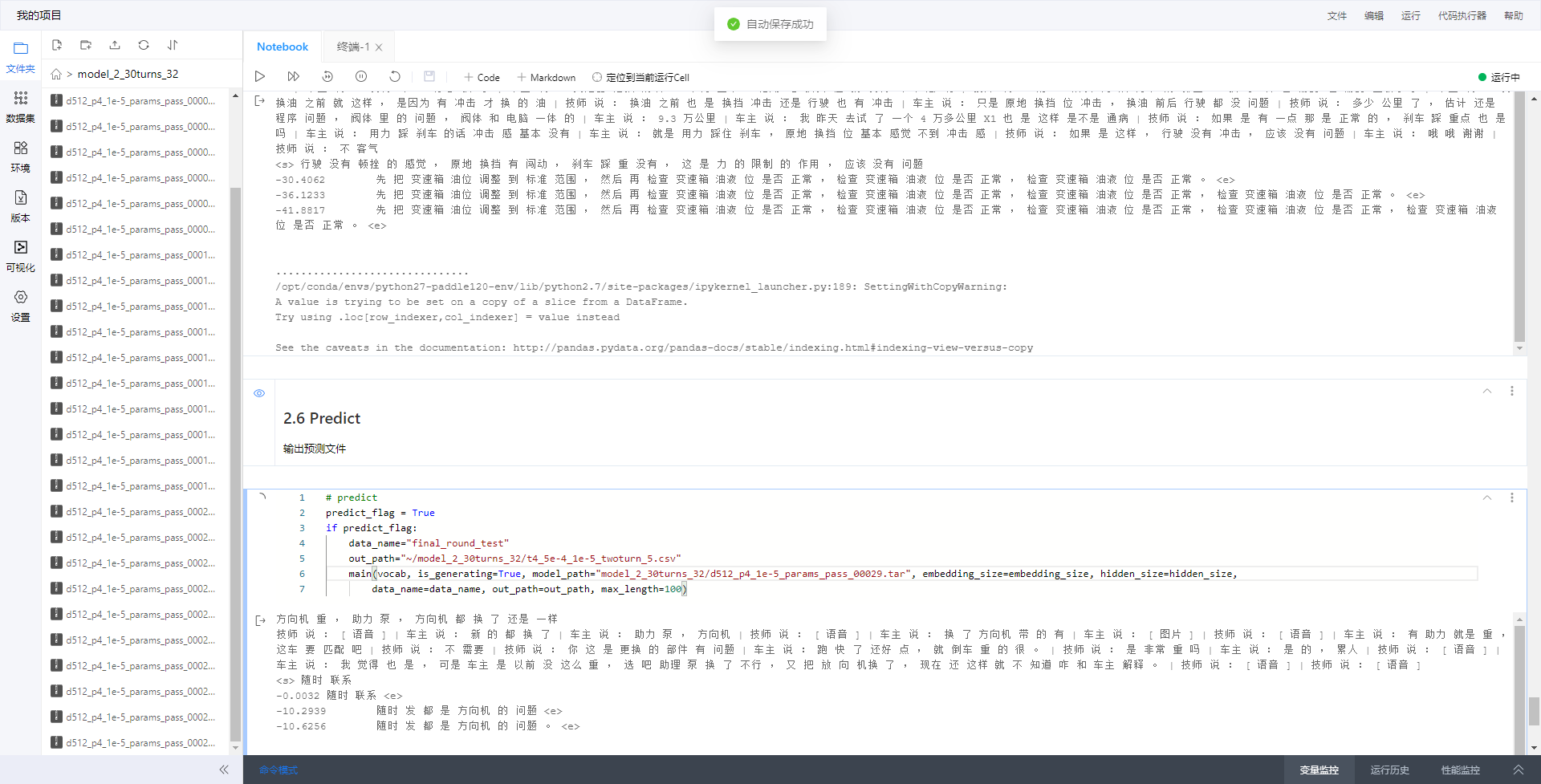


图2-15 训练范例3预测结果



图2-16 训练范例3提交分数

增大batch\_size之后，运行相同数量的轮次，分数略有下降。

**（三）示例**

**1、示例1**

示例部分将展示不同参数下预测文件的内容及得分情况。使用Excel打开预测结果的csv文件可能存在乱码情况，使用Notepad++查看。

表5 示例1参数及数值

|  |  |
| --- | --- |
| 名称 | 数值 |
| lr（第一批次） | 5e-4 |
| n\_epoch（第一批次） | 7 |
| batch\_size（第一批次） | 24 |
| lr（第二批次） | 1e-5 |
| n\_epoch（第二批次） | 2 |
| batch\_size（第二批次） | 32 |
| 提交得分 | 50.1071 |



图2-17 示例1文件内容

可以看到，每一个问题的回复基本都在一到两句，少量出现两句重复内容。

**2、示例2**

表6 示例2参数及数值

|  |  |
| --- | --- |
| 名称 | 数值 |
| lr（第一批次） | 2e-5 |
| n\_epoch（第一批次） | 3 |
| batch\_size（第一批次） | 32 |
| lr（第二批次） | 未使用 |
| n\_epoch（第二批次） | 未使用 |
| batch\_size（第二批次） | 未使用 |
| 提交得分 | 26.6318 |

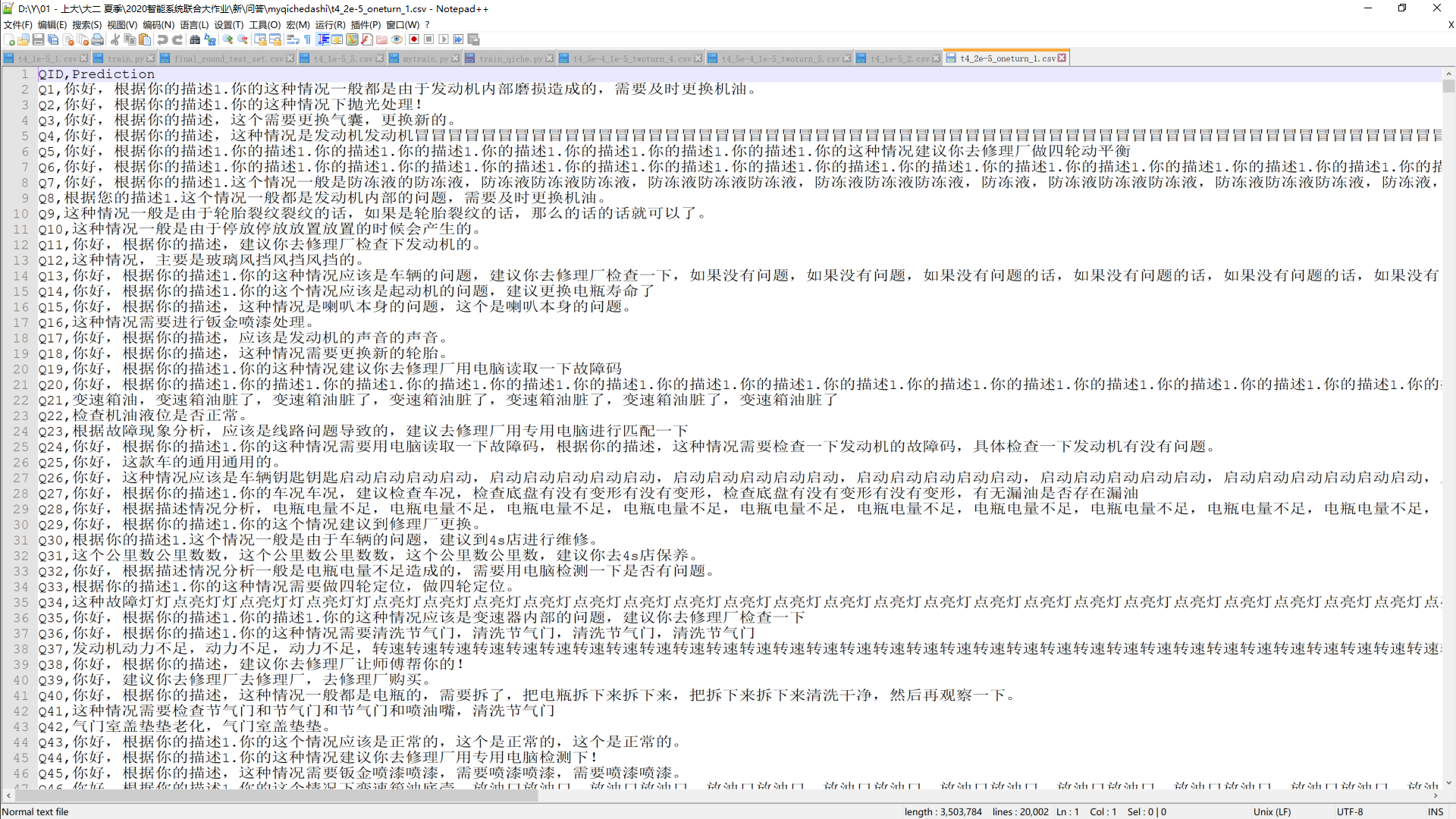


图2-18 示例2文件内容

可以看到，这个文件的内容效果就不太理想了，大量出现“根据你的描述”以及重复词的情况。

**3、示例3**

表7 示例3参数及数值

|  |  |
| --- | --- |
| 名称 | 数值 |
| lr（第一批次） | 5e-4 |
| n\_epoch（第一批次） | 15 |
| batch\_size（第一批次） | 32 |
| lr（第二批次） | 1e-5 |
| n\_epoch（第二批次） | 30 |
| batch\_size（第二批次） | 16 |
| 提交得分 | 93.4189 |

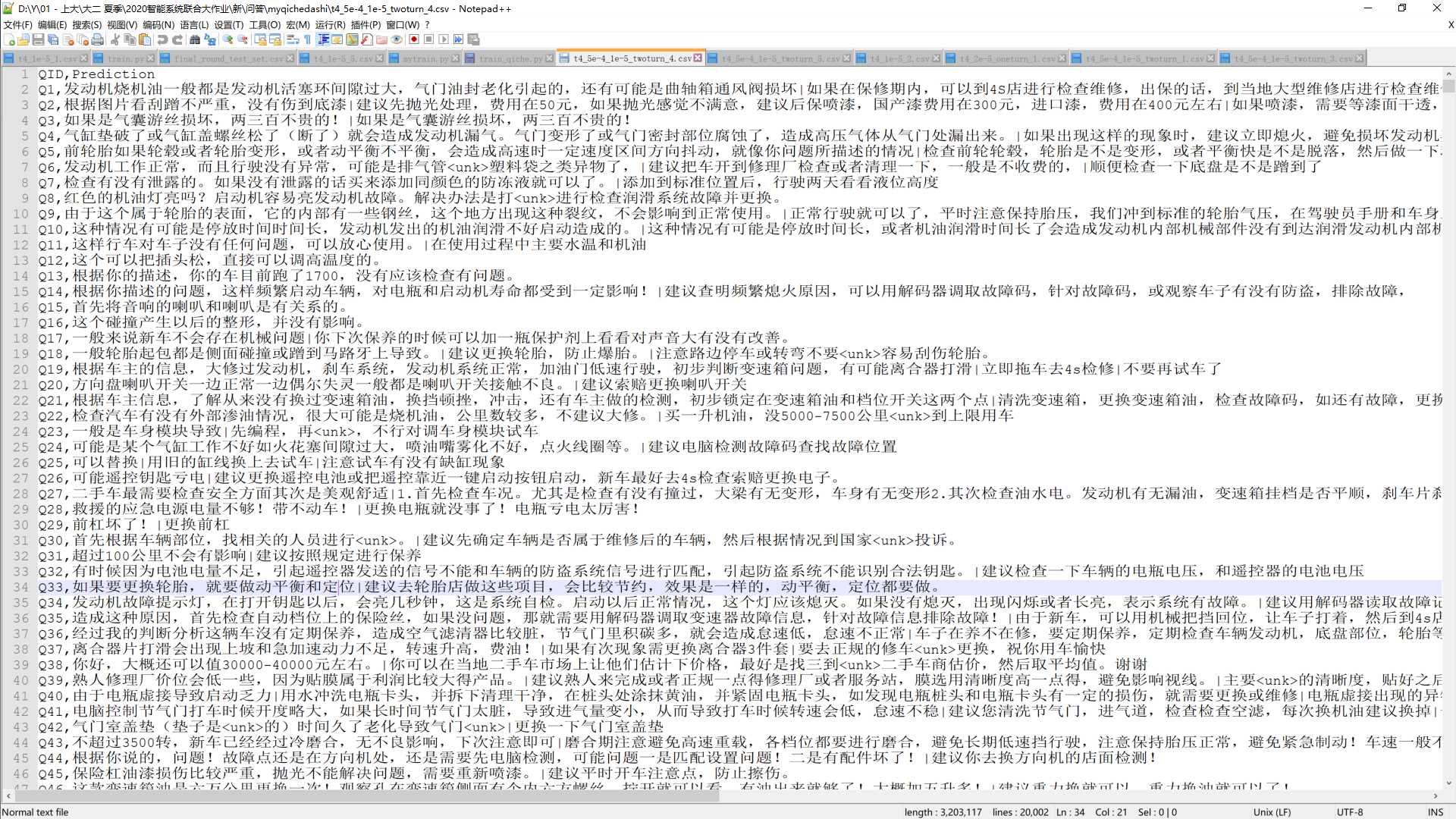


图2-19 示例3文件内容

可以看到，这个文件的内容效果就非常好了，语句都比较长，回答较详细，也出现了<unk>未知词，病句也比较少。

结合以上三个示例来看，可以说，提交分数与文件内容的效果包括语句长度、通顺程度以及我们不太能评估的相关性是息息相关的。

**三、收获与体会**

19-20学年夏季学期在疫情的影响之下，改为了线上进行，《智能系统联合大作业》课程随着这篇报告的撰写完毕结束了。回顾这几周包括暑假期间，对于本门课程的认识加深了不少，关于深度学习的相关内容，尤其是图像分类和自然语言处理也有所了解了。对此，我有几点感触，将在下文中展开叙述。

首先，关于如何分析问题。在这门课程中，以考试和比赛为首的项目占到了大比例，想要完整地实现，必不可少的就是个人分析问题的能力。就拿问答摘要与推理来说，是否进行数据预处理，选用何种模型，参数的选取和调整，都是需要我们自己去考虑的。尽管项目不同，但整体的思路是基本一致的，所以还是要从基本出发。在经过调研和不断尝试之后，最后确定了方案。

其次，关于如何学习人工智能。百度大脑平台为我们提供了很好的学习资源，使得我们能较快地上手。对于我所从事的项目，必定会有类似的其他项目，这时候他人的经验就显得很有参考价值了。通过论坛，书籍，博客等方式，收集到他人对于其实现方式的分享，对于自己如何考虑也能够有更好的认识，更能快速上手。对于不懂的问题不耻下问，再加以自己的思考，能够更快地获得解答。以上两点也正是我的做法，很好地帮到了我。

最后，说说自己的收获。这种考核方式对于我来说是不小的挑战，但也是通过这几周的时间，经历了摸索的过程，我对于如何搭建完整的人工智能项目有了一些实战经验，针对自然语言处理也有了一些浅显的认识。调参训练的过程，伴随着模型效果出众的喜悦，回想起来也是非常的难忘。截止这份报告提交时刻，客观分排在第二的位置。另外，对于如何撰写报告，我积累了一些经验，要尽可能做到言之有物，结合一些图表，完整清晰地展示自己所做的工作，并表明自己分析问题的思路和方法。

另外，由于本人的能力有限，在这份报告中，也可能会有不少纰漏和不严密的地方，敬请王老师批评指正。虽然这次对于某些知识的运用和衔接还不够熟练，但是我将在今后的学习中继续努力、不断完善。

最后的最后，感谢王老师、沈老师的辛勤付出，以及在我学习的过程中给予的帮助。

**四、项目部分源代码**

|  |
| --- |
| 1. #!/usr/bin/env python 2. # coding: utf-8 3. """ 4. Sun Aug 28 11:01:42 2020 5. @author: Badwolf\_Yao 6. """ 7. # In[ ]: 8. # 查看当前挂载的数据集目录, 该目录下的变更重启环境后会自动还原 9. # View dataset directory. This directory will be recovered automatically after resetting environment. 10. get\_ipython().system('ls /home/aistudio/data') 11. # In[ ]: 12. # 查看工作区文件, 该目录下的变更将会持久保存. 请及时清理不必要的文件, 避免加载过慢. 13. # View personal work directory. All changes under this directory will be kept even after reset. Please clean unnecessary files in time to speed up environment loading. 14. get\_ipython().system('ls /home/aistudio/work') 15. # In[1]: 16. # paddlepaddle版本降级，使用v2版本 17. get\_ipython().system('pip install jieba paddlepaddle-gpu==1.2.0.post97') 18. # In[2]: 19. get\_ipython().system('pip install gensim') 20. # In[3]: 21. ## 引入函数库 ## 22. import sys 23. import os 24. import copy 25. import pandas as pd 26. import numpy as np 27. import codecs 28. import json 29. import jieba 30. from gensim.models import Word2Vec 31. import paddle.v2 as paddle 32. import matplotlib.pyplot as plt 33. # In[ ]: 34. **print**(paddle.\_\_file\_\_) 35. # In[5]: 36. ## 参数定义 ## 37. start\_token = u"<s>" 38. end\_token = u"<e>" 39. unk\_token = u"<unk>" 40. start\_id = 0 41. end\_id = 1 42. unk\_id = 2 43. # 词典路径 44. min\_count = 5 45. vocab\_path = "vocab\_m%s.txt" %(min\_count) 46. # problem，conversation 和 report的最大长度 47. # 在训练中，超过长度的样本会被剔除 48. # 在测试中，超过长度的样本会被截断 49. max\_problem\_len = 100 50. max\_conversation\_len = 800 51. max\_report\_len = 100 52. # 词向量和隐层节点维度 53. embedding\_size=hidden\_size=512 54. # 词向量文件路径 55. word2vec\_file="gensim.sg.%d.txt" %(embedding\_size) 56. # 预处理后文件存储路径 57. data\_path = "data\_set\_m%s.json" %(min\_count) 58. # ## Part 1. Preprocessing 59. # Part 1 的代码主要包含分词、统计词典、预训练词向量和数据集转换四部分。 60. # 这四部分会产生一些中间结果，供后续程序使用，至少需要运行一次，之后可以通过把相关的flag设置为False跳过这些步骤。 61. # ### 1.1 切词 62. # In[6]: 63. do\_segmentation\_flag = False 64. # 切词函数 65. **def** seg\_line(line): 66. tokens = jieba.cut(line, cut\_all=False) 67. **return** " ".join(tokens) 68. # 分句处理函数 69. **def** proc(line): 70. tokens = line.split("|") 71. result = [] 72. **for** t **in** tokens: 73. result.append(seg\_line(t)) 74. **return** " | ".join(result) 76. **if** do\_segmentation\_flag: 77. # 读入数据集 78. # 这里的前四个文件使用了原比赛的数据集 79. # 第五个文件final\_round\_test\_set.csv即测试集，使用了课程比赛中的测试集 80. train = pd.read\_csv("train\_set.csv") 81. dev\_1 = pd.read\_csv("1st\_round\_dev\_set.csv") 82. test\_1 = pd.read\_csv("1st\_round\_test\_set.csv") 83. dev = pd.read\_csv("final\_round\_dev\_set.csv") 84. test = pd.read\_csv("final\_round\_test\_set.csv") 86. # 进行分词 87. **for** k **in** ['Brand', 'Collection', 'Problem', 'Conversation']: 88. **print** k 89. # 分别对训练集，开发集，测试集进行操作 90. train[k] = train[k].apply(proc) 91. dev[k] = dev[k].apply(proc) 92. test[k] = test[k].apply(proc) 93. dev\_1[k]=dev\_1[k].apply(proc) 94. test\_1[k]=test\_1[k].apply(proc) 96. num=len(train['Report'] ) 97. **for** i **in** range(num): 98. **if**(isinstance(train['Report'][i],float)): # 此函数用于判断某单元格内容是否为浮点类型 99. train['Report'][i]=" "  102. # 训练集，开发集含有Report列，同样进行操作 103. train['Report'] = train['Report'].apply(proc) 104. dev['Report'] = dev['Report'].apply(proc) 105. dev\_1['Report'] = dev\_1['Report'].apply(proc) 107. # 保存分词后的文件 108. train.to\_csv("train\_set.seg.csv", index=False, encoding='utf-8') 109. dev.to\_csv("final\_round\_dev\_set.seg.csv", index=False, encoding='utf-8') 110. test.to\_csv("final\_round\_test\_set.seg.csv", index=False, encoding='utf-8') 111. dev\_1.to\_csv('1st\_round\_dev\_set.seg.csv',index=False, encoding='utf-8') 112. test\_1.to\_csv('1st\_round\_test\_set.seg.csv',index=False, encoding='utf-8') 113. # ### 1.2 构建字典 114. # In[7]: 115. build\_vocab\_flag = False 116. # 统计字典函数 117. **def** stat\_dict(lines): 118. word\_dict = {} 119. **for** line **in** lines: 120. tokens = line.split(" ") 121. **for** t **in** tokens: 122. t = t.strip() 123. **if** t: 124. word\_dict[t] = word\_dict.get(t,0) + 1 125. **return** word\_dict 126. # 过滤字典函数 127. **def** filter\_dict(word\_dict, min\_count=3): 128. out\_dict = copy.deepcopy(word\_dict) 129. **for** w,c **in** out\_dict.items(): 130. **if** c < min\_count: 131. del out\_dict[w] 132. **return** out\_dict 133. # 构建字典函数 134. **def** build\_vocab(lines, min\_count=3): 135. word\_dict = stat\_dict(lines) 136. word\_dict = filter\_dict(word\_dict, min\_count) 137. sorted\_dict = sorted(word\_dict.items(), key=lambda x:x[1], reverse=True) 138. sorted\_words = [w for w,c in sorted\_dict] 139. sorted\_words = [start\_token, end\_token, unk\_token] + sorted\_words 140. vocab = dict([(w,i) for i,w in enumerate(sorted\_words)]) 141. reverse\_vocab = dict([(i,w) for i,w in enumerate(sorted\_words)]) 142. **return** vocab, reverse\_vocab 144. # 保存字典函数 145. **def** save\_vocab(vocab, path): 146. output = codecs.open(path, "w", "utf-8") 147. **for** w,i **in** sorted(vocab.items(), key=lambda x:x[1]): 148. output.write("%s %d\n" %(w,i)) 149. output.close() 151. # 构建字典函数 152. **def** load\_vocab(path): 153. vocab = {} 154. input = codecs.open(path, "r", "utf-8") 155. lines = input.readlines() 156. input.close() 157. **for** l **in** lines: 158. w, c = l.strip().split(" ") 159. vocab[w] = int(c) 160. **return** vocab 162. **if** build\_vocab\_flag: 163. # 读取分词后的训练集 164. train = pd.read\_csv("train\_set.seg.csv", encoding='utf-8') 166. lines = [] 167. # 将训练集中的Problem, Conversation, Report三列数据作为字典的构建语料 168. for k in ['Problem', 'Conversation', 'Report']: 169. lines.extend(list(train[k].values)) 171. vocab, reverse\_vocab = build\_vocab(lines, min\_count) 173. save\_vocab(vocab, vocab\_path) 174. vocab = load\_vocab(vocab\_path) 175. # ### 1.3 预训练词向量 176. # In[8]: 177. train\_word2vec\_flag=False 178. **if** train\_word2vec\_flag: 179. train = pd.read\_csv("train\_set.seg.csv", encoding='utf-8') 180. dev = pd.read\_csv("final\_round\_dev\_set.seg.csv", encoding='utf-8') 181. test = pd.read\_csv("final\_round\_test\_set.seg.csv", encoding='utf-8') 183. lines = [] 184. **for** k **in** ['Problem', 'Conversation', 'Report']: 185. lines.extend(list(train[k].apply(lambda x:x.split(" ")).values)) 186. lines.extend(list(dev[k].apply(lambda x:x.split(" ")).values)) 187. **for** k **in** ['Problem', 'Conversation']: 188. lines.extend(list(test[k].apply(lambda x:x.split(" ")).values)) 189. # min\_count参数通过设置min\_count参数进行控制，来忽略出现次数较少的单词 190. # size参数主要是用来设置神经网络的层数，这里为512 191. # workers参数用于设置并发训练时候的线程数 192. # 构建模型 193. model = Word2Vec(lines, size=embedding\_size, window=5, sg=1, min\_count=5, workers=4) 194. # 保存模型 195. model.wv.save\_word2vec\_format(word2vec\_file, binary=False) 196. # ### 1.4 文本转ID 197. # In[9]: 198. transform\_data\_flag=False 199. # 读取数据函数 200. **def** read\_data(df, max\_problem\_len, max\_conversation\_len, max\_report\_len): 201. problem\_lens = df['Problem'].apply(lambda x: len(x.split(" "))) 202. conversation\_lens = df['Conversation'].apply(lambda x: len(x.split(" "))) 203. report\_lens = df['Report'].apply(lambda x: len(x.split(" "))) 204. data = [] 205. **for** i **in** range(len(df)): 206. **if** problem\_lens[i] > max\_problem\_len or conversation\_lens[i] > max\_conversation\_len or report\_lens[i] > max\_report\_len: 207. continue 208. item = df.iloc[i] 209. data.append([[start\_token] + item['Problem'].split(" ") + [end\_token], 210. [start\_token] + item['Conversation'].split(" ") + [end\_token], 211. [start\_token] + item['Report'].split(" ") + [end\_token]]) 212. **return** data 213. # 读取测试数据函数 214. **def** read\_test\_data(df): 215. data = [] 216. **for** i **in** range(len(df)): 217. item = df.iloc[i] 218. problem\_vec = item['Problem'].split(" ")[0:max\_problem\_len] 219. conversation\_vec = item['Conversation'].split(" ")[0:max\_conversation\_len] 220. data.append([[start\_token] + problem\_vec + [end\_token], 221. [start\_token] + conversation\_vec + [end\_token]]) 222. **return** data 223. # 转换数据函数 224. **def** transform\_data(data, vocab): 225. # 将词转换为ID 226. out\_data = [] 227. **for** d **in** data: 228. tmp\_d = [] 229. **for** sent **in** d: 230. tmp\_d.append([vocab.get(t, unk\_id) for t in sent if t]) 231. out\_data.append(tmp\_d) 232. **return** out\_data 234. **if** transform\_data\_flag: 235. train = pd.read\_csv("train\_set.seg.csv", encoding='utf-8') 236. dev1 = pd.read\_csv("1st\_round\_dev\_set.seg.csv", encoding='utf-8') 237. dev2 = pd.read\_csv("final\_round\_dev\_set.seg.csv", encoding='utf-8') 238. vocab = load\_vocab(vocab\_path) 239. # 构建新开发集 240. new\_dev = dev2[~dev2['QID'].isin(dev1['QID'].values)] 241. new\_dev = new\_dev.reset\_index(drop=True) 242. train\_set = read\_data(train, max\_problem\_len, max\_conversation\_len, max\_report\_len) 243. train\_set.extend( 244. read\_data(new\_dev, max\_problem\_len, max\_conversation\_len, max\_report\_len)) 245. dev\_set = read\_data(dev1, max\_problem\_len, max\_conversation\_len, max\_report\_len) 246. **print** len(train\_set) 247. **print** len(dev\_set) 248. train\_set\_ids = transform\_data(train\_set, vocab) 249. # 写入json文件 250. output = open(data\_path, "w") 251. json.dump({"train": train\_set\_ids}, output) 252. output.close() 253. # ## Part 2 254. # ### 2.1 定义模型 255. # 代码在 [nmt model from Paddle book](https://github.com/PaddlePaddle/book/blob/develop/08.machine\_translation) 的基础上修改得到。 256. # 257. # 一些值得提到的修改为： 258. # 1. 使用同一个bi-Gru encoder（见下方代码49～63行）对problem （下方代码66~79行）和conversation （下方代码81~88行）进行编码， 259. # 2. 在attention decoder的时候不仅使用前一时刻的状态向量，还引入了problem的bi-Gru encoder结果 （下方代码120行） 260. # 3. encoder和decoder使用了同一个embedding矩阵 (见下方代码第41行和130~136行) 261. # In[10]: 262. with\_gpu=True 263. **print** "with gpu: ", with\_gpu 264. # 保存模型函数 265. **def** save\_model(trainer, save\_path): 266. with open(save\_path, 'w') as f: 267. trainer.save\_parameter\_to\_tar(f) 268. # 载入模型函数 269. **def** load\_word2vec\_file(word2vec\_file): 270. word2vec\_dict = {} 271. input = codecs.open(word2vec\_file, "r", "utf-8") 272. lines = input.readlines() 273. input.close() 274. word\_num, dim = lines[0].split(" ") 275. word\_num = int(word\_num) 276. dim = int(dim) 278. lines = lines[1:] 279. **for** l **in** lines: 280. l = l.strip() 281. tokens = l.split(" ") 282. **if** len(tokens) != dim + 1: 283. continue 284. w = tokens[0] 285. v = np.array(map(lambda x:float(x), tokens[1:])) 286. word2vec\_dict[w] = v 287. **return** word2vec\_dict, dim 288. # Seq2Seq模型函数 289. # 参考PaddlePaddle机器翻译模型文档：https://github.com/PaddlePaddle/book/tree/develop/08.machine\_translation 290. def seq\_to\_seq\_net(vocab\_size, 291. is\_generating, 292. word\_vector\_dim=100, 293. hidden\_size=100, 294. beam\_size=3, 295. max\_length=100, 296. dropout\_rate=0.): 297. # 网络结构 298. decoder\_size = hidden\_size # GRU解码器隐藏单元的维度 299. encoder\_size = hidden\_size # GRU编码器隐藏单元的维度 300. embedding\_param = paddle.attr.ParamAttr(name='embedding') 301. **def** embedding\_layer(word\_id): 302. embed = paddle.layer.embedding( 303. input=word\_id, size=word\_vector\_dim, param\_attr=embedding\_param) 304. **return** paddle.layer.dropout(input=embed, dropout\_rate=dropout\_rate) 305. **def** BiGru\_layer(embedding): 306. forward = paddle.networks.simple\_gru2( 307. input=embedding, size=encoder\_size, 308. gru\_param\_attr=paddle.attr.ParamAttr(name='gru\_forward\_encoder'), 309. #gru\_bias\_attr=False, 310. #mixed\_bias\_attr=False, 311. mixed\_param\_attr=paddle.attr.ParamAttr(name='mixed\_forward\_encoder')) 313. backward = paddle.networks.simple\_gru2( 314. input=embedding, size=encoder\_size, reverse=True, 315. gru\_param\_attr=paddle.attr.ParamAttr(name='gru\_backward\_encoder'), 316. #gru\_bias\_attr=False, 317. #mixed\_bias\_attr=False, 318. mixed\_param\_attr=paddle.attr.ParamAttr(name='mixed\_backward\_encoder')) 319. **return** forward, backward 320. #### 编码器 321. #### 将problem编码为固定长度的向量 322. problem\_word\_id = paddle.layer.data( 323. name='problem\_word', 324. type=paddle.data\_type.integer\_value\_sequence(vocab\_size)) 325. problem\_embedding = embedding\_layer(problem\_word\_id) 326. problem\_f, problem\_b = BiGru\_layer(problem\_embedding) 327. problem\_f\_last = paddle.layer.last\_seq(input=problem\_f) 328. problem\_b\_first = paddle.layer.first\_seq(input=problem\_b) 329. problem\_vector = paddle.layer.concat(input=[problem\_f\_last, problem\_b\_first]) 330. problem\_vector = paddle.layer.dropout(input=problem\_vector, dropout\_rate=dropout\_rate) 331. ##### 将conversation编码为固定长度的向量 332. conversation\_word\_id = paddle.layer.data( 333. name='conversation\_word', 334. type=paddle.data\_type.integer\_value\_sequence(vocab\_size)) 335. conversation\_embedding = embedding\_layer(conversation\_word\_id) 336. conversation\_f, conversation\_b = BiGru\_layer(conversation\_embedding) 337. conversation\_vector = paddle.layer.concat(input=[conversation\_f, conversation\_b]) 338. #### 解码器 339. conversation\_proj = paddle.layer.fc( 340. act=paddle.activation.Linear(), 341. size=decoder\_size, 342. bias\_attr=False, 343. input=conversation\_vector) 344. backward\_first = paddle.layer.first\_seq(input=conversation\_b) 345. decoder\_boot = paddle.layer.fc( 346. size=decoder\_size, 347. act=paddle.activation.Tanh(), 348. bias\_attr=False, 349. input=backward\_first) 350. # attention机制：当前一步输出St应该对齐哪一步输入，主要取决于前一步输出St-1和这一步输入的encoder结果hj 351. **def** gru\_decoder\_with\_attention( 352. enc\_vec, enc\_proj, problem\_vec, current\_word): 353. decoder\_mem = paddle.layer.memory( 354. name='gru\_decoder', size=decoder\_size, boot\_layer=decoder\_boot) 355. context = paddle.networks.simple\_attention( 356. encoded\_sequence=enc\_vec, 357. encoded\_proj=enc\_proj, 358. decoder\_state=decoder\_mem) 359. decoder\_inputs = paddle.layer.fc( 360. act=paddle.activation.Linear(), 361. size=decoder\_size \* 3, 362. bias\_attr=False, 363. input=[context, problem\_vec, current\_word], 364. layer\_attr=paddle.attr.ExtraLayerAttribute( 365. error\_clipping\_threshold=100.0)) 366. gru\_step = paddle.layer.gru\_step( 367. name='gru\_decoder', 368. input=decoder\_inputs, 369. output\_mem=decoder\_mem, 370. size=decoder\_size) 372. out = paddle.layer.mixed(size=vocab\_size, 373. act=paddle.activation.Softmax(), 374. bias\_attr=False, 375. ## 这里使用 embedding 矩阵参数 产生 report 的词 376. input=paddle.layer.trans\_full\_matrix\_projection( 377. input=gru\_step, 378. param\_attr=embedding\_param)) 380. **return** out 381. decoder\_group\_name = 'decoder\_group' 382. group\_input1 = paddle.layer.StaticInput(input=conversation\_vector) 383. group\_input2 = paddle.layer.StaticInput(input=conversation\_proj) 384. group\_input3 = paddle.layer.StaticInput(input=problem\_vector) 385. group\_inputs = [group\_input1, group\_input2, group\_input3] 386. if not is\_generating: 387. report\_embedding = embedding\_layer(paddle.layer.data( 388. name='report\_word', 389. type=paddle.data\_type.integer\_value\_sequence(vocab\_size))) 390. group\_inputs.append(report\_embedding) 391. decoder = paddle.layer.recurrent\_group( 392. name=decoder\_group\_name, 393. step=gru\_decoder\_with\_attention, 394. input=group\_inputs) 395. lbl = paddle.layer.data( 396. name='report\_next\_word', 397. type=paddle.data\_type.integer\_value\_sequence(vocab\_size)) 398. cost = paddle.layer.classification\_cost(input=decoder, label=lbl) 399. **return** cost 400. else: 401. report\_embedding = paddle.layer.GeneratedInput( 402. size=vocab\_size, 403. embedding\_name='embedding', 404. embedding\_size=word\_vector\_dim) 405. group\_inputs.append(report\_embedding) 406. beam\_gen = paddle.layer.beam\_search( 407. name=decoder\_group\_name, 408. step=gru\_decoder\_with\_attention, 409. input=group\_inputs, 410. bos\_id=0, 411. eos\_id=1, 412. beam\_size=beam\_size, 413. max\_length=max\_length) 414. **return** beam\_gen 415. # ### 2.2 训练和预测 416. # In[ ]: 417. **print**(paddle.\_\_file\_\_) 418. # In[11]: 419. **def** infer(inferer, reverse\_vocab, data, batch\_size=32, beam\_size=3): 420. **def** \_infer\_a\_batch(inferer, reverse\_vocab, batch\_data, beam\_size=3): 421. beam\_result = inferer.infer( 422. input=batch\_data, 423. field=['prob', 'id']) 424. gen\_sen\_idx = np.where(beam\_result[1] == -1)[0] 425. assert len(gen\_sen\_idx) == len(batch\_data) \* beam\_size 426. # -1是生成序列的定界符 427. # 每个生成序列的第一个元素的长度 428. batch\_out = [] 429. start\_pos, end\_pos = 1, 0 430. **for** i, sample **in** enumerate(batch\_data): 431. **for** j **in** xrange(beam\_size): 432. end\_pos = gen\_sen\_idx[i \* beam\_size + j] 433. #print("%.4f\t%s" % (beam\_result[0][i][j], " ".join( 434. # reverse\_vocab[w] for w in beam\_result[1][start\_pos:end\_pos]))) 435. if j == 0: 436. batch\_out.append("".join(reverse\_vocab[w] for w in beam\_result[1][start\_pos:end\_pos-1])) 437. start\_pos = end\_pos + 2 438. del beam\_result 439. beam\_result = None 440. **return** batch\_out 441. infer\_results = [] 442. test\_batch = [] 443. **for** idx, item **in** enumerate(data): 444. test\_batch.append(item) 445. **if** len(test\_batch) == batch\_size: 446. test\_results = \_infer\_a\_batch(inferer, reverse\_vocab, test\_batch, beam\_size=beam\_size) 447. infer\_results.extend(test\_results) 448. test\_batch = [] 449. sys.stdout.write('.') 450. sys.stdout.flush() 451. **if** len(test\_batch): 452. test\_results = \_infer\_a\_batch(inferer, reverse\_vocab, test\_batch, beam\_size=beam\_size) 453. infer\_results.extend(test\_results) 454. test\_batch = [] 455. **return** infer\_results 456. # 主函数 457. **def** main(vocab, is\_generating=False, model\_path=None, w2v\_path=None, 458. embedding\_size=100, hidden\_size=100, lr=5e-4, batch\_size=32, n\_epoch=2, 459. out\_model\_prefix=None, data\_name="dev", out\_path="dev\_result.csv", 460. max\_length=100, dropout\_rate=0.): 462. paddle.init(use\_gpu=with\_gpu, trainer\_count=1) 464. # 载入字典 465. reverse\_vocab = dict([(i,w) for w,i in vocab.items()]) 467. # 源字典和目标字典维度 468. dict\_size = len(vocab) 469. vocab\_size = vocab\_size = dict\_size 471. # 训练网络 472. **if** not is\_generating: 473. # 定义优化方法和训练器 474. optimizer = paddle.optimizer.Adam( 475. learning\_rate=lr, # 默认5e-4 476. regularization=paddle.optimizer.L2Regularization(rate=8e-4)) 477. cost = seq\_to\_seq\_net(vocab\_size, is\_generating, word\_vector\_dim=embedding\_size, 478. hidden\_size=hidden\_size, dropout\_rate=dropout\_rate) 479. **if** model\_path: 480. parameters = paddle.parameters.Parameters.from\_tar( 481. open(model\_path, "r")) 482. **else**: 483. parameters = paddle.parameters.create(cost) 484. **if** w2v\_path: 485. word2vec\_dict, dim = load\_word2vec\_file(w2v\_path) 486. assert dim == embedding\_size 487. embedding\_matrix = np.random.normal(loc=0, scale=0.1, size=(len(vocab), dim)) 488. **for** word, i **in** vocab.items(): 489. embedding\_vector = word2vec\_dict.get(word) 490. if embedding\_vector is not None: embedding\_matrix[i] = embedding\_vector 492. parameters.set('embedding', embedding\_matrix) 493. del word2vec\_dict 494. word2vec\_dict = None 495. trainer = paddle.trainer.SGD( 496. cost=cost, parameters=parameters, update\_equation=optimizer) 497. # 定义数据读取器 498. qichedashi\_reader = paddle.batch( 499. paddle.reader.shuffle( 500. train\_reader, buf\_size=100000), 501. batch\_size=batch\_size) 502. # 定义事件处理器响应 503. **def** event\_handler(event): 504. save\_every = 0 505. **if** isinstance(event, paddle.event.EndIteration): 506. **if** event.batch\_id % 30 == 0: 507. **print**("\nPass %d, Batch %d, Cost %f, %s" % 508. (event.pass\_id, event.batch\_id, event.cost, 509. event.metrics)) 510. **else**: 511. sys.stdout.write('.') 512. sys.stdout.flush() 514. **if** save\_every > 0 and not event.batch\_id % save\_every: 515. last\_save\_path = '%s\_params\_pass\_%05d\_batch\_%05d.tar' % ( 516. out\_model\_prefix, event.pass\_id, event.batch\_id - save\_every) 517. save\_path = '%s\_params\_pass\_%05d\_batch\_%05d.tar' % ( 518. out\_model\_prefix, event.pass\_id, event.batch\_id) 519. save\_model(trainer, save\_path) 520. **if** event.batch\_id > 0: 521. os.remove(last\_save\_path) 523. **if** isinstance(event, paddle.event.EndPass): 524. # 保存参数 525. save\_path = '%s\_params\_pass\_%05d.tar' % (out\_model\_prefix, event.pass\_id) 526. save\_model(trainer, save\_path) 527. # 开始训练 528. trainer.train( 529. reader=qichedashi\_reader, event\_handler=event\_handler, num\_passes=n\_epoch) 530. # 预测 531. **else**: 532. # 输出三个样例看看 533. # 使用前三个样本进行生成 534. gen\_data = [] 535. real\_data = [] 536. gen\_num = 3 537. **for** item **in** train\_reader(): 538. gen\_data.append([item[0],item[1]]) 539. real\_data.append([item[0],item[1],item[2]]) 540. **if** len(gen\_data) == gen\_num: 541. **break** 542. beam\_size = 3 543. beam\_gen = seq\_to\_seq\_net(vocab\_size, is\_generating, word\_vector\_dim=embedding\_size, 544. hidden\_size=hidden\_size, beam\_size=beam\_size, max\_length=max\_length, 545. dropout\_rate=dropout\_rate) 546. # 载入训练好的模型 547. parameters = paddle.parameters.Parameters.from\_tar( 548. open(model\_path, "r")) 549. # prob 是预测概率，id 是预测词 550. beam\_result = paddle.infer( 551. output\_layer=beam\_gen, 552. parameters=parameters, 553. input=gen\_data, 554. field=['prob', 'id']) 556. gen\_sen\_idx = np.where(beam\_result[1] == -1)[0] 557. assert len(gen\_sen\_idx) == len(gen\_data) \* beam\_size 558. # -1是生成序列的定界符 559. # 每个生成序列的第一个元素的长度 560. start\_pos, end\_pos = 1, 0 561. **for** i, sample **in** enumerate(real\_data): 562. **print**( 563. " ".join([reverse\_vocab[w] for w in sample[0][1:-1]]) 564. ) # 在打印源句时跳过开始和结束标记 565. **print**( 566. " ".join([reverse\_vocab[w] for w in sample[1][1:-1]]) 567. ) 568. **print**( 569. " ".join([reverse\_vocab[w] for w in sample[2]]) 570. ) 571. **for** j **in** xrange(beam\_size): 572. end\_pos = gen\_sen\_idx[i \* beam\_size + j] 573. print("%.4f\t%s" % (beam\_result[0][i][j], " ".join( 574. reverse\_vocab[w] for w in beam\_result[1][start\_pos:end\_pos]))) 575. start\_pos = end\_pos + 2 576. print("\n") 577. # 在测试集/开发集 上输出 578. # 预测数据 579. df = pd.read\_csv("%s\_set.seg.csv" %(data\_name), encoding='utf-8') 580. data = read\_test\_data(df) 581. data = transform\_data(data, vocab) 583. inferer = paddle.inference.Inference( 584. output\_layer=beam\_gen, parameters=parameters) 585. infer\_results = infer(inferer, reverse\_vocab, data, batch\_size=32, beam\_size=3) 586. #for r in infer\_results: 587. # print(r) 588. submit = df[['QID']] 589. submit['Prediction'] = infer\_results 590. submit.to\_csv(out\_path, encoding="utf-8", index=False) 591. # ### 2.3 载入字典和数据 592. # In[12]: 593. ## 读取器生成函数 ## 594. **def** reader\_creator(data): 595. **def** reader(): 596. **for** p\_ids, c\_ids, r\_ids **in** data: 597. yield p\_ids, c\_ids, r\_ids[:-1], r\_ids[1:] 598. return reader 599. # In[13]: 600. ## 载入字典 ## 601. vocab = load\_vocab(vocab\_path) 602. ## 载入数据 ## 603. input = open(data\_path) 604. data = json.load(input) 605. input.close() 606. train\_set\_ids = data['train'] 607. #dev\_set\_ids = data['dev'] 608. #test\_set\_ids = data['test'] 609. del data 610. train\_reader = reader\_creator(train\_set\_ids) 611. # ### 2.4 训练 612. # 训练可以分为两个阶段: 613. # \* 第一个阶段是初始训练开始，默认学习率是5e-4 614. # \* 第二个阶段可以载入现有模型继续训练，主要可以手动调节学习率 615. # In[ ]: 616. print word2vec\_file 617. # In[ ]: 618. # 第一次训练 619. first\_train\_flag = True 620. **if** first\_train\_flag: 621. main(vocab, w2v\_path=word2vec\_file, embedding\_size=embedding\_size, hidden\_size=hidden\_size, 622. out\_model\_prefix="/home/aistudio/model/d%d" %(hidden\_size), n\_epoch=15, batch\_size=32) 623. # In[ ]: 624. # 持续训练 625. continue\_train\_flag = True 626. **if** continue\_train\_flag: 627. main(vocab, model\_path="model/d512\_params\_pass\_00014.tar", out\_model\_prefix="/home/aistudio/model\_2\_30turns/d512\_p4\_1e-5", 628. embedding\_size=embedding\_size, hidden\_size=hidden\_size, 629. n\_epoch=30, batch\_size=16, lr=1e-5) 630. # ### 2.5 验证 631. # 需要手动输入要在开发集上测试的模型的名称 632. # 633. # 另外： 1st\_round\_dev\_set.csv 这个文件是数据集中开发集文件仅保留QID和Report两列的结果。 634. # In[ ]: 635. dev\_flag = True 636. **if** dev\_flag: 638. dev\_model="model/d512\_p4\_1e-5\_params\_pass\_00001" 639. data\_name="1st\_round\_dev" 640. out\_path="%s.%s.csv" %(dev\_model, data\_name) 641. main(vocab, is\_generating=True, model\_path=dev\_model+".tar", embedding\_size=embedding\_size, hidden\_size=hidden\_size, 642. data\_name=data\_name, out\_path=out\_path, max\_length=60) 643. # ### 2.6 预测 644. # 输出预测文件 645. # In[1]: 646. # predict 647. predict\_flag = True 648. **if** predict\_flag: 649. data\_name="final\_round\_test" 650. out\_path="~/model\_2/t4\_5e-4\_1e-5\_twoturn\_3.csv" 651. main(vocab, is\_generating=True, model\_path="model\_2/d512\_p4\_1e-5\_params\_pass\_00014.tar", embedding\_size=embedding\_size, hidden\_size=hidden\_size, 652. data\_name=data\_name, out\_path=out\_path, max\_length=100) |

**注意：**

课程报告用PDF格式在超星平台（www.elearning.shu.edu.cn）上交。

文件名格式：学号-姓名.PDF，例如：18120000-张三.PDF