语义计算 QA 大作业报告

【前言】

本项目分为两大部分，其一是基于知识库的问答系统，其二是基于文档的问答系统。报告的后续部分将对这两者分别描述和总结。

【任务一：KBQA】

1. 任务描述

给定一个中文知识库，知识库中包含实体以及实体之间的关系。给定一个问题，求这个问题在中文知识库中的答案。

2. 算法设计

我们通过以下几步来完成答案搜索：

（1） 分词、找到句子中所有可能的 mention

- 通过 ICTCLAS2016 进行中文分词，通过评测提供的 Baseline 中识别 mention 的程序找到这个句子中所有可能的 mention。

（2） 找到句子中实体所对应的 mention

- 在找到当前句子对应的所有可能的 mention 后，我们需要找出句子中实体所对应的 mention，我们发现可以通过规则来识别句子实体所对应的 mention，比如对于书名来说，一般句子的实体就是书名号中的内容；句子的实体一般在句子的起始部分，当过滤掉一部分停顿词后。

（3） 在 mention2id 文件中找到所有的 id

- 通过 mention2id 数据库找到当前 mention 可能对应的所有 id。

（4） 计算所有 id 和当前问题的相似度，找到最相似的 id

- 在找到所有 id 后，我们需要找到最相似的 id，我们的做法是对 id 和问题计算相似度，找出最可能的 id，我们通过平均单词向量得到的句子向量以及 id 和问题的字符重合率来计算相似度。

（5） 计算最相似 id 中所有谓词属性同当前问题所有分词结果的相似度，找出最相似的谓词，根据这个谓词在数据库中找到答案

- id 中所有谓词属性同当前问题所有分词结果的相似度的计算方法和（4）中计算相似度的方法相同。

3. 程序实现

（1） 分词、找到句子中所有可能的 mention

- 通过调用 baseline 中的部分程序输出分词和所有的 mention 结果。

（2） 找到句子中实体所对应的 mention

- 在 mention_extractor.py 文件中实现，通过 get_entity 方法得到一个句子中的实体，主要通过定义一些规则进行识别。

（3） 在 mention2id 文件中找到所有的 id

- 在 mention_id.py 文件中实现，将所有的 mention2id 可能的组合存储在一个字典中，通过实体的 mention 名字找到所有可能的 id。

（4） 计算相似度

- 在 similarity.py 文件中实现，通过 similarity 方法计算两个 string 的相似度，主要通过调用两个函数计算相似度，similarity_overlap 用来计算两个 string 字符重合的数量，similarity_word_vector 用来计算两个 string 向量的相似度。最后进行一个加权处理就可以得到两个 string 的相似度。

（5） 训练和测试函数

- 训练在 main_training.py 文件中实现，主要按照 2 中的步骤进行调参
- 测试在 main_testing.py 文件中实现，主要按照 2 中的步骤进行识别

4. 实验结果

我们对 train 数据集进行了测试，实验结果如下表所示：

Precision	Recall	F1
0.447	0.447	0.447

5. 感想

第一次做 KBQA 相关的问题，思路从无到有，我们在这个过程中学习了很多，实验可以通过以下几个方向进行改进：

（1） 识别 id 后在再计算该 id 属性和句子中单词相似度的方法会导致连贯错误，可以对 id 和属性综合考虑。

【任务二：DBQA】

1. 任务描述

每次给出一个问题、答案的二元组，判断这两个句子是否构成一个逻辑上的问答对。训练语料有三列，分别是问题、答案、标签，测试语料只有问题和答案。总体来说相当于一个二分类问题。

2. 算法设计

我们通过以下几步来完成答案搜索：

（1） 分词，构建单词与 ID 之间的映射

- 调用结巴分词，把每个答案、问题转换成一个单词序列，进一步再转换成单词 ID（即一个整数）的序列。考虑到有些词在训练语料中有而测试预料中没有，所以对未登录词、数字、英文单词、标点符号等进行了特殊处理。

（2） 利用 LSTM 单元构建神经网络，把变长的问题和答案变成定长的 embedding，用来建模句子的语义

- 先对较短的句子做填充，对较长的句子做截断，使得所有句子长度为 200
- 用一层 LSTM 神经元把问题变成 64 维向量，用另一层 LSTM 神经元把答案也变成 64 维向量

（3） 计算问题和答案的 embedding 的余弦夹角

- 若问题和答案确实是配套的，那么希望夹角余弦值为 1（即两个向量方向相同）；否则希望夹角余弦值为 0（即两个向量互相垂直）

3. 程序实现

（1） 数据预处理

- 在 preprocess.py 中定义了一个数据预处理的类，先用结巴分词对句子进行切分，然后给每个词分配一个 ID。经过观察发现直接这么做效果不太好，于是对字母、数字、标点进行了专门的单独处理。

（2） 模型搭建

- 在 model.py 中，使用 Keras 提供的 functional API 实现。
- 单词的 Embedding 使用 Baseline 提供的词向量初始化，代码在 embedding.py 中。

（3） 主程序入口

- 在 model_test.py 文件中实现，实验发现迭代 20 轮左右验证集上的损失就基本不再变化，故设定迭代轮数为 20.
- 中间每一轮迭代的结果和最终测试集上的预测结果都被 dump 下来，可以用 npy_to_score.py 转换成文本文件方便测评程序使用。

4. 实验结果

在 train 数据集上，实验结果如下表所示：

MAP	MRR
0.313	0.314

5. 感想

拿到这个题目以后形成思路很自然，但是结果却不尽如人意。在几个 Baseline 程序中，除了机器翻译的效果比我们的结果差以外，其他几个模型（比如 Word Overlap 等）都超过了我们搭建的 LSTM 神经网络。这可能有两方面的原因：

（1） 其一是网络结构设计的过于简单，或者是目标函数设置不当，直接要求问题和答案的 embedding 相互接近可能不是很合理，也许让问题的 embedding 做一个变换之后再跟答案的 embedding 相接近可能更好。

（2） 其二是没有利用起来单词在拼写上的特点，例如问题和答案里可能有些单词属于换了一种表述但是有些汉字是重叠的，如果把这一部分信息加进来可能会更好。