

智能阅读系统的构建

总体分为两部分：第一部分粗匹配相似句子，第二部分对候选回答排序，选择置信度最高的正确回答。

第一部分可使用 TF-IDF、LSI 等传统方法。

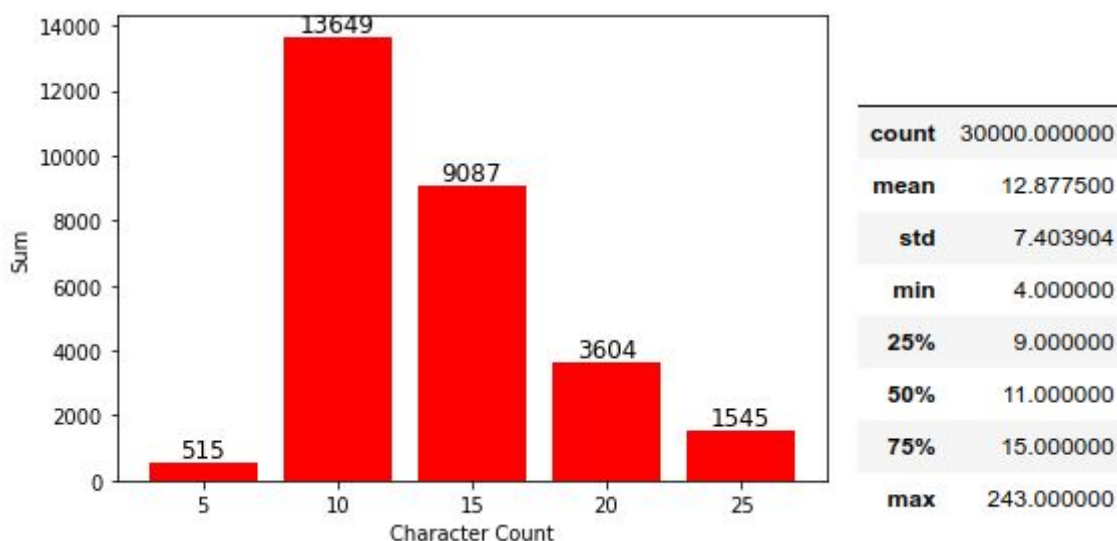
第二部分可使用 基于深度学习的问答系统。

本文主要构建第二部分，第一部分可参考：

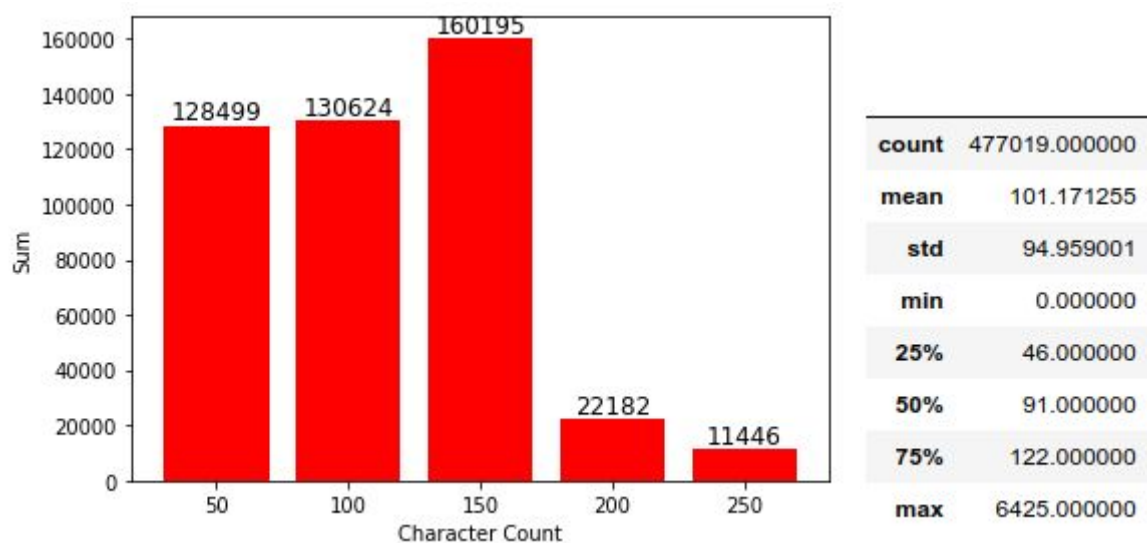
<https://github.com/littleredhat1997/doc-similarity>

一、数据统计

1. 1 分词前统计

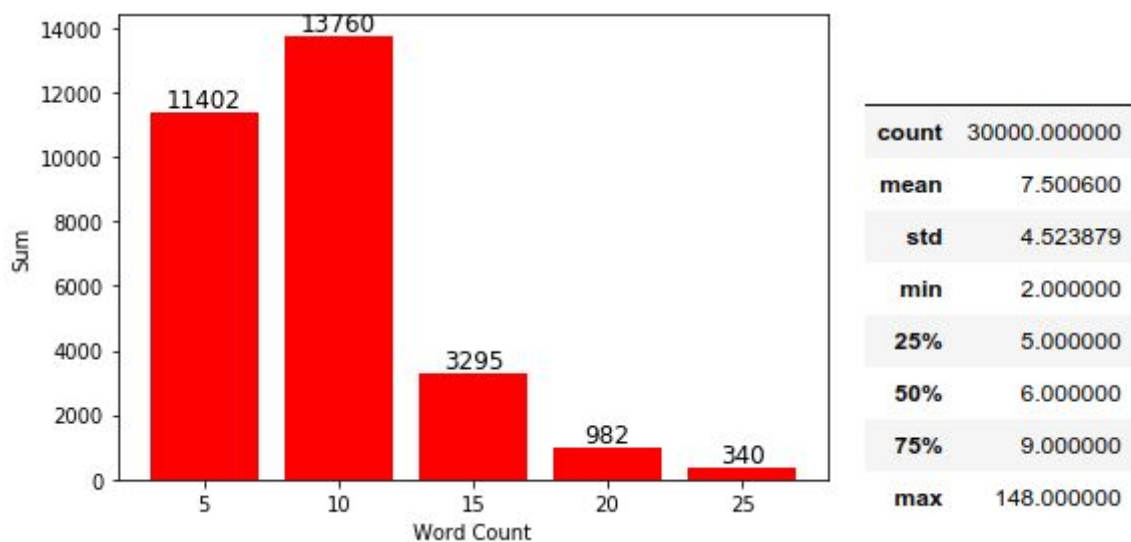


问题数量	最长问题	最短问题	平均长度
30000 个	243 字	4 字	13 字

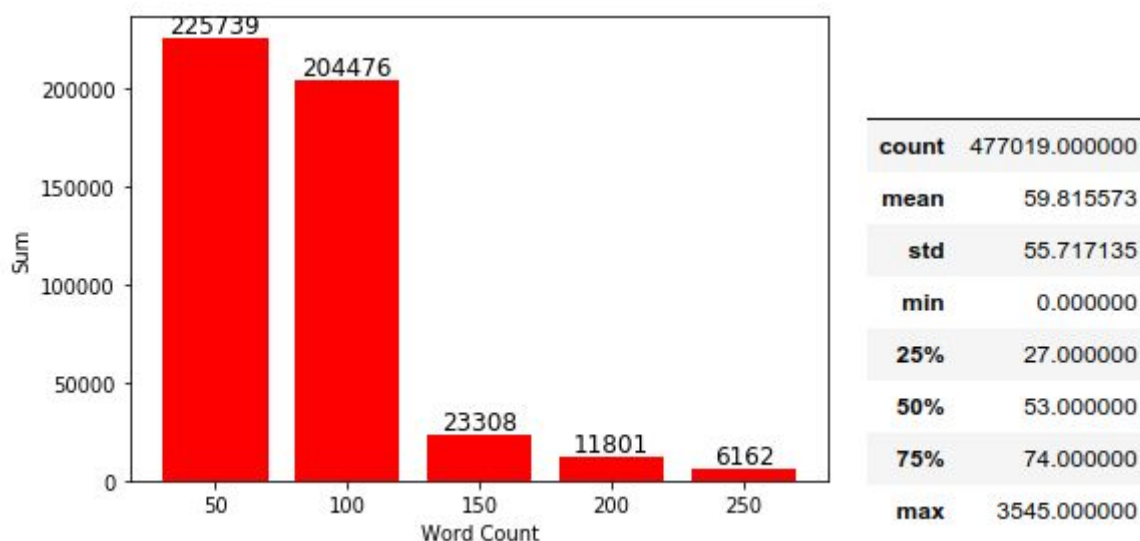


回答数量	正确数量	错误数量	最长回答	最短回答	平均长度
477019 个	127328 个	349691 个	6425 字	0 字	95 字

1. 2 分词后统计



问题数量	最长问题	最短问题	平均长度
30000 个	148 词	2 词	8 词



回答数量	正确数量	错误数量	最长回答	最短回答	平均长度
477019 个	127328 个	349691 个	3545 词	0 词	60 词

经过统计，我们可以得出以下结论：

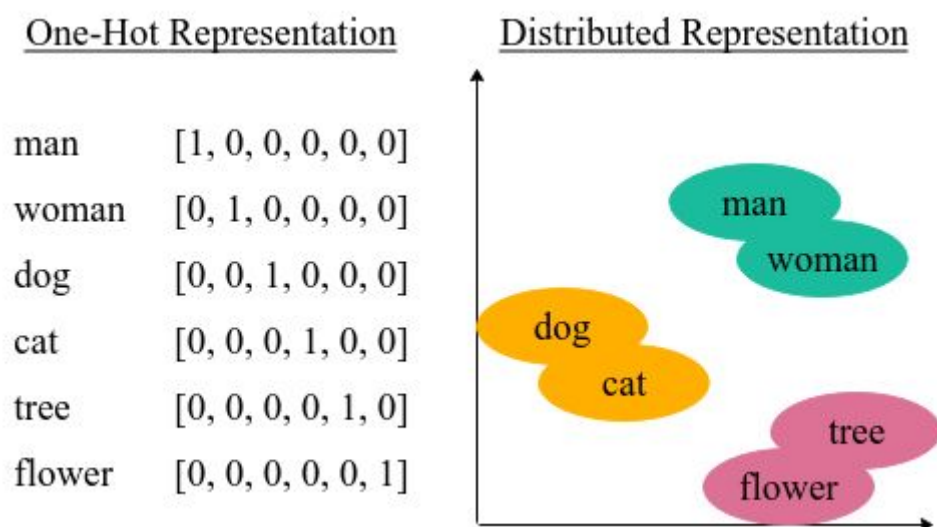
- ①问题和回答的比例大约为 1:15，基本符合候选回答的比例。
- ②正确回答和错误回答的比例大约为 1:3，正负样本比例属于正常范围。
- ③问题普遍较短，平均 13 个字，分词后为 8 个词；回答普遍较长，平均 95 个字，分词后为 60 个词。考虑到问题种类的多样性（事实类/非事实类、实体类/论述类等），该长度比例属于正常范围。
- ④某些回答长篇大论，对于人类而言，实际上并不需要完整理解内容就能判断正误。在训练时，综合回答长度和训练效率，可以限制单个回答长度不超过 200 个词。
- ⑤数据集中存在噪音，例如某些回答为空，某些回答重复。对于这部分数据可以直接剔除，防止干扰训练。

二、数据预处理

自然语言处理的问题要转化为机器学习的问题，第一步肯定是要找一种方法把这些语言符号数学化。

在深度学习应用中，语音应用将音频的信号所构成的向量作为输入；图像应用将图片的像素所构成的矩阵作为输入。这些数据可以自然地表示为一个连续的数字，那在自然语言处理中呢？

另一方面，文字作为人类进化了几百万年的交流工具，具有高度抽象的特征。任意两个词语，可能互为近义词或者反义词，也可能毫无关系。判断两个词语是否存在联系，还需要更多的背景知识才能作出回答。



1. 1 独热编码 (One-Hot Representation)

One-Hot 编码把每个词表示为一个稀疏向量，向量维度是词典大小，其中绝大多数元素为 0，只有一个元素为 1。这种表示方法容易导致“维度灾难”：当维度增加时，需要的存储空间呈指数增长。另一个重要问题就是“词汇鸿沟”：任意两个词之间都是孤立的，光从这两个向量中看不出两个词是否存在关系。

1. 2 词嵌入 (Distributed Representation)

“词向量”（词嵌入）是一类将词的语义映射到向量空间中去的自然语言处理技术。即将一个词用特定的向量来表示，向量之间的距离（例如，任意两个向量之间的 L2 范式距离或更常用的余弦距离）一定程度上表征了的词之间的语义关系。由这些向量形成的几何空间被称为一个嵌入空间。

例如，“椰子”和“北极熊”是语义上完全不同的词，所以它们的词向量在一个合理的嵌入空间的距离将会非常遥远。但“厨房”和“晚餐”是相关的话，所以它们的词向量之间的距离会相对小。

理想的情况下，在一个良好的嵌入空间里，从“厨房”向量到“晚餐”向量的“路径”向量会精确地捕捉这两个概念之间的语义关系。在这种情况下，“路径”向量表示的是“发生的地点”，所以你会期望“厨房”向量 - “晚餐”向量（两个词向量的差异）捕捉到

“发生的地点”这样的语义关系。基本上，我们应该有向量等式：晚餐 + 发生的地点 = 厨房（至少接近）。如果真的是这样的话，那么我们可以使用这样的关系向量来回答某些问题。例如，应用这种语义关系到一个新的向量，比如“工作”，我们应该得到一个有意义的等式，工作+ 发生的地点 = 办公室，来回答“工作发生在哪里？”。

词向量通过降维技术表征文本数据集中的词的共现信息。方法包括神经网络（“Word2vec”技术），或矩阵分解。

1. 3 预处理流程

在进行模型训练之前，需要将问题和回答转换为对应的词向量。以

问题：“**射雕英雄传中谁的武功天下第一**”

回答：“**王重阳武功天下第一**”

为例，生成一个词向量具体步骤如下所示：

①分词 (Cut)：

```
>>> ['射雕 英雄传 中 谁 的 武功 天下第一', '王重阳 武功 天下第一']
```

分词采用 Python 自然语言处理工具 jieba。开发者可以指定自己自定义的词典，以便包含 jieba 词库里没有的词。虽然 jieba 有新词识别能力，但是自行添加新词可以保证更高的正确率。

②字典化 (Tokenizer)：

```
>>> {'中': 5, '天下第一': 2, '射雕': 3, '武功': 1, '王重阳': 8, '的': 7, '英雄传': 4, '谁': 6}
```

将分词后的词语编号，映射到一个数字，用以标识这个词语。

③序列化 (Sequences)：

```
>>> [[3, 4, 5, 6, 7, 1, 2]]
```

```
>>> [[8, 1, 2]]
```

将一个句子中的词语序列化成词向量列表。

④填充字符 (Padding)：

```
>>> [[0 0 0 3 4 5 6 7 1 2]]
```

```
>>> [[0 0 0 0 0 0 0 8 1 2]]
```

深度学习的输入数据为固定长度，因此需要对序列进行填充或截断操作。小于固定长度的序列用 0 填充，大于固定长度的序列被截断，以便符合所需的长度。

1. 4 导出数据集

根据上述流程生成训练数据集和测试数据集，包括：

tokenizer.pkl (语料字典)

train_a.npy (训练问题集) test_a.npy (测试问题集)

train_q.npy (训练回答集) test_q.npy (测试回答集)

train_y.npy (训练标签集) test_y.npy (测试标签集)

后续深度学习中，我们将在模型中的“嵌入层”训练词向量，训练出来的词向量可以更好的适应自然语言处理任务。

三、模型设计

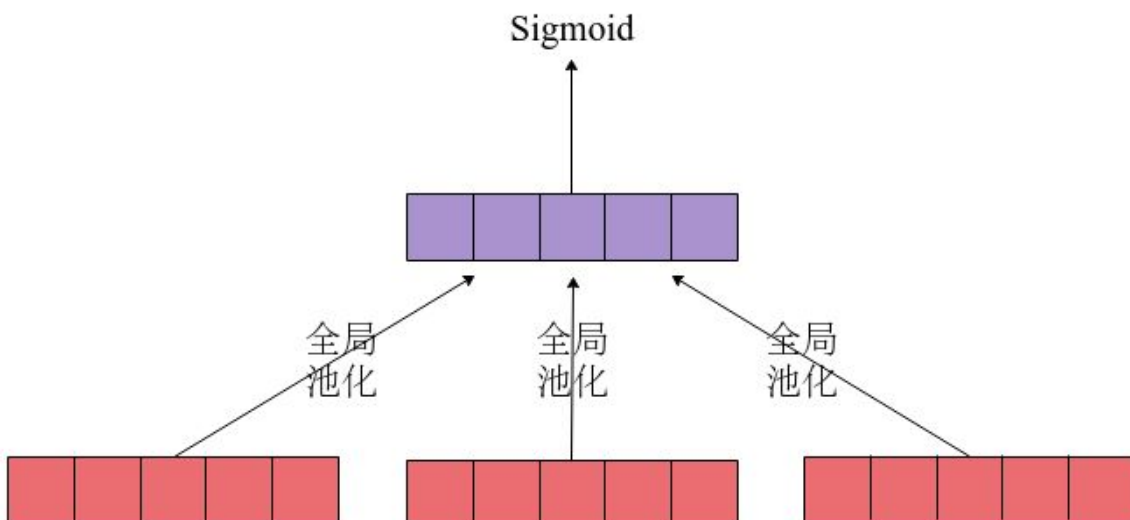
模型架构：FastText、CNN、CNN with Word2Vec、Bi-LSTM、Attention。

评估指标：训练效率、准确率、MRR、MAP、TOP-1。

多输入：问题 Q、回答 A，成对输入。

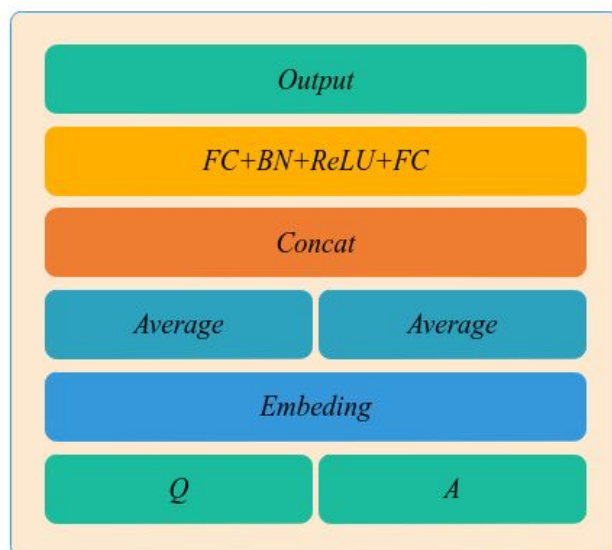
单输出：输出 0 到 1 之间的浮点数，本质是文本二分类，代表问题和回答的匹配程度。0 代表毫无关系，1 代表完全匹配。

1. 1 基于 FastText 模型

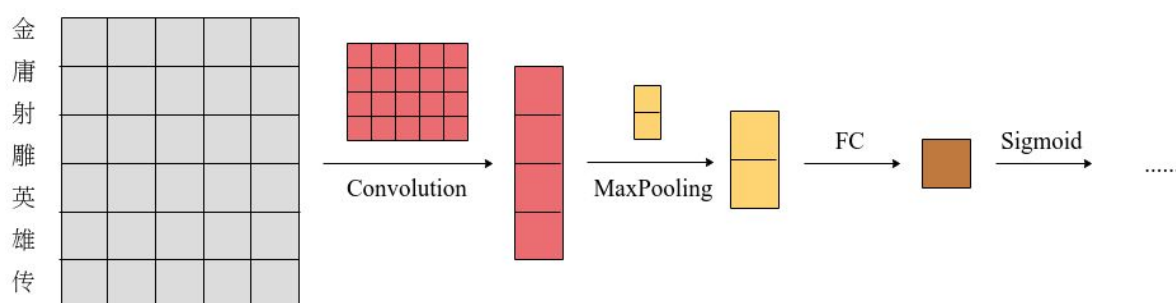


FastText 是一种快速文本分类模型，其核心思想是：将整篇文本的词及 N-gram 向量叠加平均得到文本向量，然后使用文本向量分类。适合海量数据和高速训练，能将训练时间从几小时缩短到几分钟。

>>> **模型架构**：词嵌入后，隐藏层只是一个简单的平均池化层，然后连接经过池化的问题和回答向量，最后的全连接层作为分类器使用。注意这里的输入可以是单词，也可以是 N-gram 组合。



1. 2 基于 CNN 模型



一个普通的 CNN 网络结构包含：

①卷积层 (Convolution Layer)

卷积神经网络最早是应用在计算机视觉当中，而如今 CNN 也早已应用于自然语言处理的各种任务。在图像中卷积核通常是对图像的一小块区域进行计算，而在文本中，一句话所构成的词向量作为输入。

②池化层 (Pooling Layer)

卷积神经网络的一个重要概念就是池化层，一般是在卷积层之后。池化层对输入做降采样。池化的过程实际上是对卷积层分区域求最大值或者对每个卷积层求最大值。

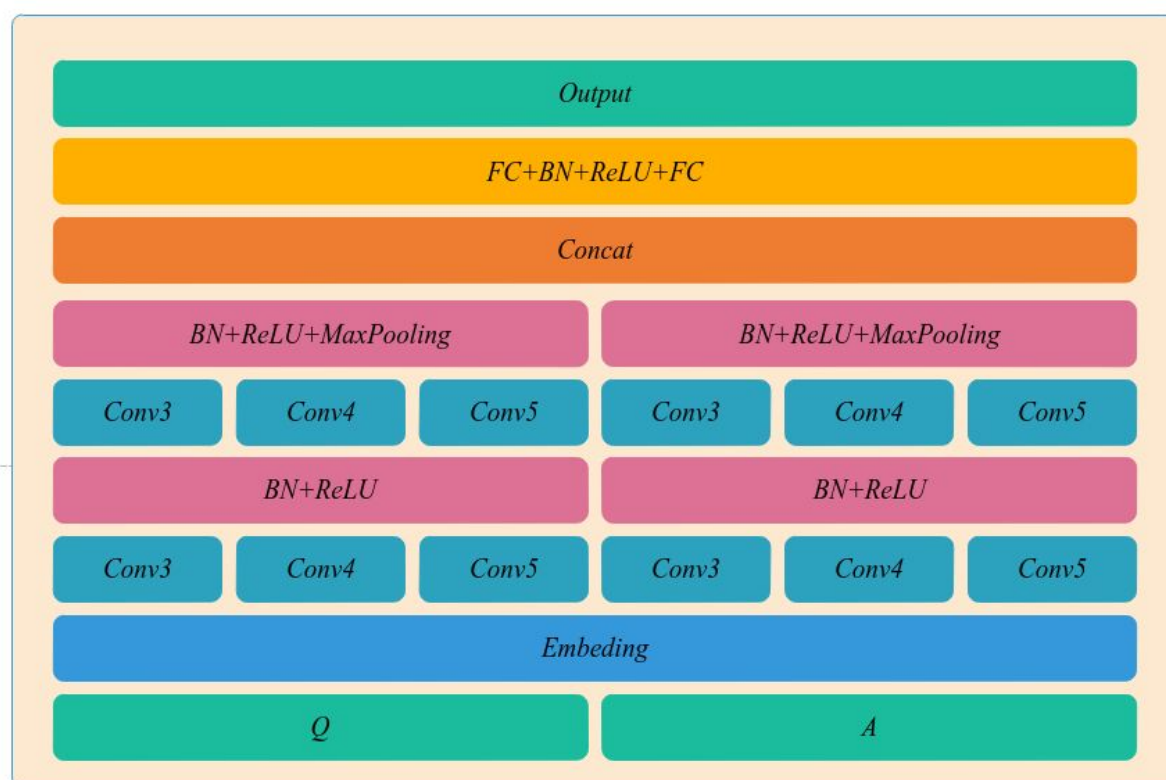
③全连接层 (Fully Connected Layer)

在整个卷积神经网络中起到“分类器”的作用。如果说卷积层、池化层等操作是将原始数据映射到隐层特征空间的话，全连接层则起到将学到的“分布式特征表示”映射到样本标记空间的作用。

④激活函数 (Activation Function)

神经网络用于实现复杂的函数，非线性激活函数可以使神经网络随意逼近复杂函数。没有激活函数带来的非线性，多层神经网络和单层无异，甚至无法实现异或门等简单函数。常见的激活函数有：Sigmoid、Tanh、ReLU等。

>>> **模型架构**：词嵌入后，分别对问题和回答进行两次卷积核大小为3、4、5的卷积操作，经过最大池化层后，将池化的向量连接起来。使用 ReLU 激活函数，防止反向传播过程中的梯度问题（梯度消失和梯度爆炸）；使用 Batch Normalization 批规范化，加速收敛。



1. 3 基于 CNN 模型 (改进版 - with Word2Vec)

我们目前使用的词嵌入是作为深度学习模型的一部分，词嵌入与模型本身一起学习。嵌入层使用随机权重初始化，并将学习所有数据集中的词嵌入。然而嵌入层是一个灵活的层，可以加载预训练的词嵌入模型，实现迁移学习。

在自然语言处理中使用预训练的词向量作为特征是非常有效的，预训练的词向量引入了外部语义信息，往往对模型很有帮助。

Word2Vec，为一群用来产生词向量的相关模型。这些模型为浅而双层的神经网络，用来训练以重新建构语言学之词文本。网络以词表现，并且需猜测相邻位置的输入词，在Word2Vec中词袋模型假设下，词的顺序是不重要的。训练完成之后，Word2Vec 模型可用来映射每个词到一个向量，可用来表示词对词之间的关系。该向量为神经网络之隐藏层。

Word2Vec 采用 CBOW 和 Skip-Gram 来建立神经网络词嵌入。CBOW 是已知当前词的上下文，预测当前词。而 Skip-Gram 相反，是在已知当前词，预测当前词的上下文。

实验使用维基百科中文语料生成 Word2Vec 模型，最终可以从 1.5+G 的原始语料中，提取到 900+M 的词向量模型，将近 80 万条词向量，每条词向量 100 维。

大致流程如下：

① 下载语料

<https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2>

② 提取正文

将 xml 格式的 wiki 数据转换为 text 格式。

③ 繁简转换

如果抽取中文的话需要将繁体转化为简体(维基百科的中文数据是繁简混杂的，里面包含大陆简体、台湾繁体、港澳繁体等多种不同的数据)。可以使用opencc进行转换，也可以使用其它繁简转换工具。

④ 编码转换

由于后续的分词需要使用utf-8格式的字符，而上述简体字中可能存在非utf-8的字符集，避免在分词时候进行到一半而出现错误，因此先进行字符格式转换。使用iconv命令将文件转换成utf-8编码。

⑤ 分词处理

使用 jieba 分词工具。

⑥ 训练 Word2Vec

```
In [ ]: from gensim.models import word2vec
import logging

input = './wiki.zh.text.jian.utf8.seg'
output = './wiki.vector'
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
sentences = word2vec.LineSentence(input)
model = word2vec.Word2Vec(sentences, size=100, window=5, min_count=5, workers=4)
model.wv.save_word2vec_format(output, binary=False)
```

⑦ 测试 Word2Vec

```
In [1]: from gensim.models.keyedvectors import KeyedVectors
model = KeyedVectors.load_word2vec_format('./wiki.vector', binary=False)
```

```
In [2]: # model['女人'] + model['国王'] - model['男人'] = model['皇后']
model.most_similar(positive=['woman', 'king'], negative=['man'])
```

```
Out[2]: [('queen', 0.7293198108673096),
('bride', 0.683803915977478),
('mistress', 0.6707652807235718),
('prince', 0.6648019552230835),
('wives', 0.6588137149810791),
('princess', 0.6529775857925415),
('queens', 0.6459839344024658),
('daughters', 0.6443694829940796),
('mother', 0.6377485990524292),
('godmother', 0.6289682388305664)]
```

```
In [3]: model.similarity('woman', 'man')
```

```
Out[3]: 0.6361447854063201
```

```
In [4]: model.similarity('queen', 'king')
```

```
Out[4]: 0.6681771014772736
```

可以看到，在 Word2Vec 词向量模型中，存在一些有趣的现象：

- ① “女人” + “国王” - “男人” \approx “皇后”。
- ② “女人”和“男人”（或“皇后”和“国王”）在空间向量上接近。

也成功验证了“Word2Vec 模型可用来映射每个词到一个向量，可用来表示词对词之间的关系”这一结论。

>>> **模型架构与 CNN 一致。**

1. 4 基于 Bi-LSTM 模型

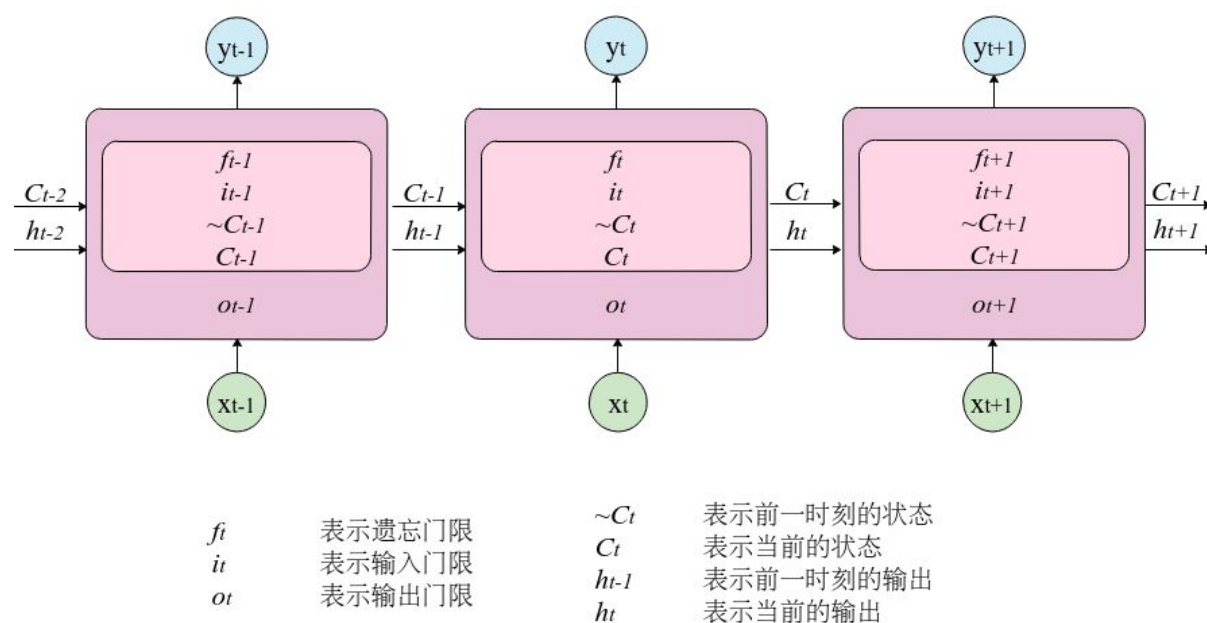
近两年深度学习在自然语言处理领域取得了非常好的效果。深度学习模型可以直接进行端到端的训练，而无须进行传统的特征工程过程。在自然语言处理方面，主要的深度学习模型是 RNN，以及在 RNN 之上扩展出来的 LSTM。

LSTM 是一种带有选择性记忆功能的 RNN，它可以有效地解决长时间依赖问题，并能学习到之前的关键信息。它增加了一条状态线，以记住从之前的输入学到的信息，另外增加三个门(gate)来控制该状态，分别为忘记门、输入门和输出门：

忘记门的作用是选择性地将之前不重要的信息丢掉，以便存储新信息。

输入门的作用是根据当前输入学习到新信息，更新当前状态。

输出门的作用是根据当前输入和当前状态得到一个输出，该输出除了作为基本的输出外，还会作为下一个时刻的输入。



单向 LSTM，根据前面的信息推出后面的信息，但有时候只看前面的信息是不够的。例如：今天天气__，风刮在脸上仿佛刀割一样。

如果根据“天气”，可能推出“晴朗”、“暖和”、“寒冷”等。但是如果加上后面的形容，能选择的范围就变小了，“晴朗”、“暖和”不可能选，而“寒冷”被选择的概率更大。

LSTM 虽然解决了长期依赖问题，但是无法利用文本的下文信息。双向 LSTM 同时考虑文本的上下文信息，将时序相反的两个 LSTM 网络连接到同一个输出。前向 LSTM 可以获取输入序列的上文信息（历史数据），后向 LSTM 可以获取输入序列的下文信息（未来数据）。两个方向有各自的隐藏层，相互之间没有直接连接，只是最后一起连接到输出节点上，模型准确率得到大大提升。

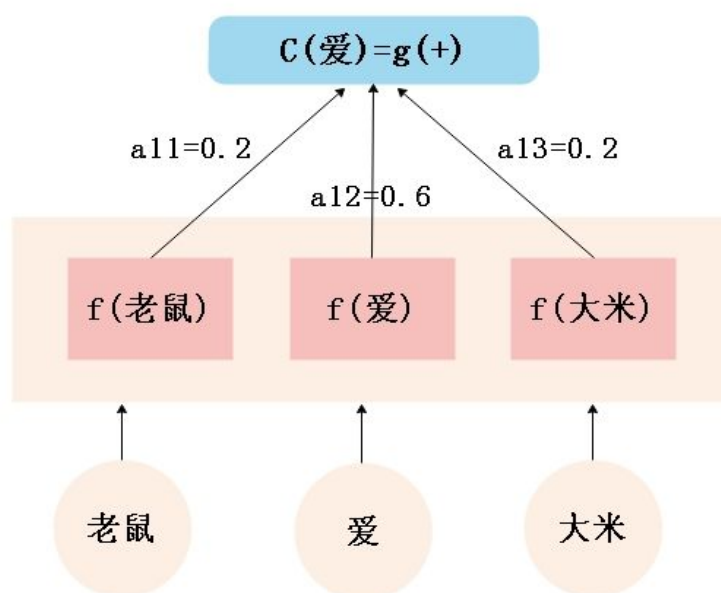
>>> 模型架构：与 FastText 大致相同，见附录。

1. 5 基于 Attention 模型

基于深度学习的 NLP 研究方法，基本上都是先将句子分词，然后每个词转化为对应的词向量序列。第一个思路是 RNN 层，第二个思路是 CNN 层，而 Google 的大作《Attention Is All You Need》提供了第三个思路：Attention，注意力机制！

注意力机制来源于人脑，我们在阅读的时候，注意力通常不会平均分配在文本中的每个词。如果直接将每个时刻的输出向量相加再平均，就等于认为每个输入词对于文本表示的贡献是相等的。但实际情况往往不是这样，比如在情感分析中，文本中地名、人名这些词应该占有更小的权重，而情感类词汇应该享有更大的权重。所以在合并这些输出向量时，希望可以将注意力集中在那些对当前任务更重要的向量上。也就是给他们都分配一个权值，将所有的输出向量加权平均。

注意力机制的思路是：原先都是相同的中间语义表示 C 会替换成根据当前生成单词而不断变化的 C_i ，每个 C_i 对应着不同单词的注意力分配概率分布。 f 函数代表 Encoder 对单词的某种变换函数， g 函数代表 Encoder 根据单词的中间表示合成整个句子中间语义表示的变换函数，一般来说， g 函数是 f 函数加权求和。



第一个过程是根据 Query 和 Key 计算权重系数，第二个过程根据权重系数对 Value 进行加权求和。

>>> 模型架构：与 FastText 大致相同，见附录。

1. 6 其他尝试

①用“字符级别”替换“单词级别”，最终效果不佳，原因可能是字符级别的粒度太细，无法提取到有效的特征。

②去除数据中的停用词，例如“的”、“啊”、标点符号等，最终效果不佳，原因可能是大部分问题和回答的长度较短，去除停用词之后，可以提取的特征更少了。

四、模型评估

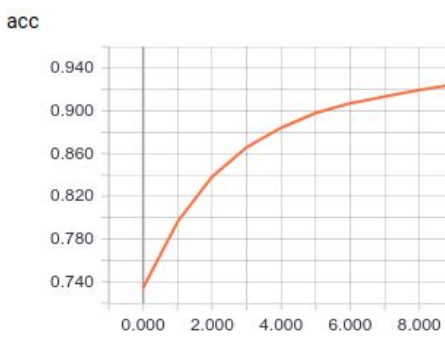
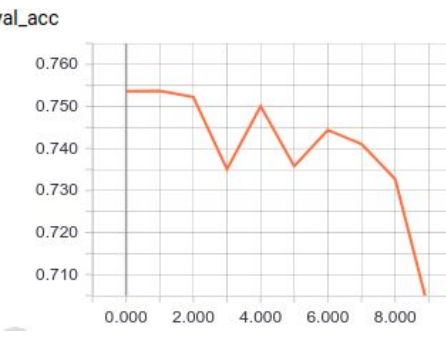
1. 1 训练效率

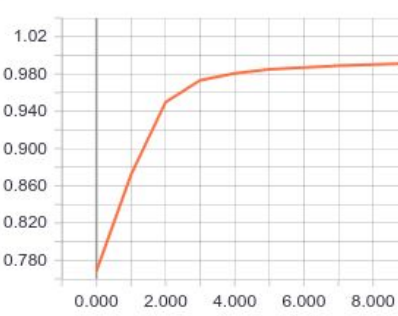
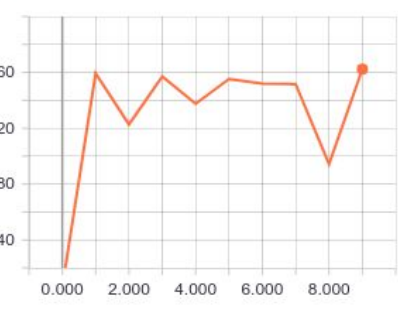
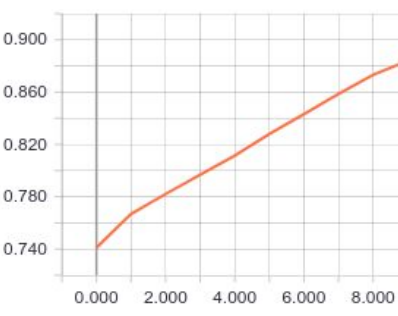
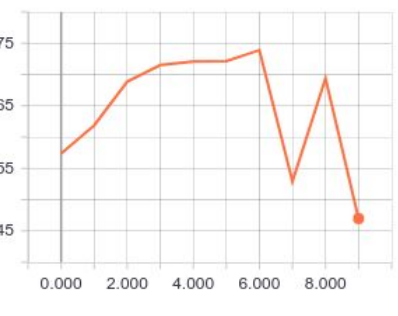
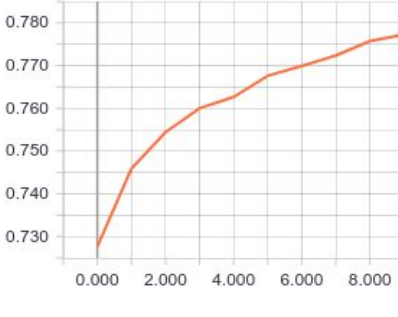
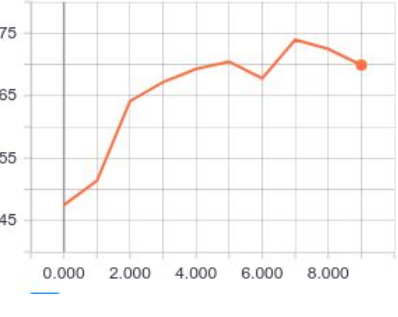
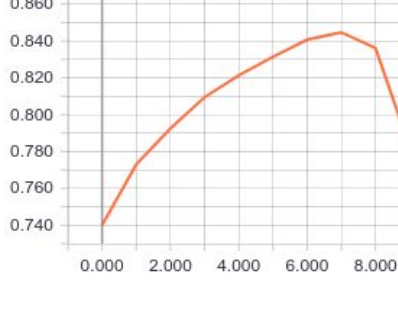
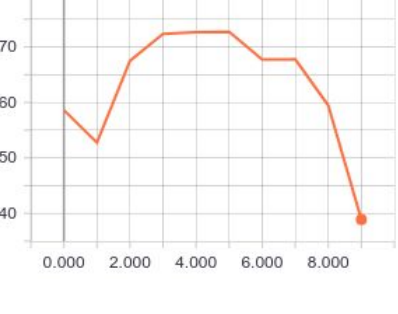
模型	FastText	CNN	CNN with Word2Vec	Bi-LSTM	Attention
时间/s	94	300	455	2888	3889

使用 4 个 1080Ti GPU 训练，可以看到：FastText 确实有利于高速训练，CNN 在训练上也有优势。而 Bi-LSTM 由于需要维持过去和未来所有时间步的隐藏状态，所以训练起来比较慢。Attention 模型基于 Bi-LSTM 模型，在训练过程中还要计算单词注意力权重，最后通过加权得到一个表示问题或回答的向量，进一步增加了训练时间。

1. 2 训练效果

统一在 Epoch 为 10 的条件下训练模型，得到的效果为：

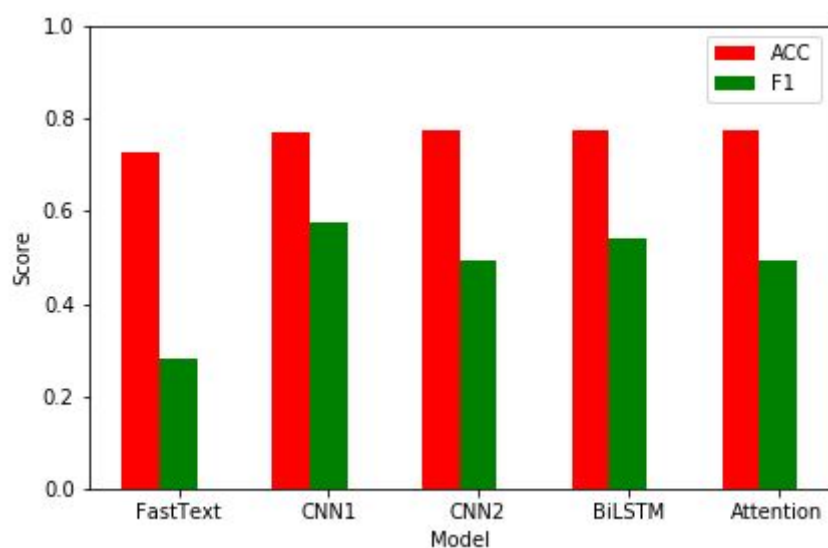
	训练 准确率 (Train accuracy)	验证 准确率 (Validation accuracy)
FastText		

CNN	<p>acc</p> 	<p>val_acc</p> 
CNN with Word2Vec	<p>acc</p> 	<p>val_acc</p> 
Bi-LSTM	<p>acc</p> 	<p>val_acc</p> 
Attention	<p>acc</p> 	<p>val_acc</p> 

1. 3 Accuracy, F1-score

$$\text{Accuracy} = \frac{\text{“预测正确”的样本数}}{\text{总样本数}}$$

$$\text{F1-score} = \frac{\text{“预测标签为 1 且真实标签也为 1”的样本数} \times 2}{\text{“预测标签为 1”的样本数} + \text{“真实标签为 1”的样本数}}$$



FastText 由于本身模型的简单性，所以表现较差。而经过精心设计的几种模型都获得较高的准确率和 F1 分数，同时在测试集上表现良好。

1. 4 其他评估指标

① **MRR**(Mean Reciprocal Rank) 平均倒数排名

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$

MRR 是一个国际上通用的对搜索算法进行评价的机制，即第一个结果匹配，分数为 1，第二个匹配分数为 0.5，第n个匹配分数为 1/n，如果没有匹配的句子分数为 0。最终的分数为所有得分之和。

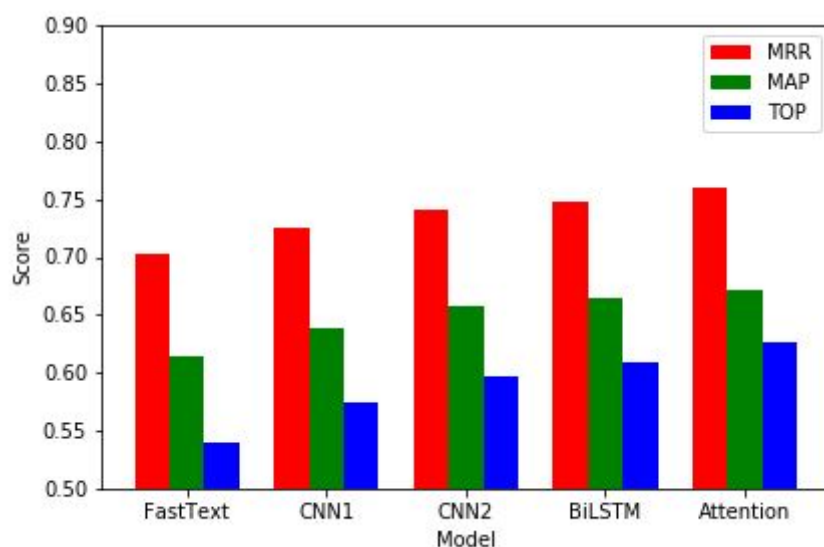
② **MAP**(Mean Average Precision) 平均精度均值

$$\text{MAP} = \frac{\sum_{q=1}^Q \text{AveP}(q)}{Q}$$

一组查询的平均准确率是每个查询的平均精度分数的平均值。MAP 是反映系统在全部相关文档上性能的单值指标。

③ TOP-1

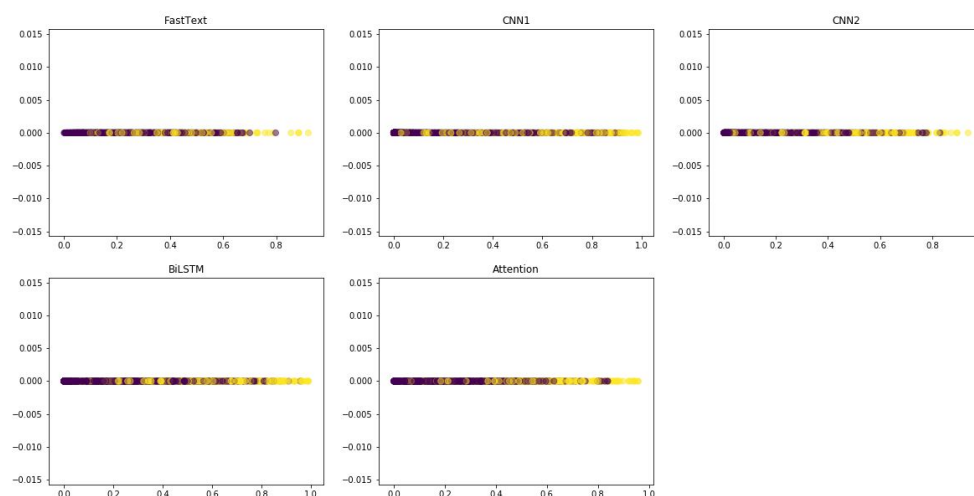
对于输出的未知答案，如果概率最大的是正确答案，认为预测正确。类似的还有 TOP-5，即对于输出的未知答案，如果概率前五个最大的是正确答案，认为预测正确。不过对于问答系统，往往第一个回答才是最重要的，因此我们选择 TOP-1 作为模型的评价指标之一。



除了 FastText 模型，其余四种模型的指标大致相同，平均 MRR $\approx 75\%$ ，MAP $\approx 65\%$ ，TOP-1 $\approx 62\%$ 。能够达到实现智能阅读系统的水平，让用户“更自然”、“更低成本”地实现人与机器的交流。

1. 5 概率分布图

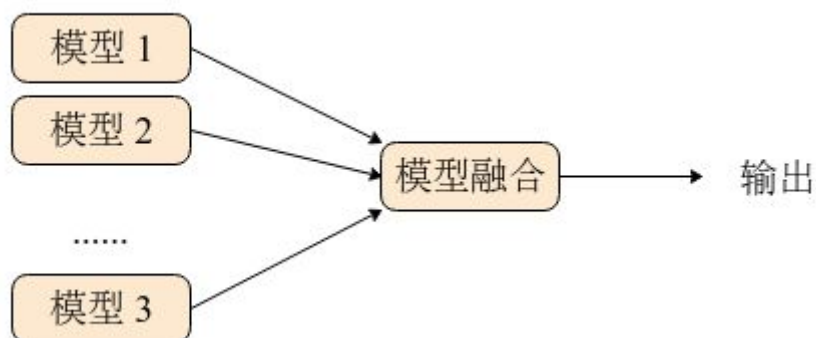
下图为在测试数据集中，真实答案和预测答案的概率分布图（其中紫色为正确回答，黄色为错误回答，横坐标为预测概率）：



概率分布图符合预期结果，接近 0 的基本为错误回答，接近 1 的基本为正确回答。

五、模型融合

使用单一模型的风险是很大的。首先，模型可能对数据集存在片面性，不能考虑到所有影响因素。其次，模型可能对数据集存在依赖性，容易造成过拟合。如果采用集成学习，通过将多个学习器进行结合，通常可以获得比单一学习器显著优越的泛化性能。



1. 1 理论计算

假设存在三个模型（正确率均为 70%），以“投票集成”模型为例，简单概率运算：

全部正确	$0.7 * 0.7 * 0.7 = 0.3429$
------	----------------------------

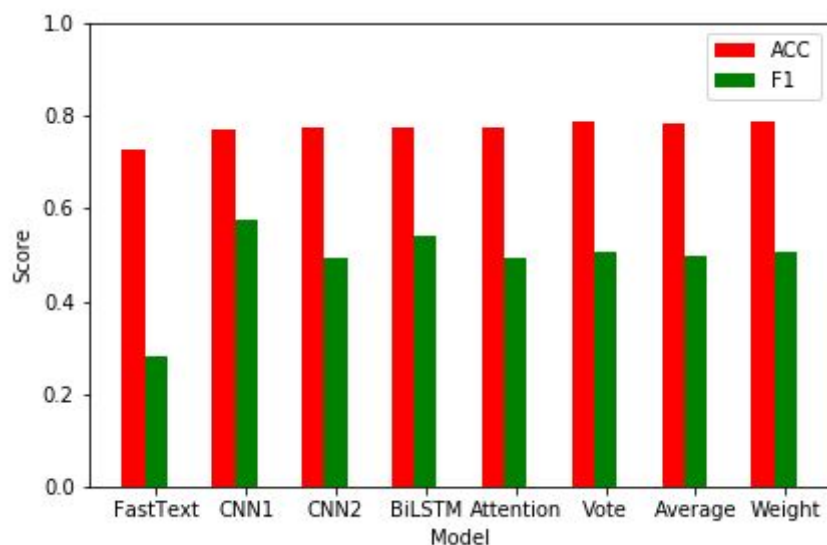
两个正确	$0.7 * 0.7 * 0.3 + 0.7 * 0.3 * 0.7 + 0.3 * 0.7 * 0.7 = 0.4409$
一个正确	$0.3 * 0.3 * 0.7 + 0.3 * 0.7 * 0.3 + 0.7 * 0.3 * 0.3 = 0.189$
全部错误	$0.3 * 0.3 * 0.3 = 0.027$

理论正确率 = $0.3429 + 0.4409 = 0.7838 \approx 78\%$ ，说明集成学习理论上确实有效。

1. 2 实际测试

方案	投票	平均(1:1:1:1:1)	权重 (1:2:2:2:3)
说明	即少数服从多数原则，分类得票数超过一半的作为预测结果。	将所有预测结果相加取平均值。	将所有预测结果按照权重计算。

经过测试，三种模型融合方案均能提高大约 1% 的准确率，以及保持较高的 F1 分数，说明集成学习实际上确实有效。



六、泛化能力

1. 1 定义

泛化能力是指机器学习算法对新鲜样本的适应能力。学习的目的是学到隐含在数据对背后的规律，对具有同一规律的学习集以外的数据，经过训练的网络也能给出合适的输出，该能力称为泛化能力。

1. 2 性质

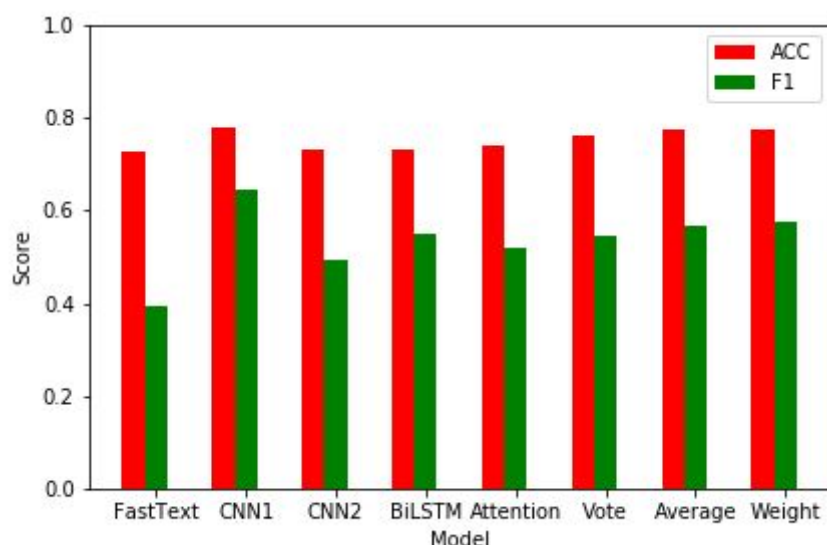
通常期望经训练样本训练的网络具有较强的泛化能力，也就是对新输入给出合理响应的能力。应当指出并非训练的次数越多越能得到正确的输入输出映射关系。网络的性能主要用它的泛化能力来衡量。

1. 3 实验

使用百度开源数据集 WebQA (<https://spaces.ac.cn/archives/4338>)，该数据集含有 44 万条问答记录。实验前需要将 WebQA 数据集转换为赛题数据集格式：answer 对应赛题数据集的 label，其中 no_answer 代表 0，其余回答代表 1。

1. 4 评估

① Accuracy、F1-score

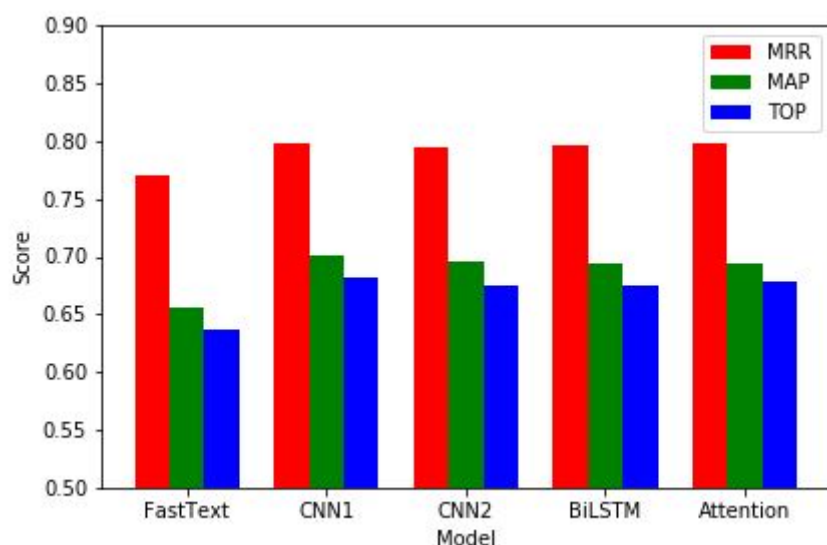


单个模型泛化效果最好的是 CNN without Word2Vec，准确率和 F1 分数甚至高于平均水平大约 5%。而其他设计复杂的模型，泛化效果反而降低了，原因可能是：

- 1、数据集数量较少，过拟合严重。
- 2、数据集质量较低，由于 WebQA 数据集来源于百度知道等问答社区，社区存在大量复制粘贴现象，以及牛头不对马嘴的回答现象。
- 3、数据集标记中存在一定的噪音，可能打标的时候工作人员自身水平有限，导致数据错误分类现象。

不过在进行了模型融合之后，还是能够在大数据下继续保持高准确率和 F1 分数，表明我们的模型泛化能力良好，设计合理。

② MRR、MAP、TOP-1



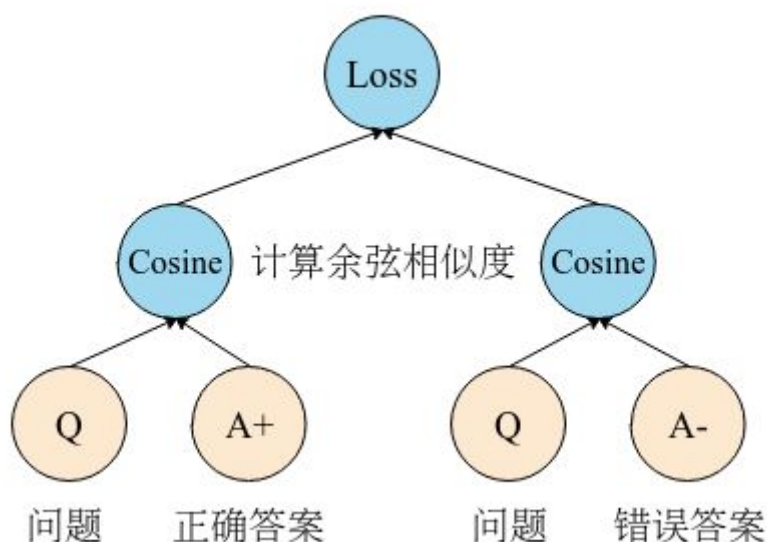
除了 FastText 模型，其余四种模型的指标大致相同，平均 MRR $\approx 80\%$ ，MAP $\approx 70\%$ ，TOP-1 $\approx 68\%$ 。能够达到实现智能阅读系统的水平，让用户“更自然”、“更低成本”地实现人与机器的交流。

七、未来工作

1. 改进损失函数——Triplet Loss

深度学习模型有固定的输入和输出，并且损失函数为真实值与预测值的某种损失函数，然而有些模型并非如此，比如 Triplet Loss。

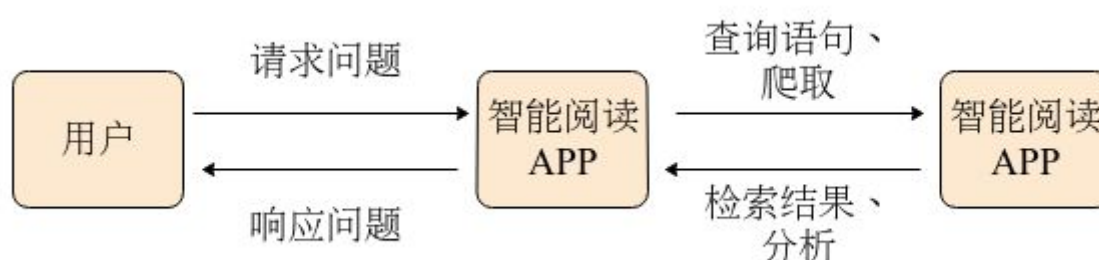
$$Loss = \max(0, m + \text{Cos}(Q, A+) - \text{Cos}(Q, A-))$$



(Q, A_+) 为一对正样本对, (Q, A_-) 为一对负样本对。Loss 计算公式中的 m 是一个大于零的正数, 我们希望正样本分数越高越好, 负样本分数越低越好, 但二者得分之差最多到 m 就足够了, 差距增大并不会有任何奖励。使用 Triplet Loss 有助于细节区分, 在众多候选回答中更容易找出正确回答。

1. 2 基于Web的问答系统

基于 Web 的问答系统 (Web-based Question Answering, WQA) 以开放的互联网上的 Web 文档作为问答系统的知识来源, 从搜索引擎返回的相关网页片段中抽取用户所提问题的答案。



WQA 系统同时具有搜索引擎和问答系统的优点:

- ① 能通过现有成熟的搜索引擎来获取整个互联网上的各种相关信息, 这些信息无所不包, 不受领域的限制, 且与时俱进, 不断更新。
- ② 一般能够较好地处理和回答事实型问题。
- ③ 能够利用自然语言进行人性化的交互。

作为一种特殊的问答系统, WQA 从搜索引擎返回的搜索结果中抽取用户所需的答案。WQA 可以单独使用, 也可以与其他问答系统一起混合使用。对 WQA 进行深入研究能够促进问答系统的进一步发展。

1. 3 数据众包服务

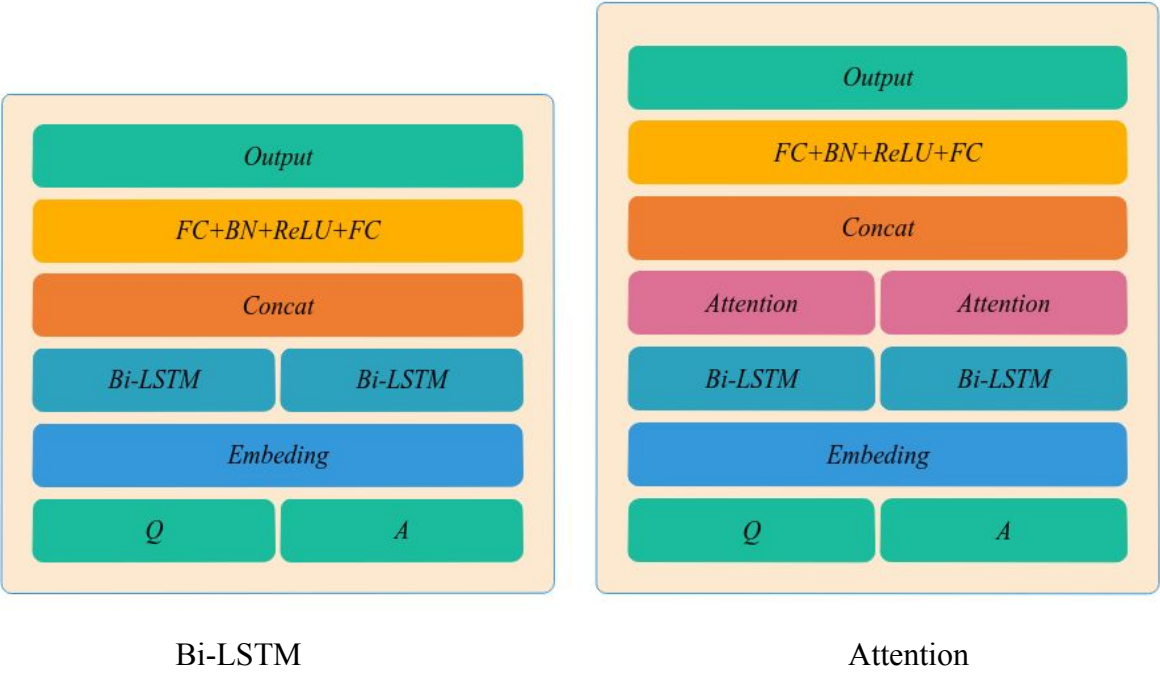
数据众包服务使用低成本高效率的众包模式满足客户对数据的需求, 可采集到大量的原始数据, 通过数据标注对原始数据进行加工, 最终提供计算机可以识别的高质量数据, 帮助数据科学家更精准地训练算法模型、开展机器学习工作, 提高在 AI 领域的竞争力。

附：

开发平台

操作系统	Ubuntu 16.04
内存容量	128 G
硬盘容量	8 T 固态硬盘
显卡型号	GTX 1080Ti * 4
开发框架	Keras / Tensorflow (Python)

模型架构



参考文献

.....