COMP130141.01 自然语言处理

让 AI 做英语单选题———基于深度学习语言模型的尝试

计算机科学技术学院

刘婧源 15307130385

2018 年 1 月 21 日

目　录

让 AI 做英语单选题———基于深度学习语言模型的尝试

刘婧源 15307130385

Abstract

英语考试中，单选题是最为常见的题型，需要在多个选项中选出最符合语法规则和上下文语境的正确答案。本项目通过深度学习方法，训练自然语言模型，自动完成英语单选题。在项目实现过程中，我先编写爬虫，从网站爬取英语单选题及答案共计 60,962 道，其中小学题目 14,104 道，初中 27,164 道，高中 19,694 道；应用 state-of-the-art 的深度学习方法，训练了单词级别 (word-level) 的语言模型；并设计了基于语言模型做题的算法。为了选出合适的网络结构以及最佳的参数，我做了大量探索性实验，例如：比较不同模型容量的影响，比较不同的 RNN 结构，引入 nltk 的外部语料库和微调 (fine-tune) 模型等。最终，训练得到的最好模型在小学、中学、高中的题目上分别达到了 80.30%、70.67%、65.99% 的准确率。项目代码运行方法及数据集见附录部分。

1. 数据搜集

想要训练出靠谱的人工智能模型，数据是不可或缺的，也是至关重要的基础。由于找不到任何关于英语单选题的公开数据集，所以我的第一个任务就是自己建立一个这样的数据集。

1.1. 数据来源网站

在网上，有很多提供英语试题的网站，但这些网站大多提供的是 word 文档，题目和答案放在文档的不同位置，这使得用程序提取数据变得非常困难。幸运的是，经过一番搜索，我找到了学而思的官方网站 (http://tiku.xueersi.com/)，如图 1. 所示，在学而思的英语题库中，已经分小学、初中、和高中题型整理好了试题，点开题目详情页面就可以获取答案。

图 1. 学而思英语网站

```
{
    "_id": "xrs_594975", // 题目的id
    "question": "<BLANK> to meet you!", // 题干，<BLANK>表示需填的空白
    "choices": [  //选项
        ["OK"],
        ["Nice"],
        ["Sure"],
        ["Can"]
    ],
    // primary、middle、high分别表示小学、初中和高中
    "difficulty": "primary",
    "answer": 1, // 答案的index，从0开始
    "blank_num": 1,  // 题目中空白的数理
}
```

图 2. 表示单个试题的 Json 串.Json 可以以结构化的方式表示数据，Json 串中，包含了训练模型所需的试题、选项、答案等数据

1.2. 爬虫爬取数据

接下来的问题就相对简单啦：编写一个爬虫将学而思的英语单选题库爬取下来。用 Python 作为编写爬虫的语言，使用如图 2. 所示的 Json 串来表示单个选择题数据。单个 Json 串包含了题干、选项、答案在内的所有信息，处理起来非常方便，此外也可以使用方便保存 Json 格式的 MongoDB 数据库作为后端。

在爬取数据的过程中，遇到了以下困难：

- 单线程爬取太慢。
  使用 Python 的 Queue 编写了一个多线程爬虫解决了这个问题。在实际爬取时，设置了线程数 =10。

- 处理的网页格式不统一，抽取数据时经常失败。
  针对这个问题，可以多使用正则表达式，尽可能提高成功率。例如，单选题中经常会出现表示需要填空的下划线，在 HTML 中，既可以用“<u>< /u>”标签来表示下划线，也可以直接使用字符来表示，对此，可以用正则表达式“<u>.*?</u>|_+”来找出题干中需要填空的位置。
- 在请求过程中经常发生网络错误。
  对此，在每次请求中间加入 0.5 秒的延时。当发生网络错误（即返回状态码不为 200）时，让程序暂停 1 分钟再重新爬取。

经过一番努力，最终爬取到了 60,962 英语单选题，其中小学题目 14,104 道，初中 27,164 道，高中 19,694 道。数据充分，对下面的模型建立打下了很好的基础。项目所建立和使用的数据集分享至百度网盘：https://pan.baidu.com/s/1i5Vj1yt，密码：bsv8，见附录6.4。

2. 方法

2.1. n-gram 语言模型的缺陷

先来考虑一个非常简单的英语题：There are some _____. 选项为 apples 和 apple。很显然，根据我们的英语知识，横线上应该填单词 apples 而不是 apple。如果用数学语言表达就是：“当句子里出现 There are some 时，下一个单词是 apples 的概率要比 apple 的概率要大。”设单词依次为 w_1, w_2, w_3, w_4，此时，$w_1 = There, w_2 = are, w_3 = some$，那么有：

$$P(w_4 = apples|w_1, w_2, w_3) > P(w_4 = apple|w_1, w_2, w_3)$$

在自然语言处理领域，可以用语言模型来估算概率 $P(w_4|w_1, w_2, w_3)$。语言模型实际上是关于语言中各个单词的联合分布概率 $P(w_1, w_2, w_3, ...)$。例如在上述问题中，如果我们知道了 $P(w_1, w_2, w_3, w_4)$，就可以根据 $P(w_4|w_1, w_2, w_3) = \frac{P(w_1,w_2,w_3,w_4)}{P(w_1,w_2,w_3)}$ 计算出 w_4 的条件概率，由此得到正确的选项。

在课上曾经学过建立语言模型的经典算法 n-gram。在概率表达式 $P(w_1, w_2, ..., w_n)$ 中，根据链式规则可得

$$P(w_1, w_2, ..., w_n) = P(w_1)P(w_2|w_1)...P(w_m|w_1, ..., w_n)$$

这个概率不易计算，不妨利用马尔科夫模型的假设：

- 有 n 个状态，且存在一个离散的时间序列
- 在每个时刻系统只能处于唯一的一个状态，当前状态只与前面相邻的 n 个状态有关（无后效性）
  即 $P(w_i|w_1, ..., w_{i-1}) = P(w_i|w_{i-n+1}, ..., w_{i-1})$

特别地，n=2 时，得到二元模型（bigram model）：

$$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

可以用该模型解决上述英语单选题：当前一个单词是 some 时，很显然后一个单词是 apples 的概率比 apple 的概率要大，即 $P(w4 = apples|w_3 = some) > P(w_4 = apple|w_3 = some)$

n-gram 模型的问题在于，如果 n 取的太大，往往会由于数据少而无法估计概率，这使得模型很难捕捉到长程上的语言规律。由于在英语单选题中，经常需要用到长程上的依赖，例如试题：_____ is the nearest park?(选项为 What 和 Where)，做题者需要通过”park” 这个单词来判断横线处应该填什么，这需要用到非常大的语料库，建立一个 n=5 的 n-gram 模型。更不用说在很多题目存在远大于 n=5 的依赖。因此，在实践中，我放弃了 n-gram 模型，转而使用更合理有效的深度学习语言模型。

2.2. 深度学习语言模型

目前，深度学习是建立语言模型的 state-of-the-art 的算法 [8]。相比 n-gram 方法，深度 RNN 网络 [4]表达能力更加强大，还克服了 n-gram 方法无法捕捉长程依赖的缺陷。

一个简单的 RNN 网络如图 3. 所示。$x_1, x_2, x_3, ...$ 是输入的序列型数据；$h_0, h_1, h_2, h_3, ...$ 表示隐层状态，计算公式为

$$h_t = f(Wh_{t-1} + Ux_t + b)$$

其中 W, V, b 是模型需要学习的参数，f 为激活函数，在 RNN 中常用的激活函数有 Tanh 函数、ReLU 函数等。

从隐含层状态到输出，还需要经过一次变换，假设要处理的是一个分类问题，即输出的 y_t 表示各个类别的概率，那么

$$y_t = softmax(Vx_t + c)$$

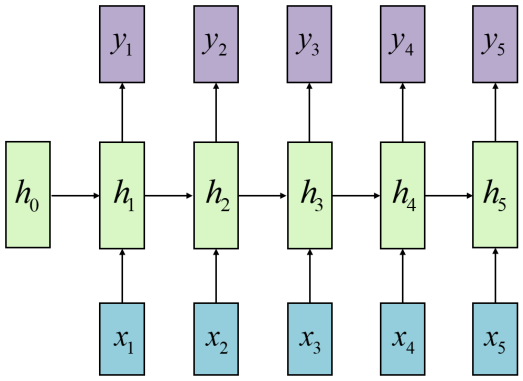


图 3. 经典的 RNN 结构

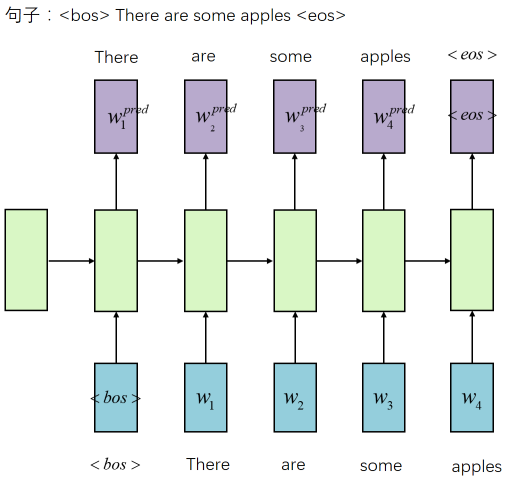


图 4. 使用 RNN 建立语言模型

其中 V, c 是需要学习的参数。

除了 RNN，还有两种常用的计算单元：LSTM[2]和 GRU[1]。和 RNN 一样，LSTM 和 GRU 同样由输入、隐含状态、输出三部分组成，只是在计算隐含状态时采用了更加复杂的计算公式。在这篇文章中所有使用 RNN 的地方都可以用 LSTM 和 GRU 代替，所以在下面的描述中将不再明确区分这三种计算单元。在3.3中会对它们的性能进行对比。

使用 RNN（或 LSTM、GRU，下同）建立语言模型的方法是：使用前 t 个单词来预测 t+1 个单词。一般地，设一个句子中的单词依次为 $w_1, w_2, ..., w_n$，通常在句子开头加上开始标记 <bos>，句子末尾加上结束标记 <eos>。RNN 的输入就依次是 <bos>, $w_1, w_2, ..., w_n$，输出的目标就是 $w_1, w_2, ..., w_n$, <eos>，如图 4. 所示。第一个输出 w_1^{pred} 实际上就对应概率 $P(w_1|$ <bos>$)$，第二个输出 w_2^{pred} 就对应 $P(w_2|$<bos>$, w_1)$，接下来依次是 $P(w_3|$<bos>$, w_1, w_2)$，$P(w_4|$<bos>$, w_1, w_2, w_3)$，以此类推。将所有这些概率乘起来就得到了需要的语言模型：$P(w_1, w_2, w_3, ..., w_n|$<bos>$)$[7]。

2.3. 应用深度学习语言模型做题

训练好语言模型后，就可以用它来解题了。还是举之前的例子：There are some _____. 选项为 apple 和 apples。如图 5. 所示，在深度学习语言模型中，依次输入 <bos>、There、are、some，此时得到的输出就是关于下一个单词是什么的概率分布。可以找到概率 $P(w_4 = apples|w_1, w_2, w_3)$ 和概率 $P(w_4 = apple|w_1, w_2, w_3)$ 并根据这两个概率做出判断。

至此，只讨论了最简单的选项只有一个单词、且选项依赖前面语境的情况，还有两个重要的问题没有解决：

- 单个选项中有多个单词。此时各个选项中单词的数目还有可能不同。
- 选项依赖后面的语境。如题目：_____ is the nearest park?(选项为 What 和 Where)

如何解决这两个问题？其实，可以转换一下思路，将各个选项填入句子中，应用深度学习语言模型来判断每个句子的合理性。

设一个句子是 $w_1, w_2, w_3, ..., w_n$，它的合理性就可以用概率 $P(w_1, w_2, ..., w_n|$<bos>$)$ 来表示，其值越高，说明该句子越合理。这种做法其实也比较符合我们日常的语言习惯，即最符合语感（被最多数曾经出现过的搭配所认可）的句子是正确的。比如说，将上述问题中的 What 和 Where 填入空白，分别形成句子 “What is the nearest park?” 和 “Where is the nearest park?”，通过计算句子整体的概率，可以判断出 “Where is the nearest park?” 更加合理，从而得到正确的选项。

对于选项中有多个单词的情况可以使用相同的方法来处理。但直接使用概率 $P(w_1, w_2, ..., w_n|$<bos>$)$ 会导致模型更偏好长度较短的答案。原因在于：这个概率是应用语言模型连乘得到的，每次的乘数都是一个小于 1 的数，句子长度越长，值就会越小。为了修正长度带来的影响，可以取几何平均来作为每个句子合理

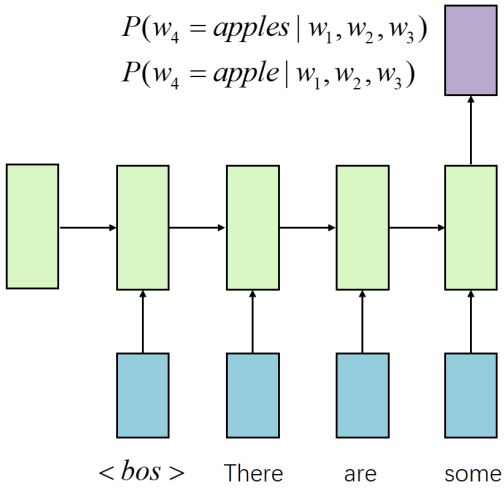


图 5. 使用深度学习语言模型预测答案

性的打分，即

$$Score = \sqrt[n]{P(w_1, w_2, ..., w_n| < bos >)}$$

举一个实际的例子，题目：I love ____ books. 选项为：A. reading B. to reading C. talking。将 A 选项带入句子中得到"I love reading books"，利用深度学习语言模型分别计算 $p_1 = P(I|<\text{bos}>)$,$p_2 = P(love|<\text{bos}>, I)$，$p_3 = P(reading|<\text{bos}>, I, love)$，$p_4 = P(books|<\text{bos}>, I, love, reading)$，最后的打分就是 $\sqrt[4]{p_1p_2p_3p_4}$。对于 B、C 选项也做类似计算。在计算得到的结果中，可以发现 B 选项里的 $P(reading|<\text{bos}>, I, love, to)$ 会非常小，C 选项里的 $P(books|<\text{bos}>, I, love, talking)$ 会非常小，因为它们都不符合语言习惯。最终，A 选项的打分最高，应当选择 A。

2.4. 打分的近似计算

设 $p_1 = P(w_1| < bos >)$, $p_2 = P(w_2| < bos >, w_1)$,...,$p_n = P(w_n| < bos >, w_1, w_2, ..., w_{n-1})$，句子合理性的打分就是：

$$Score = \sqrt[n]{p_1p_2...p_n}$$

在实际计算中，有两个会造成数据精度溢出的情况：

- 由于 $p_1, p_2, ..., p_n$ 都是非常接近 0 的数，直接运算会导致精度不足，解决方法是计算它的对数值：

$$ln(Score) = \frac{ln(p_1) + ln(p_2) + ... + ln(p_n)}{n}$$

- 在计算 $ln(p)$ 时，需要计算一次 softmax 函数，设经过 softmax 函数之前的值为 $\alpha = (\alpha_1, \alpha_2, .., \alpha_m)$，$m$ 为类别数，p 对应其中第 i 个单词类别的概率，那么：

$$\begin{aligned} ln(p) &= \frac{e^{\alpha_i}}{e^{\alpha_1} + e^{\alpha_2} + e^{\alpha_3} + ... + e^{\alpha_m}} \\ &= \alpha_i - ln(e^{\alpha_1} + e^{\alpha_2} + e^{\alpha_3} + ... + e^{\alpha_m}) \end{aligned}$$

在实际计算中，我发现这里的 $\alpha_1, \alpha_2, .., \alpha_m$ 会比较大，可能取到 2000，3000 这样的数值，直接计算 e^{α_k} 会造成数值溢出。在查阅相关资料后，发现可以采取下面的近似公式：

$$ln(e^{\alpha_1}+e^{\alpha_2}+e^{\alpha_3}+...+e^{\alpha_m}) = max\{\alpha_1, \alpha_2, .., \alpha_m\}$$

解决溢出和精度的问题，并且大大降低了计算复杂度。
以上是计算打分的全过程。

3. 实验

3.1. 数据集和预处理

训练时，将所有题目的正确答案填写到空白上，作为语料库使用。对每个句子，做下面的预处理：

- 将所有字母变成小写（这可以减少总单词的数量，使模型更容易训练）
- 用 nltk 库中的 word_tokenize 函数将句子切分开
- 每个句子的开头添加 <bos>，结尾添加 <eos>。

为了验证模型性能，以 8:1:1 的比例将原始数据分割为训练集、验证集、测试集。最后，各个年级题库的句子数分别为：

- 小学的题目总计 14,104 道，训练集共 11,283 题，验证集共 1,410 题，测试集共 1,411 题。
- 初中的题目总计 27,164 道，训练集共 21,731 题，验证集共 2,716 题，测试集共 2,717 题。
- 高中的题目总计 19,694 道，训练集共 15,755 题，验证集共 1,969 题，测试集共 1,970 题。

3.2. 深度学习模型训练策略

在下面的所有实验中，为了公平地比较结果，采取统一的深度学习训练策略：初始学习率为 20，每训练完一个 epoch，就在验证集上验证当前语言模型的性能（以困惑度作为验证指标），如果模型性能较之前有提升，就把该模型保存下来。如果性能没有提升，说明该学习率上的训练已接近饱和，就将学习率降低为原来的 $\frac{1}{4}$ 继续训练。若模型连续 5 个 epoch 性能都没有任何提升，就退出训练，并把历史上在验证集上表现最好的模型作为最终的模型。使用这个模型在测试集上测试语言模型的性能，以及在测试集上做题的准确率。

3.3. 不同 RNN 计算单元对模型的影响

首先探索了不同 RNN 计算单元对模型的影响。实验的计算单元有：RNN（分别使用 Tanh 和 ReLU 作为激活函数）、GRU 和 LSTM。在训练时，都使用 2 层的结构，Dropout 概率 [6]设置为 0.5，word embedding 的维数和隐藏状态的维数都设置为 500。

训练的结果如表 1. 所示。从表中可以看出，最好的计算单元是 LSTM，在验证困惑度、测试困惑度、测试做题准确率上都全面优于 GRU 和 RNN。GRU 的性能不如 LSTM，但普遍优于 RNN。RNN 的效果非常差，当 RNN 的激活函数为 Tanh 时，模型可以训练，但是困惑度很高，准确率也很差；当 RNN 激活函数为 ReLU 时，在小学的数据集上表现还好，但在初中和高中的数据集上出现梯度爆炸的现象，完全无法训练。

在接下来的所有实验中，将会只考虑 LSTM 这一种结构。

3.4. 不同模型容量对模型的影响

模型中，有两个超参数比较关键：word embedding 空间的维数和 RNN 隐藏层的维数（将这两个超参数分别记为 n_emed 和 n_hidden），它们决定了模型的容量。这两个超参数越大，模型要训练的参数就越多，训练也就越慢，同时模型的表达能力就会越强。问题是：模型的容量会对训练过程以及最后的正确率产生怎么样的影响？多大的模型容量是最合适的？

根据论文 [5]和 [3]，在输入和输出单词时使用相同的 embedding 参数，不仅可以减少参数数量，还可以提高模型性能，但此时必须保持 n_emed=n_hidden。遵循论文的做法，在实验时，我使用 2 层的 LSTM，保持 Dropout 率为 0.5 不变，分别令 n_emed=n_hidden=200,300,500,1000,1500 训练小学、初中、高中的模型。

在训练过程中，各个模型在验证集上的困惑度随 epoch 变化的曲线如图 6. 所示。在训练小学的模型时，各个容量的模型收敛效果相似，可能是因为小学的问题比较简单。初中和高中的模型表现为：模型容量较小（如 200、300）时，模型的困惑度很难下降，只有当模型容量达到一定数值后（如 500 及以上），才能收敛到不错的结果。

最终的测试结果如表 2. 所示。从中可以看出：

- 如果只关注最后做题的准确率，小学和初中的题目最佳超参数是 n_emed=n_hidden=1000，而高中题目的最佳超参数是 n_emed=n_hidden=1500。可以理解为：年级越高时，对应的知识越多，就需要更大的模型容量。
- 一个有趣的现象是：在测试集上困惑度最低的模型，做题的准确率不一定是最高的。按理说，当困惑度越低时，模型预测下一个单词应当越准确。但这里往往存在一定的过拟合现象，例如模型可能倾向于预测一些较常出现的单词，尽管这样可以降低困惑度，但是却忽略了上下文之间的联系，导致做题准确率下降。

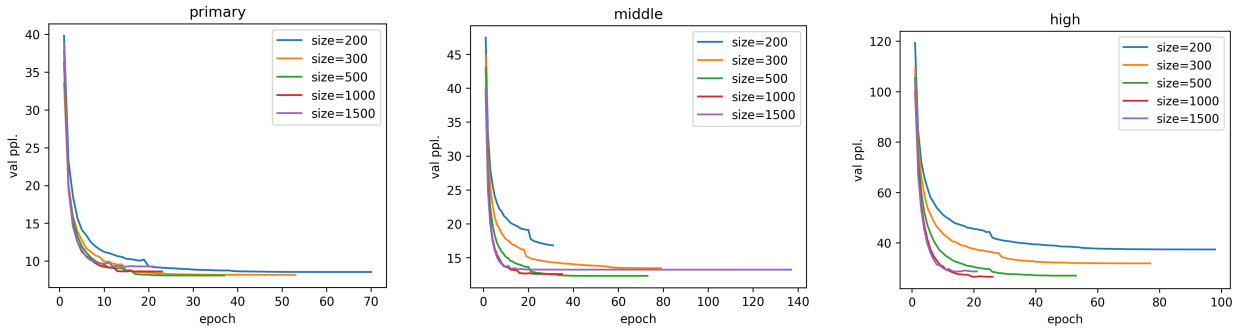
3.5. 不同 Dropout 概率对模型的影响

我在初中的数据集上探索了不同 Dropout 概率对模型的影响。使用双层 LSTM，保持模型容量 n_embed=n_hidden=500 不变，Dropout 概率分别设置为 0.1，0.3，0.5，0.6，0.7，0.8。训练时，验证集上困惑度的随 epoch 数的变化曲线如图 7. 所示，模型验证和测试的结果如表 3. 所示。

可以看出，总的来说，当 Dropout 概率越大时，模型收敛的速度越慢。Dropout 概率从 0.1 变到 0.8，最佳模型需要的 epoch 数依次为 31, 29, 67, 65, 69, 116。即，Dropout 概率越大，模型“越难训练”。从性能角度来看，还是要选择中间的 Dropout 概率才能得到最佳困惑度和测试准确率。当 Dropout 概率特别小的时候，模型很快就收敛了，此时可能还没有训练完全；当 Dropout 概率特别大时，模型不仅收敛慢，而且无法收敛到比较理想的极小值。在下面的实验中，统一取

	模型	验证集上的困惑度	测试集上的困惑度	测试集上的准确率
小学	RNN_Tanh	198.71	203.27	29.27%
	RNN_ReLU	11.15	11.16	67.26%
	GRU	8.96	8.87	75.76%
	LSTM	**8.09**	**8.03**	**78.31**%
初中	RNN_Tanh	296.27	291.98	26.54%
	RNN_ReLU	nan	nan	22.75%
	GRU	13.72	13.45	64.85%
	LSTM	**12.33**	**12.11**	**69.93**%
高中	RNN_Tanh	450.55	462.93	24.37%
	RNN_ReLU	nan	nan	25.23%
	GRU	31.31	31.43	58.38%
	LSTM	**26.99**	**27.04**	**62.79**%

表 1. 使用不同的 RNN 计算单元对结果的影响



(a) 验证集上的困惑度随 epoch 变化（小学）(b) 验证集上的困惑度随 epoch 变化（初中）(c) 验证集上的困惑度随 epoch 变化（高中）

图 6. 不同验证集上的困惑度随 epoch 的变化

Dropout 的概率为 0.5。

3.6. 跨数据集预测和微调

这里有一个问题是：可以跨数据集进行测试吗？例如：使用在小学的数据集上训练的模型去做中学的题目；或者反过来，用在中学数据集上训练的模型去做小学的题目，正确率会是多少呢？

我使用3.4中在测试集上表现最好的几个模型，进行交叉测试（小学、初中的选择 n_embed=n_hidden=1000 的模型，高中的选择 n_embed=n_hidden=1500 的模型），结果如表4所示。

首先，可以肯定的是，模型拥有一定的跨数据集泛化能力，例如初中的模型在小学的试题上也有 72.22% 的正确率。但模型的泛化能力并不是特别强，每个数据集上性能最好的模型依然是从这个数据集上训练出来的，这说明各个数据集都有一定的倾向性，只有在这个数据集上训练出来的模型才能获得最好的表现。

基于此，为了提高性能，可以考虑用微调 (fine-tune) 来训练更好的模型。以训练小学的模型为例，可以先在初中的数据集上训练，再以训练好的模型为起点，再在小学的数据集上重新训练。由于小学的词汇量远远少于初中的词汇量，如果以小学数据集上的模型

	n_embed/n_hidden	验证集上的困惑度	测试集上的困惑度	测试集上的准确率
小学	200	8.55	8.49	76.40%
	300	8.17	8.16	77.82%
	500	**8.09**	**8.03**	78.31%
	1000	8.62	8.66	**79.52%**
	1500	9.27	9.29	78.60%
初中	200	16.81	16.65	61.17%
	300	13.47	13.35	64.81%
	500	**12.33**	**12.11**	69.93%
	1000	12.6	12.2	**70.48%**
	1500	13.24	12.79	70.37%
高中	200	37.4	37.73	54.31%
	300	31.88	32.47	58.27%
	500	26.99	**27.04**	62.79%
	1000	**26.56**	27.23	64.47%
	1500	28.68	29.03	**65.13%**

表 2. 不同模型容量对结果的影响

	Dropout 概率	最佳模型的 epoch 数	验证集上的困惑度	测试集上的困惑度	测试集上的准确率
初中	0.1	31	14.96	14.29	68.45%
	0.3	29	13.16	12.77	69.67%
	0.5	67	**12.33**	**12.11**	**69.93%**
	0.6	65	12.8	12.66	66.88%
	0.7	69	14.59	14.44	62.75%
	0.8	116	18.31	18.23	57.38%

表 3. 不同 Dropout 概率对结果的影响

	小学试题	初中试题	高中试题
小学模型	79.52%	39.09%	22.99%
初中模型	72.22%	70.48%	33.65%
高中模型	58.11%	52.15%	65.13%

表 4. 模型的交叉预测，表中的数据为在测试集上的准确率

为起点，在初中数据集上重新训练的话，会导致大部分词被识别为未知单词，效果比较糟糕。初中和高中数据集之间也存在同样的问题。所以，我只做了 3 组实验：高中、初中的模型在小学数据集上的微调，以及高中的模型在初中数据集上的微调，实验结果如表 5. 所示。

从表 5. 中可以看出，使用微调的方法确实对正确率的提高有一定帮助。以初中的模型为起点，在小学的数据集上微调后，准确率达到了 80.30%，比在小学数据集上直接训练的正确率 79.52% 要高。不过，使用高中的模型进行微调并没有取得更好的效果。

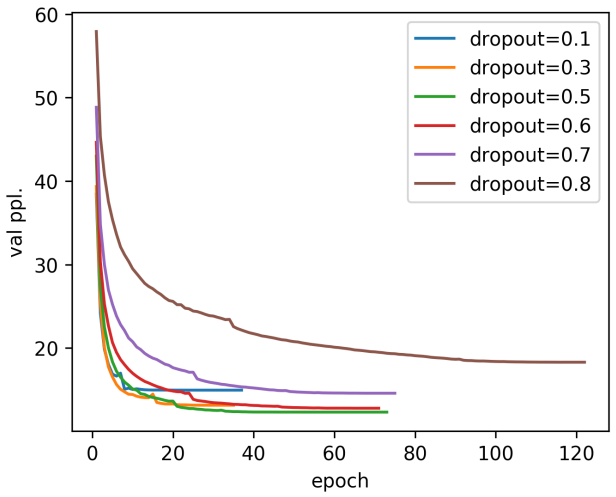


图 7. 使用不同 Dropout 概率训练时，验证集上的困惑度随 epoch 的变化

	模型起点	在测试集上的准确率
小学	/	79.52%
	初中	**80.30%**
	高中	78.31%
初中	/	**70.48%**
	高中	70.04%

表 5. 以不同年级的模型作为起点微调 (fine-tune) 的结果。表中用 “/” 表示不微调，直接训练。

3.7. 引入外部语料库

在章节 3.6 中，我尝试了使用不同数据集上的模型进行微调的方法。一个自然而然的想法是：既然可以在自己的数据集之间进行微调，那是否可以使用外部语料库先训练出模型，然后在这些模型的基础上进行微调呢?

我选取了 nltk.corpus 中的 3 个语料库进行尝试，它们分别是：

- Brown 语料库[1]。它是世界上第一个超过 100 万个单词的英文语料库。(以下简写为 brown)
- Sentiment polarity 数据集[2]。它是一个情感分析数据集，包含 1000 条积极的和 1000 条消极的电影评论，也被称为 Movie Review 数据集。(以下简写为 mr)
- Australian Broadcasting Commission 数据集[3]。一个澳大利亚广播节目的语料库。(以下简写为 abc)

	模型起点	在测试集上的准确率
小学	/	**79.52%**
	brown	79.31%
	mr	78.53%
	abc	77.96%
初中	/	70.48%
	brown	69.89%
	mr	69.45%
	abc	**70.67%**
高中	/	65.13%
	brown	**65.99%**
	mr	65.63%
	abc	65.24%

表 6. 使用外部语料库模型作为起点微调的结果。brown 表示 Brown 语料库，mr 表示 Sentiment polarity 电影评论数据集，abc 表示 Australian Broadcasting Commission 数据集。“/” 表示直接进行训练，不微调。

在外部语料库上训练模型时，分别使用参数 n_embed=n_hidden=1000 和 1500。由于外部语料库中的单词非常多，不可能全部包含在模型里，所以我用 nltk.FreqDist 找出最频繁的 25000 个单词进行训练，其余单词标记为 <unk>。分别使用之前效果最好的参数对应的模型进行微调。微调的结果如表 6. 所示，从中可以看出：

- 在训练初中和高中的模型时，引入外部语料库进行微调对结果有所助益！初中的模型在使用 abc 数据集微调后准确率为 70.67%，较之前的 70.48% 有小幅上升。在训练高中的模型时，三个语料库都可以使预测准确率上升。原因可能是高中题目更加接近现实世界中的英语环境，因此使用外部语料库帮助较大。

[1] http://www.hit.uib.no/icame/brown/bcm.html

[2] http://www.cs.cornell.edu/people/pabo/movie-review-data/

[3] http://www.abc.net.au

- 在训练小学的模型时，引入外部语料库反而会造成准确率的下降，原因可能是小学的试题和真实世界的语料库在联合分布上相差较大。

4. 最佳模型参数

最后，给出在所有实验中，最佳模型的参数和训练方法。分为小学、初中和高中三个模型：

- **小学**：使用 2 层 LSTM，在初中的训练集上以参数 n_embed=n_hidden=1000 进行训练，Dropout 概率设为 0.5，训练好的模型再在小学的训练集上进行微调，测试准确率是 80.30%。
- **初中**：使用 2 层 LSTM，在 Australian Broadcasting Commission 数据集上以参数 n_embed=n_hidden=1000 进行训练，Dropout 概率设为 0.5，训练好的模型再在初中的训练集上进行微调，测试准确率是 70.67%。
- **高中**：使用 2 层 LSTM，在 Brown 数据集上以参数 n_embed=n_hidden=1500 进行训练，Dropout 概率设为 0.5，训练好的模型再在高中的训练集上进行微调，测试准确率是 65.99%。

5. 总结

在本次 NLP 项目中，我尝试应用自然语言处理技术来完成一个有趣的项目：自动做英语单选题。为此，我首先使用爬虫在网上爬取了小学、初中、高中的 6 万余道习题。利用深度学习建立语言模型，在此过程中做了大量实验，探索了不同 RNN 计算单元、不同模型容量、不同 Dropout 概率、不同模型微调起点对结果的影响。小学、初中、高中的测试集分别包含 1411、2717、1970 道试题，最佳模型在这些测试集上的准确率分别达到了 80.30%，70.67% 和 65.99%，结果比较令人满意。

这次项目经历让我对自然语言处理技术（尤其是其中建立语言模型的相关知识）有了更加深入的了解，锻炼了自己编写爬虫，使用 PyTorch 搭建深度学习模型的能力，收获颇丰。由于时间关系，还有一些后续的想法没有来得及验证，可以在今后继续尝试优化模型，如：

- 更多的参数组合（如 n_embed、n_hidden、Dropout 概率，学习率等等）。

- 更合理的训练方法。如在微调模型时，我采取的是先在一个数据集上训练，再在另一个数据集上微调的策略。可以考虑一种混合策略：混合两个数据集，然后在每个 epoch 结束时逐步减少其中一个数据集的比例，直到最后训练完成，也许可以取得更好的效果。

- 更复杂的网络结构。在建立语言模型时，采取了比较简单的双层 RNN 结构。或许可以针对这个问题设计一个更复杂、更专用的网络结构，来获得性能上的提高。

6. 附录

6.1. 项目文件列表

项目文件结构如下：

```
/15307130385_刘婧源
    /code
        /data
            |...
        /lm
            |...
        /scrape
            |...
        |gen_nltk_corpus.py
        |gen_xrs_json.py
        |split_all_txt.py
        |split_xrs_json
    | 项目报告.pdf
```

文件夹内细节如图8. 所示：

图 8. 文件夹细节及解释

6.2. 模型训练、测试的运行方法

code/lm/是与深度学习语言模型相关的代码，包含了模型训练、测试、绘图等功能。需要的运行环境为：

- Python 2.7
- PyTorch 0.2.0（需支持 GPU）

训练模型的入口程序是 train.py，这个文件接收一个 --tvt_corpus 参数，表示语料库所在的文件夹，文件夹下必须含有 train.txt、valid.txt、test.txt 三个文件，分别表示训练集、验证集、测试集。在 code/data/中提供了我们收集的三个数据集：code/data/primary 为小学的试题，code/data/middle 为中学的试题，code/data/high 为高中的试题。

和模型相关的超参数有 --lr、--model、--emsize、--nhid、--nlayers、--tied，分别表示学习率，计算单元类型（可选的有 RNN_TANH、RNN_RELU、GRU 和 LSTM）、word_embedding 的维数、隐层状态的维数、计算单元层数、是否使用 tied 模型（即输入输出使用相同的 embedding）。

最后，参数--cuda 表示是否使用 GPU 训练，--model_dir 表示存储模型的根目录，程序会根据模型的参数生成一个子目录用于存储模型。例如指定：

```
python train.py \
        --emsize 1000 --nhid 1000 \
        --tied --dropout 0.5 \
        --tvt_corpus ../data/primary \
        --model_dir models/primary/ \
        --cuda
```

程序会读取../data/primary 中的语料库进行训练，训练好的模型及相关信息会保存到文件夹 models/primary/emsize1000nhid1000dropout0.50tiedTrue/中。一共会保存四个文件：

- train.log：训练中产生的日志，日志最后有在测试集上做题的准确率。
- train.csv：训练中各个 epoch 结束时的损失、困惑度，主要用于画图。
- model.pt：最佳模型文件。
- dict.pkl：将单词映射到 id 的字典文件。

在 code/lm/scripts 中提供了一些方便的脚本，可以方便地验证3中的所有结果。它们是：

- train_all_rnn_unit.sh（在 code/lm/文件夹下，使用命令 sh scripts/train_all_rnn_unit.sh 运行，下同）：验证3.3中的结果。
- train_all_lstm.sh：验证3.4中的结果。
- train_dropout.sh：验证3.5中的结果。
- train_local_finetune.sh：验证3.6中的结果。
- train_all_finetune.sh：验证3.7中的结果。在运行这个脚本之前，必须先制作 nltk 中的三个数据集，方法是回到 code/文件夹中，依次运行：

```
python gen_nltk_corpus.py
# 下面三条命令需要回车确认
python split_all_txt.py data/abc/all.txt
python split_all_txt.py data/mr/all.txt
python split_all_txt.py \
            data/brown/all.txt
```

6.3. 爬虫运行方法

code/scrape 是爬虫代码，它可以将题目爬取下来，并存储到 MongoDB 中，需要的运行环境为：

- Python 3.6（注意和语言模型运行时需要的环境不同）
- 一些程序中用到的 Python 的库，如连接数据库的 pymongo，HTTP 请求的 requests，解析 HTML 的 BeautifulSoup 等。
- MongoDB 3.6.1。需要运行在默认的 27017 端口，不设置用户名和密码。事先创建好一个空的名为“nlp”的数据集。

环境准备好后，直接在 code/scrape 中先建立一个 logs 文件夹用于保存爬虫日志，接下来分别运行下面

的命令就可以启动爬虫了：

```
python get_xrs_primary.py
python get_xrs_middle.py
python get_xrs_high.py
```

数据会以 Json 形式写在 nlp 数据库名为 “data” 的 collection 中。日志会分别保存为 logs/primary.log、logs/middle.log 和 logs/high.log。

6.4. 数据集

项目所构建的数据集已分享至百度云盘：https://pan.baidu.com/s/1i5Vj1yt, 密码：bsv8：。

在 data 文件夹中，分为高中试题（high）、初中试题（middle）、小学试题（primary）。在每个文件夹中有四类文件：all 为所有的试题，test 为测试试题，train 为训练试题，valid 为验证试题，各类数量如表7. 所示。

	train	valid	test	all
primary	11,283	1,410	1,411	14,104
middle	21,731	2,716	2,717	27,164
high	15,755	1,969	1,970	19,694

表 7. 项目数据集概览

每种文件有两个类型，分别为 json 和 txt 格式：

- json 格式用 Json 串来表示单个选择题数据。单个 Json 串包含了题干、选项、答案在内的所有信息。
- txt 格式为填入正确答案之后的题干。

参考文献

[1] K. Cho, B. V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. Computer Science, 2014. 4

[2] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997. 4

[3] H. Inan, K. Khosravi, and R. Socher. Tying word vectors and word classifiers: A loss framework for language modeling. 2016. 6

[4] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. Nature, 521(7553):436, 2015. 3

[5] O. Press and L. Wolf. Using the output embedding to improve language models. 2016. 6

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1):1929–1958, 2014. 6

[7] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In Interspeech, pages 601–608, 2012. 4

[8] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. 2017. 3