

第七课 背单词软件

身边不少同学在学习编程的阶段，同时也在进行英语的学习。所以不少同学都想过定制自己的背单词软件。在这节课，我们来实现一个背单词软件的系统。我们先初步设想一下这个背单词软件所具有的功能。

背单词软件的功能设计

- 一般来说，市面上的背单词软件都支持对于一个单词，用选择题的方式让用户从几个干扰选项中，选择出正确的英文意思。所以这个需求我们首先会考虑实现。
- 同时，我们的背单词程序还要能够管理每个单词的熟练度。知道同学对每个单词的掌握情况。在每天开机的时候，帮助同学复习自己最不熟练的单词。
- 因为在前面几节课我们已经尝试使用了大语言模型的api，所以在这里我们想额外增加一个功能，即使用若干个学生掌握得不太好的生词，进行“造句”。帮助学生理解和学习新的单词。

单词的数据结构

这里我们首先要考虑单词在我们这里存储的数据结构。我们可以向GPT询问下列问题

- 1 我想制作一个Python背单词软件， 用List存储每个单词(english_word)和对应的中文意思(chinese_meaning)
- 2
- 3 这里List的每个元素用什么类型的结构会比较好？

GPT的回复

在Python中，存储单词及其对应中文意思的最佳方式是使用字典（dict）或者自定义类（class）。以下是两种方式的示例：

使用字典：

字典可以很方便地通过键值对的方式存储单词和中文意思。

```
1 word_list = [  
2     {"english_word": "apple", "chinese_meaning": "苹果"},  
3     {"english_word": "banana", "chinese_meaning": "香蕉"},  
4     # 更多单词...
```

...

这里我们涉及到了

字典的基础知识

同学们可以向GPT询问

- 1 什么是Python中的Dict字典？

来获取字典数据类型的基础知识。这里比较关键的是，字典可以用变量名['键值名']的方式，来获取字典变量的每个字段。相比之下，如果我们使用data[0]来存储英文，data[1]来存储中文意思，时间长了以后我们就无法记得0和1对应的意思。使用字典和字段名可以有效地增加程序的可读性。

除了这种结构化存储，字典还会用来建立映射。这和“字典”这个词的本身意思很像。在我们后面背单词的例子中，会出现两个english2score 和english2chinese的字典，这里的字典就是用来做映射和查询作用的。

字典相关的prompt

字典在Python中是个很常用的数据类型。并且经常会和我们之前学到的List进行组合，形成List of Dict。字典中的每个key会对应一个value。下面是我们在编程中，可能会围绕字典出现的一些prompt 语句

- 1 {变量名}是一个Dict，他有a,b,c,d这几个字段
- 2
- 3 将结果输出为一个字典，其中字段a是，字段b是，。。。

用GPT生成例子数据

这里我们想让语言模型先帮我们生成一些单词的例子，可以用下面的prompt，让语言模型先多生成一些例子

- 1 我正在设计一个背单词的程序
- 2
- 3 我希望构造word_datas，一组长度为30的List of Dict，每个元素有字段 english 和 chinese_meaning

```
4
5 english是单词，chinese_meaning是单词的中文示意
6
7 请帮助我用word_datas= [...]的形式生成一组单词表
```

GPT的输出为

```
1 word_datas = [
2     {"english": "apple", "chinese_meaning": "苹果"},
3     {"english": "banana", "chinese_meaning": "香蕉"},
4     # ...
5     {"english": "door", "chinese_meaning": "门"},
6     {"english": "flower", "chinese_meaning": "花"}
7 ]
```

这里我们把word_datas的赋值语句放在lesson7/utils.py中，这样对于同目录的代码，就可以用from utils import word_datas来载入这个List了。

单词管理类

我们已经有了基础的数据。现在我们希望建立一个WordManager的类。这个类不包含任何前端交互的内容，只管理“用户背单词的熟练度”。显然，这个类应该能够获取到word_datas的信息，或者至少知道有哪些words。

```
1 请帮我实现一个WordManager类，包含下面的方法
2
3 __init__( word_datas )， 可以通过一个List of Dict初始化，其中每个元素有english和
  chinese_meaning字段
4
5 并且我希望每个单词有一个熟练度"score"，所有的单词初始熟练度定义为20 （可以建立一个
  english2score的map）
6
7 同时定义save()和load( file_name = None ) 的方法，默认把english到熟练度的映射存储在
  data/word2score.jsonl中，第一次初始化的时候如果没有这个文件，则新建。
8
9 定义get_word_data( ) 方法，返回熟练度最低的word_data，如果有数个熟练度相同的单词，随机
  返回其中一个
10
11 定义add_score( english_word , delta = 1) 方法，将word的熟练度增加delta，delta默认为
  1
12 定义minus_score( english_word, delta = 1 ) 方法，将word的熟练度减少delta，
13 同时注意保持熟练度最小值是0 最大值是100
```

```
14
15 定义get_alternative( english_word, n = 3 )方法, 返回一个list of string, 返回n个不是
    word的chinese_meaning
16
17 请帮助我实现这个python类, 并编写合适的测试。
```

因为这个类一次性包含的功能很多, 我们逐个成员函数来讲解GPT对应生成的代码

__init__(word_datas)

这里__init__非常直白,

```
1  def __init__(self, word_datas):
2      self.english2score = {}
3      self.english2chinese = {}
4      for word_data in word_datas:
5          self.english2score[word_data['english']] = 20
6          self.english2chinese[word_data['english']] =
            word_data['chinese_meaning']
7      self.save_path = 'data/word2score.jsonl'
8      self.load()
```

这里GPT除了我们要求的english2score字典之外, 还额外生成了english2chinese的一个映射。这样程序就可以用self.english2chinese['{英语单词}'] 来查询对应英语单词的中文意思。并且在__init__中, GPT用了一个循环来构造对应的这两个map。这里看起来每个单词的熟练都被初始化为20, 但是在self.load中, 实际上会去更新这个熟练度。

save和load

```
1  def save(self):
2      if not os.path.exists('data'):
3          os.makedirs('data')
4      with open(self.save_path, 'w') as f:
5          for english, score in self.english2score.items():
6              f.write(json.dumps({english: score}) + '\n')
7
8  def load(self, file_name=None):
9      file_name = file_name or self.save_path
10     if os.path.exists(file_name):
11         with open(file_name, 'r') as f:
12             for line in f:
13                 data = json.loads(line)
```

save非常直白，如果对应的data文件夹不存在，则会新建这个文件夹。在load中，如果存在过往的背单词的记录，则会对english2score即每个单词的熟练度进行更新。

get_word_data和更新score

```
1 def get_word_data(self):
2     min_score = min(self.english2score.values())
3     lowest_score_words = [word for word, score in self.english2score.items() if
4                             score == min_score]
5     selected_word = random.choice(lowest_score_words)
6     return {'english': selected_word, 'chinese_meaning':
7             self.english2chinese[selected_word]}
8
9 def add_score(self, english_word, delta=1):
10    self.english2score[english_word] = min(100,
11                                             self.english2score[english_word] + delta)
12
13 def minus_score(self, english_word, delta=1):
14    self.english2score[english_word] = max(0, self.english2score[english_word]
15                                             - delta)
```

这里更新score的部分非常直白。至于get_word_data，GPT生成的代码先求出所有分数中的最小值，再选取分数中和最小值相等的英语单词。从中随机抽取一个之后返回单词和对应的中文意思。

get_alternative

这个函数是用来设计生成备选选项的

```
1 def get_alternative(self, english_word, n=3):
2     alternatives = [meaning for word, meaning in
3                     self.english2chinese.items() if word != english_word]
4     return random.sample(alternatives, n)
```

测试代码

这里GPT给出了合理的测试代码。注意这些代码被if __name__ == "__main__":的条件包裹，这样我们可以把测试代码和类代码一起放到单独的文件WordManager.py，并且可以单独运行类的文件进行测试。而在import WordManager的时候，不会被测试代码干扰（因为这个时候__name__是WordManager）。

```

1 if __name__ == "__main__": # 测试代码
2     word_datas = [...] # word_datas的赋值代码, 这里要修改为from utils import
    word_datas
3
4     manager = WordManager(word_datas)
5     print("初始单词数据:", manager.get_word_data())
6
7     manager.add_score("apple", 5)
8     print("增加分数后:", manager.get_word_data())
9
10    manager.minus_score("banana", 3)
11    print("减少分数后:", manager.get_word_data())
12
13    alternatives = manager.get_alternative("apple")
14    print("苹果的替代词:", alternatives)
15
16    manager.save() # 保存数据
17
18    new_manager = WordManager(word_datas) # 加载数据
19    new_manager.load()
20    print("加载后的分数:", new_manager.english2score)

```

将WordManager存储到单独文件

我们需要把之前的WordManager的代码保存到一个独立的WordManager.py的文件中。这个时候同学们要做两件事来进行测试，一是运行WordManager.py验证这个类是能被正确运行的。（这里老师保存在了lesson7/WordManager.py中）

```

初始单词数据: {'english': 'banana', 'chinese_meaning': '香蕉'}
增加分数后: {'english': 'banana', 'chinese_meaning': '香蕉'}
减少分数后: {'english': 'banana', 'chinese_meaning': '香蕉'}
苹果的替代词: ['酸奶', '木琴', '铅笔']
加载后的分数: {'apple': 30, 'banana': 14, 'cat': 20, 'dog': 20, 'elephant': 20, 'fish': 20, 'ket': 20, 'kite': 20, 'lemon': 20, 'monkey': 20, 'nose': 20, 'orange': 20, 'pencil': 20, 'e': 20, 'umbrella': 20, 'violin': 20, 'water': 20, 'xylophone': 20, 'yogurt': 20, 'zoo': 20, 'ower': 20}
PS D:\aistudio\Learn-Python-with-GPT> ^C

```

另一方面，需要在同一个目录新建一个main.py，测试下面的代码是可以被顺利运行的

```

1 from WordManager import WordManager
2 word_datas = [...] # word_datas的赋值语句
3 word_manager = WordManager(word_datas)

```

至此，我们的单词管理类已经被顺利建立了。

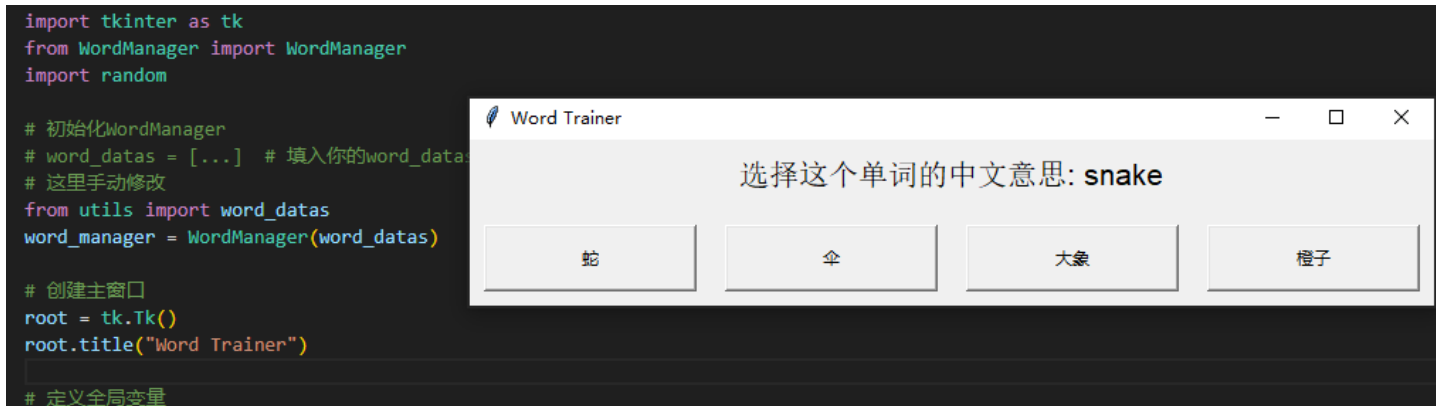
为背单词软件设计交互的界面

接下来我们需要为测试好的WordManager类编写合理的界面，

```
1 我已经实现了WordManger类，这个类可以用
2
3 from WordManager import WordManager
4 word_datas = [...] # word_datas的赋值语句
5 word_manager = WordManager(word_datas)
6
7 这样的方式载入，对应的功能如下
8
9 __init__( word_datas ), 初始化
10
11 get_word_data( ) 方法，返回熟练度最低的word (dict形式，有english和chinese_meaning字
    段)
12
13 add_score( english_word , delta = 1) 方法，将word的熟练度增加delta, delta默认为1
14
15 minus_score( english_word, delta = 1 ) 方法，将word的熟练度减少delta,
16
17 get_alternative( english_word, n = 3 )方法，返回一个list of string, 返回n个不是word
    的chinese_meaning
18
19 现在我希望实现一个tinker demo
20
21 这个demo每次会用get_word_data()获得一个单词，并渲染在第一个textbox上，label为"选择这个
    单词的中文意思"
22
23 然后使用get_alternative()方法，获得三个额外的chinese_meaning
24
25 然后混淆三个备选项和本身word的chinese_meaning，随机显示在4个选项上
26
27 当用户选择的时候，如果选择正确，则调用add_score(word)方法，否则，调用minus_score(word)
    方法
28
29 在修改分数之后，调用word_manager.save()保存，然后重新获取一个新的单词进行渲染
```

可以看到prompt的前半段描述了WordManager的功能，后半段则提出了构造选择题的界面的需求。

需要注意的是这里每次生成的界面形态是会有所不同的



当然同学们可以增加更多限定性的语句，来加强整体界面的排版。在写这章的时候我们发现要求GPT，使用tinker或者PyQT的库来实现的时候，生成代码的准确率比较高。而使用gradio生成的时候，报错的概率比较高。（这是由于在训练GPT的时候，gradio的版本还不稳定，所以往往GPT学习了两个版本的gradio代码的数据，认知上会出现很大的混乱）所以在上课的时候，我们建议使用tinker库来实现。但是gradio有一些好处比如可以一键设置共享链接，支持多人访问，并且界面看起来也更现代一些。在作业中同学们可以尝试修改为gradio实现。或者采用第四课中提到的结合例子文档让GPT进行生成的方法来生成。这个版本的代码我们保存在lesson7/word_mem_simple.py中。

进一步修改WordManager方法

初步测试了一下我们的界面后，我们发现后端的WordManger在使用上有一些小的瑕疵。

- 假如有一个单词的score明显低于其他单词，那么在get_word_data方法抽取单词的时候，会反复抽取到这个单词，用户的学习体验上是比较差的。

另一方面，前面提到我们想利用语言模型，来对M个单词生成一段包含这M个生词的句子，来加强这M个单词的学习。那我们就需要对应增加一个函数get_M_words的方法，来获得这M个单词。

我们考虑在之前成功运行的WordManager.py上进行修改

- 1 {复制WordManager.py的代码}
- 2
- 3 这段python代码可以成功运行，我希望对WordManager类进行修改
- 4
- 5 将get_word_data() 改为get_word_data(K = 4)
- 6
- 7 会先从所有数据中，找到K个分数最低的单词，再从中随机挑选一个，返回对应的dict
- 8
- 9 另外实现一个get_M_words(M = 4)的方法，先从所有数据中，找出2M个分数最低的单词，
- 10
- 11 再从中随机挑选M个，以list of string的形式返回
- 12
- 13 并修改对应的测试代码

这里GPT很轻松地完成了对应两个方法地修改

```
1     def get_word_data(self, K=4):
2         sorted_words = sorted(self.english2score.items(), key=lambda item:
    item[1])
3         lowest_score_words = [word for word, score in sorted_words[:K]]
4         selected_word = random.choice(lowest_score_words)
5         return {'english': selected_word, 'chinese_meaning':
    self.english2chinese[selected_word]}
6
7     def get_M_words(self, M=4):
8         sorted_words = sorted(self.english2score.items(), key=lambda item:
    item[1])
9         lowest_score_words = [word for word, score in sorted_words[:2 * M]]
10        selected_words = random.sample(lowest_score_words, M)
11        return selected_words
```

并且通过运行测试代码，我们可以确保WordManager类仍然是正常的。

用生词造句

我们想使用这样一个prompt，来为生词进行造句。

```
1 我正在学习英语，我对下列单词不是很熟悉：
2
3 'grape', 'elephant', 'jacket', 'ice'
4
5 请帮我用这些单词，生成一个包含这些单词的简短的故事，再将生成的结果翻译回中文
```

因为这里的单词，实际上是WordManager的get_M_words方法返回的，那这个时候，我们就需要GPT来写一个函数，来组装这个prompt

```
1 from utils import word_datas
2 from WordManager import WordManager
3 manager = WordManager(word_datas)
4 words = manager.get_M_words()
5
6 这段代码可以顺利返回M个 我最不熟悉的单词
7
8 我希望实现一个函数words2prompt，输入是words，输出是形如
9
```

```

10 ```
11 我正在学习英语，我对下列单词不是很熟悉：
12
13 'grape', 'elephant', 'jacket', 'ice'
14
15 请帮我用这些单词，生成一个包含这些单词的简短的故事，再将生成的结果翻译回中文
16 ```
17
18 的字符串prompt，把中间的单词替换为对应的words

```

GPT的实现

```

1 def words2prompt(words):
2     # 将单词列表转换为字符串，每个单词用单引号包围，并用逗号和空格分隔
3     words_str = "', '".join(["'{}'".format(word) for word in words])
4
5     # 构建prompt字符串
6     prompt = (
7         "我正在学习英语，我对下列单词不是很熟悉:\n\n"
8         f"{words_str}\n\n"
9         "请帮我用这些单词，生成一个包含这些单词的简短的故事，再将生成的结果翻译回中文"
10    )
11
12    return prompt

```

这里老师将这个函数贴在了lesson7/words2prompt.py中，并且编写了对应的测试代码

```

1 from utils import word_datas
2 from WordManager import
  WordManager
3
4 word_manager =
  WordManager(word_datas)
5 words =
  word_manager.get_M_words()
6
7 prompt = words2prompt(words)
8 print(prompt)
9
10 from question2response import
  question2response
11 response =
  question2response(prompt)

```

```

我正在学习英语，我对下列单词不是很熟悉：
'jacket', 'dog', 'elephant', 'banana'

请帮我用这些单词，生成一个包含这些单词的简短的故事，
---
它们

故事：

In the sunny afternoon, a small dog named Jack was p
d a big banana from the tree and gave it to Jack. Ja

翻译：

在一个阳光明媚的下午，一只名叫Jack的小狗和它的象朋友
蜜的味道。

```

```
12 print("---\n",response)
```

运行对应的代码。这段代码是带着我们之前语言模型的question2response一起进行测试的。在图中可以看到语言模型顺利地用四个单词组合出了对应的故事。

将造句功能组合进界面

这里我们继续之前的lesson7/word_mem_simple.py代码进行改进

```
1 {复制word_mem_simple.py}的代码
2
3 这段python代码可以正确运行。我希望在当前界面下方，增加一块新的功能
4
5 增加一个一定高度的文本框，再加一个按钮“陌生词造句”
6
7 这个按钮会先调用
8 words = word_manager.get_M_words()
9
10 再调用
11 from words2prompt import words2prompt
12 prompt = words2prompt(words)
13
14 再调用
15 from question2response import question2response
16 response = question2response(prompt)
17
18 这里的response就是针对M个最不熟悉的词语进行的造句。
```

这里我们可以看到GPT能够比较顺利地地完成对应地代码生成



这里老师额外让GPT增加了文本框对应地滚动框，以及手动修改了一行代码，增加参与造句的单词。同学们可以阅读lesson7/word_mem_v0.py查看对应的代码。

总结

这节课，我们实践了一个背单词的软件。并且学习了字典数据类型的使用。在背单词软件开发的时候，我们先实现了后台的WordManager类。然后为其制作了配套的界面。然后我们又进一步增加了为最不熟悉的M个单词进行造句的功能。

课后练习

单词表的替换

- 对于目前的程序实现，如果我想替换单词，要修改哪个部分的代码呢？
- 将目前的从代码载入单词，替换为从一个特定的文本或者表格进行载入。
- 尝试让GPT从特定的课文文本中，总结最难的几个单词，存储为生词

更科学的单词学习

- 在标准的背单词软件中，往往会使用“艾宾浩斯记忆曲线”的方法。简单来说就是同一个单词，假设学生每次都能背对，那么其展示间隔会成倍增长。这样的记忆方式是最有效的。这个时候我们如何修改我们的单词管理类，达到这样的效果？

修改界面为Gradio

- 在课上我们使用了tinker作为我们的界面实现。尝试将其修改为pyqt或者是gradio的实现。

更进一步的修改

- 生成的阻塞。当点击“陌生词造句”的时候，程序会有6-10秒左右的停止响应。这是因为生成的过程特别是question2response发生了“阻塞”。尝试和GPT讨论在交互上怎么缓解这个问题。让程序不要显得是没有响应的。
- 语音的接入。尝试寻找TTS语音的API接口，并进行接入。
- 定制每天的学习计划。对于一个新的单词列表，可以显示每天的学习计划，批量加入到背单词学习中。