第十周作业文档

崔萌

文档需求：

问题描述

使用 tensorflow 中的 rnn 相关操作，以作业提供的《全宋词》为训练数据，训练一个人工智能写词机。word embedding 部分在本地完成：参考 https://www.tensorflow.org/tutorials/word2vec 的内容，以下述脚本为基础，完成对本作业提供的《全宋词》的 embedding.
https://www.github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/word2vec/word2vec_basic.py

详情见：https://gitee.com/ai100/quiz-w10-code

解题提示

全宋词资料不同于英文，不使用分词，这里直接将每个单字符作为一个 word。

全宋词全文共 6010 个不同的单字符，这里只取出现次数最多的前 5000 个单字符。

后面的 RNN 训练部分，需要使用 embedding 的 dictionary，reversed_dictionary,请使用 json 模块的 save 方法将这里生成的两个字典保存起来。utils 中也提供了一个字典的生成方法，RNN 作业部分，如果不使用这个作业生成的 embedding.npy 文件作为 model 的 embeding 参数（参考 model 的 build 方法中的 embedding_file 参数）的时候可以使用这个 utils 中提供的方法直接生成这两个字典文件。

matplotlib 中输出中文的时候会出现乱码，请自行搜索如何设置 matplotlib 使之可以输出中文。

按照 tensorflow 官方代码中给出的设置，运行 40W 个 step 可以输出一个比较好的结果，四核 CPU 上两三个小时左右。

对于文本的处理，可以搜到很多不同的处理方式，大部分文本处理都要删掉所有的空格，换行，标点符号等等。这里的训练可以不对文本做任何处理。

本作业中，涉及大量中文的处理，因为 python2 本身对 UTF-8 支持不好，另外官方对 python2 的支持已经快要结束了，推荐本项目使用 python3 进行。

构建 RNN 网络需要的 API 如下，请自行查找 tensorflow 相关文档。

tf.nn.rnn_cell.DropoutWrapper

tf.nn.rnn_cell.BasicLSTMCell

tf.nn.rnn_cell.MultiRNNCell

RNN 部分直接以 embedding 作为输入，所以其 hiddenunit 这里取 128,也就是 embedding 的维度即可。

RNN 的输出是维度 128 的，是个 batch_sizenum_steps128 这种的输出，为了做 loss 方便，对输出进行了一些处理，concat，flatten 等。具体请参考 api 文档和代码。

RNN 输出的维度与 num_words 维度不符，所以需要在最后再加一个矩阵乘法，用一个 128*num_words 的矩阵将输出维度转换为 num_words。

RNN 可能出现梯度爆炸或者消失的问题，对于梯度爆炸，这里直接对 gradient 做了裁剪，细节参考 model 代码。

这里模型的规模比较小，所以输出的内容可能不是特别有意义，而且训练过程中，不同的 checkpoint，其输出也有一些区别。

数据处理中，data 为文本中一段随机截取的文字，label 为 data 对应的下一个标号的文字。以苏轼的江神子（江城子）为例：输入为 “老夫聊发少年”，则对应的 label 为"夫聊发少年狂"。

训练过程至少要到第二个 epoch 才能看到一些比较有意义的输出，第一个 epoch 的输出可能是大量的标点，换行等等。而且这种情况后面还会有。

这里的代码，train_eval.py 用于在 tinymind 上运行训练和采样，按照代码中默认的设置，运行一个 e

poch 需要 19220 步，在 tinymind 上需要半小时左右。

rnn 作业中，dictiionary 和 reverse_dictionary 为汉字的索引，可以使用 word embeding 作业生成的，也可以重新生成这两个字典。如果 model.build 中使用 word embeding 作业中生成的 embeding_file.npy，则为了保证汉字索引的对应关系，必须使用与 embeding_file.npy 一起生成的 dictionary 和 reverse_dictionary

批改标准

代码请使用本作业提供的代码。学员需要实现 RNN 网络部分,RNN 数据处理部分和 RNN 训练部分。

train.py 训练 -20 分

utils.py 数据处理 -20 分

model.py 网络 -20 分

文档描述 -40 分

训练的输出 log 输出中可以看到下述内容：

2018-01--- --:--:-,114 - DEBUG - sample.py:77 - ==========[江神子]==========

2018-01--- --:--:-,114 - DEBUG - sample.py:78 - 江神子寿韵）

一里春风，一里春风，一里春风，一里春风，不是春风。

一里春风，不是春风，不是春风。不是春风，不是春风。

浣溪沙（春

2018-01--- --:--:-,556 - DEBUG - sample.py:77 - ==========[蝶恋花]==========

2018-01--- --:--:-,557 - DEBUG - sample.py:78 - 蝶恋花寿韵）

春风不处。一里春风，一里春风，不是春风。不是春风，不是春风，不是春风。

一里春风，不是春风，不是春风。不是春风，不是

2018-01--- --:--:-,938 - DEBUG - sample.py:77 - ==========[渔家傲]==========

2018-01--- --:--:-,940 - DEBUG - sample.py:78 - 渔家傲

一里春风，一里春风，一里春风，一里春风，不是春风。

水调歌头（寿韵）

春风不处，一里春风，一里春风，一里春风，不是春风。

鉴于代码不是很好判定，而且比较费事，本作业的评判，只要运行结果能出现有意义的词就认为前面三项全部完成，给 60 分

提供一个文档，描述自己对 rnn 的理解和训练 rnn 的过程中的心得体会。对自己输出的结果的理解以及输出的解释。40 分

一、LSTM 的构建

1、使用 tensorFLow 中的函数:

BasicLSTMCell
DropoutWrapper
MultiRNNCell

代码实现

```
# 创建包含 rnn_size 个神经元的 lstm cell
        cell = tf.contrib.rnn.BasicLSTMCell(self.dim_embedding,state_is_tuple=True；
# 使用 dropout 机制防止 overfitting 等
        drop = tf.contrib.rnn.DropoutWrapper(cell, output_keep_prob=keep_prob,input_keep_prob=1.0)
# 创建 2 层 lstm 层
        cell = tf.contrib.rnn.MultiRNNCell([drop for _ in range(self.rnn_layers)], state_is_tuple=True)
# 初始化状态为 0.0
        #print(data_shape_0)
        init_state = cell.zero_state(data_shape_0, tf.float32)
```

2、调用 dynamic_rnn 函数启动 rnn 网络

3、返回输出及状态值供下个循环使用

二、训练的过程

1、数据处理：

先将文本中的空行去掉
整个作业过程中，数据分别截取 10k，100k，1m 做测试，结果合理后，再跑 5m 的全数据

2、获得每次训练的 batch。

用 numStep*batch 将整个数据分份，再逐步推移做循环训练。
获得 batch 数据的函数为 get_train_data，实现请参看源码。

三、验证的过程

1、验证方式：

作业中将默认的验证方式修改了一下，看到输出结果好了很多。

默认的是取一个字当输入，然后输出一个字

这里改成了输入为 ABC，输出为 XYZ，BC 和 XY 会有高相关，然后取 Z 当预测结果，再用 BCZ 当输入，依次循环。

具体实现请参看源码。

四，输出结果：

2018-10-23 11:57:19,505 - DEBUG - sample.py:146 - ==============[夜半乐]==============

2018-10-23 11:57:19,505 - DEBUG - sample.py:147 - 夜半乐，林名一浅横，浅鸾峰已断。画馆、韶华如结。暗会不，罄光地。寸永

夕枕甚却底时，遇笑时佩。玉蛾翠香艳，入新云用，衷愁无情，却衾春息更

2018-10-23 11:57:19,510 - DEBUG - sample.py:148 - 夜半乐_[2]__[67]__[267]__[15]__[232]__[412]__[2]__[232]__[757]__[462]__[264]__[57]__[1]__[89]__[366]__[4]__[474]__[223]__[54]__[535]__[1]__[241]__[207]__[14]__[2]__[1257]__[81]__[210]__[1]__[694]__[164]__[3]__[512]__[191]__[493]__[278]__[778]__[17]__[2]__[479]__[142]__[17]__[1234]__[1]__[90]__[598]__[84]__[37]__[183]__[2]__[243]__[99]__[13]__[969]__[2]__[945]__[58]__[11]__[21]__[2]__[278]__[284]__[18]__[451]__[79]_

2018-10-23 11:57:19,350 - DEBUG - sample.py:146 - ==============[木兰花]==============

2018-10-23 11:57:19,350 - DEBUG - sample.py:147 - 木兰花（四之一。

长要归愁珠面、灯心风，一子夜死，旧把云人，几事如枝春少。高片凤，乘兴兰塘，杏靄清晴，数声点风紧。

郎仓处人、非眉凄，关

2018-10-23 04:56:11,070 - DEBUG - sample.py:146 - ==============[江神子]==============

2018-10-23 04:56:11,070 - DEBUG - sample.py:147 - 江神子

一笑金瓯，一笑风流。

人间。

菩萨蛮

浣溪沙

一叶一枝。

水龙吟

一声声。

西江月（和秋）

一声声。

水调歌头

一笑春风。

浣

从结果看是有意义的。

心得体会：

1、这次的作业明显感觉比之前的作业要难一些。虽然做完了感觉东西不多，但是学习的过程是略微盲目的、主要还是对 lstm 模型的理解，和 tensorflow 里这种参数的对应理解。

2、WordEmbedding 从理解上是个难点。比如如何从验证结果通过 argsort 来获得预测结果。

3、之前 dictionary 是用自己方法写的，不太完善。后面用作业自带的，但是 index 对应方式上有些不同，闹了一个不小的乌龙。作业里是用出现频率排序，然后当序号。

4、Lstm 的那个 rnnLayer 参数，测试过很多次，最后得到的结果是用 1 的时候 loss 降的最低，用 2 的时候输出的结果可能更广泛一些。但是再高的时候，输出的结果经常单字重复。

5、在小数据上使用单字验证的时候看不太出来很多问题，但是在大数据时，单字验证的结果很不好，经常一堆换行。而且如果用默认的 100 层 rnn，训练过程是极其缓慢的…最后还是取了 2 层的 rnn 和三字输入验证。

6、印象很深刻的是 rnn 模型的参数，在作业过程中几乎将所有参数的可变性试了个遍，加深了对 rnn 模型的理解和转化。

7、Tinymind 上的运行地址是：https://www.tinymind.com/executions/fxqp6cbh，

全宋词的数据跑出来的结果。

8、暂时回忆出来的问题是这些。最后还是谢谢老师及助教的帮助，谢谢！