

MiniGLM模型训练实验报告

2022011547 吕博涵

注：和模型相关的进行过修改的代码存在MiniGLM文件夹中，部分微调数据存在datasets文件夹中。

数据的准备

数据内容

默认的训练方法是先用金庸小说原文进行训练，然后使用问答集进行微调。但是经实验证明，这样的训练方法最多通过过拟合让模型能答对微调数据集中的问题，但是不能让模型拥有对语言的基本的理解能力：如理解“郭靖”是一个人名，每一句应该有主谓宾等结构等。因此，除了基本的小说外，我准备了更大量的中文预训练预料对模型进行训练，让模型拥有基本的语言模型的遣词造句的能力，如分清人名、地名，识别句中的主谓宾等。

我从Hugging Face上以及github上获取了众多预训练数据，其中有一些数据的格式是parquet，而服务器上没有能解析这种格式的包，因此我在本地将其转化为jsonl格式后上传到服务器。

我准备的预训练预料如下：

- 历史上三年的搜狐新闻语料（约1G）
- 和金庸及其作品相关的百科
- 大量知乎问答语料（约2G）
- 筛选出与金庸及其作品相关的问答语料

以上的两种知乎问答语料我做了两种处理，一个是处理成预训练的格式，一个是处理成微调的格式。因此我的微调包含以下问答数据集：

- 大量知乎问答
- 金庸及其作品相关的知乎问答
- 小说内容问答（以及从中提取的精选问答）

其中小说内容问答就是在基础上扩充的微调数据集，我与俞鹤扬同学、朱子晗等同学、刘建东等同学共同补充微调数据集，总共积累了数万条高质量数据。

我生成微调数据集使用了以下方法：

- 从网上获取金庸相关的题库并对其格式进行处理得到一部分高质量问题集。
- 直接指定小说给定格式使用ChatGPT、Claude2、ChatGLM等LLM生成问答对。（在这三个中，ChatGLM的准确度最高。不过有趣的是，ChatGPT和ChatGLM都会主动说出“小昭成为了张无忌的义女”这样相同的错误的内容，怀疑是都学了不太好的语料。）
- 给Claude2提供维基百科上和小说内容相关的长文本，让其根据这些内容生成问答对。
- 使用ChatGPT根据示例生成可能出现的问题，并使用形如"Q: <Example Question>, A: <Example Answer>, Q: <New Question>, A:"的prompt批量调用ChatGPT的api（使用3.5-turbo）生成答案并进行记录（属于One-Shot Learning）。

其中最后一种方法生成的效果最好，但是成本较高；第三种方法次之，由于给定了参考的材料，其生成问答的能力远远强于第二种方法。

数据处理

由于不同的数据集格式不同，我分别设计了不同的数据处理函数。数据处理函数主要分为两类：一类是处理与训练数据，一类是处理微调数据。这里着重介绍一下以prepare_sft.py为例的微调数据处理程序和data_utils.py中的get_batch_sft()函数。

在prepare_sft.py中，我首先使用combined_qa = question + "[SEP]" + answer对每个问答对的问题和答案进行拼接，然后使用process(content, block_size)函数对拼接好的问答进行处理，对长于block_size的字符串进行截断，并对短于block_size的字符串用enc.eot_token进行补全。

```
def process(content, block_size):
    tokens = list(enc.encode(content))
    if len(tokens) > block_size:
        return tokens[:block_size]
    else:
        return tokens + [enc.eot_token] * (block_size - len(tokens))
```

data_utils.py中的get_batch_sft()函数的主要部分如下：

```
def get_batch_sft(split, batch_size, block_size, device):
    ix = torch.randint(len(data) // block_size, (batch_size,))
    x = torch.stack([torch.from_numpy((data[i*block_size:
(i+1)*block_size])).astype(np.int64)) for i in ix])
```

```

y = torch.cat((x[:, 1:], torch.full((x.shape[0], 1), enc.eot_token,
dtype=torch.int64)), dim=1)

sep_sequence = enc.encode("[SEP]")
sep_first_id = sep_sequence[0]

init_positions = (y == sep_first_id).nonzero(as_tuple=True)
loss_mask = torch.zeros_like(x, dtype=torch.float64)
for i, sep_position in enumerate(init_positions):
    try:
        actual_answer_length = (y[i,
sep_position[1]+len(sep_sequence):] != enc.eot_token).sum().item()
        loss_mask[i,
sep_position[1]+len(sep_sequence):sep_position[1]+len(sep_sequence)+actual_answer_length+1] = 1
    except:
        pass

```

我做了如下设计：

1. `ix`与`x`的设置与预训练不同，只保留问题起始部位的位置对应的序号，这样能让模型每次的学习被限制在一个问题之内，而不会从前一个问题的答案开始，到后一个问题的问题内容结束。这样的操作通过对`ix`和`x`的特殊处理实现。
2. `y`的定义是`x`从第二项开始的`block_size-1`个字符加上一个`enc.eot_token`，保证`y`不会进入后一个问题。
3. `loss_mask`是从`y`的问题开始到第一个`enc.eot_token`为1，其他为0。这样的方式是通过实验得出的。我曾经尝试过另外两个不同的设置：1.从答案开始全部为1；2.只有答案部分为1。结果是第一种方式训练出的模型的答案普遍较短，第二种方式训练出的模型的答案普遍收不住。而将`loss_mask`设置为从`y`的问题开始到第一个`enc.eot_token`为1这样的方式，得到的效果最好。

如此处理的示意图如下:

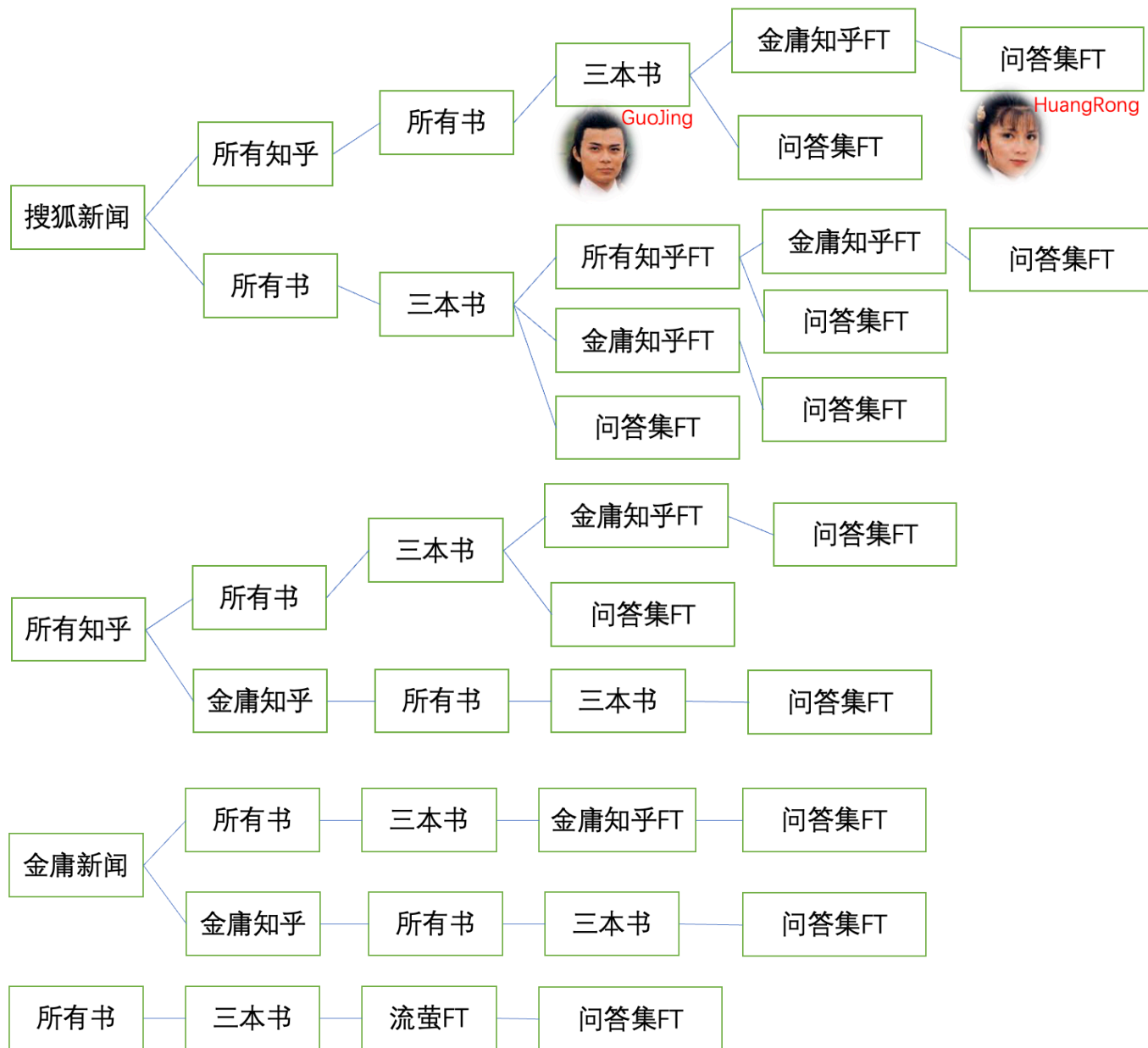
X: 郭 靖 是 谁 ? [SEP] 郭 靖 是 黄 蓉 的 丈 夫 。 [EOT]

Y: 靖 是 谁 ? [SEP] 郭 靖 是 黄 蓉 的 丈 夫 。 [EOT] [EOT]

MASK: 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0

模型训练

训练过程

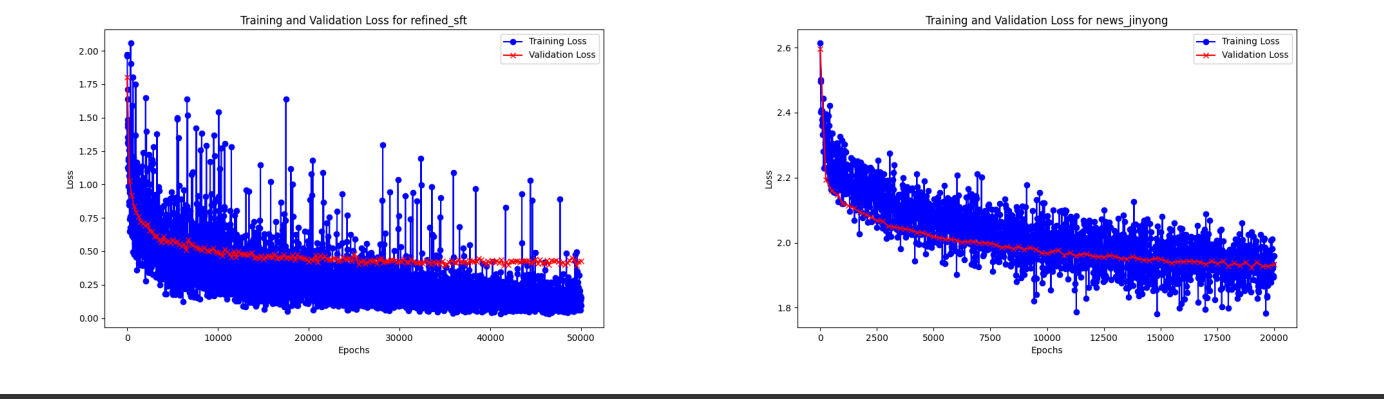


我用不同的预训练数据和微调数据的多种排列组合训练了众多模型，经过评测后分别为文本补全和问题回答选择了两个最好的模型：GuoJing (Generative unified operative Junction for integrated narrative growth)和HuangRong (Harmonized utility for answers and novel guidance with Rear-time organize neural grasp)。

模型正如小说中的人物，郭靖（GuoJing）木讷，只能补全信息，但是很强大，能够对文本结构产生理解；黄蓉（HuangRong）巧舌如簧，能顺利回答出各种刁钻的问题。

该流程的训练过程如下：搜素新闻20000轮，所有知乎20000轮，所有书6000轮，三本书60000轮，金庸类知乎问答微调20000轮，问答集微调20000轮。

部分训练过程中的损失函数曲线如下：



训练结果

极为突出的是，由于我的模型训练文本丰富，训练有序，因此不仅能很好地续写并正确回答问题，还没有出现过拟合。

预训练模型续写评测：

选择模型

Pretrain

prompt

黄蓉

Temperature

0.8

Clear

Submit

Output 1

黄蓉听他说话的话，便道：“老毒物，你毒物便不知道，已不许旁人着。”郭靖道：“这玩第三句话法如何？”江南六怪和他有一句话，登时脸如死灰，插口道：“你们两位说了毒物，能有什么毒物？”江南六怪道：“不成，不成，我还得道从头上说。咱们丘师父不愿丘师父。”黄蓉道：“瞧咱们丘师父的话，不用说甚么，你还有什么好处？”江南六怪笑道：“老毒物不算了，岂不是老毒物的个个倒不会武功，只是一个自南六怪都是第一流的高手，若不是自己与他老人家那一流好手，我能仿了你们的闺女。”尹志平道：“你想丘师父是谁？慕容

Output 2

黄蓉笑道：“爹爹，你别烦我，我养一十天，别烦我。”黄药师见她不起，心中一惊，说道：“郭靖，我别讨好看。”黄药师道：“蓉儿，你还是我师父一次后患。”郭靖一招招讨问，候地伸出，右手一招“啊”的一声，另一名曲曲折折了一根枯柴，向她肩头拍去。黄蓉顺手一挥，只听得头顶门口呼啸，一枝枯柴一个粗鲁的鲁有脚身子落地，正是爹爹的小小白脸。黄蓉暗暗好笑，双手钢抓看弯柴，抓住他肩头，自己一惊，招招拆招，下盘膝坐在她空中。黄药师一惊，想起小白脸上的模样，知道哪里还

Output 3

黄蓉道：“谁是你的？”郭靖道：“那也不用说，我总好了。你说他是你的，那也不用说。”丘处机道：“你若跟我说我是谁？”郭靖道：“咱们找到了一人，定然坐了。”又想：“我父母那奸贼，可是比武之后，那自是他父母也不能跟谁一般见识。你我自己这五人虽然不会武功，亦不自禁的坐下，可也是始终不出。”韩小莹摇头道：“我……我只消有此意。”丘处机道：“嗯，是你此间知道我，我父亲本来非同小可。”郭靖道：“你是谁？”丘处机就觉得不对，转身出门去，只见两个青衣女郎身上夹着一个青衣女郎给他，给他引过了几个坑，只

Output 4

黄蓉与郭靖见了这等情陷气，心下暗暗称奇，暗筹思：“且慢！”欧阳锋笑道：“不错，是我生平最大的，但我三弟是个威严。”郭靖心想：“这当口是我夫妻，那知可不小心心生。”黄蓉更是恼怒，柔声道：“我求道：快走！”黄蓉嫣然一笑，道：“这是我教训得的。她是我的宝贝儿子，是我教训得的。”郭靖急叫：“她如何？你不怕吗？”黄蓉道：“拜见没有。我教训教的好处。”郭靖道：“莫非你教训教训她既然照料？”黄蓉道：“这是我推门之极的。我不知道。”郭靖道：“我总是推门之极，确是为了你而精疑团

Output 5

黄蓉接口道：“你们骗我，我就来救你。”黄蓉携着洪七公，脸上全无色，心想“我偏偏要救你啦，就可惜他，但他不敢杀你们，我就不肯救你。”转头向黄蓉道：“你偏要我这小姑娘，一来不可忘了我性命，我大叔也算不了他。”黄蓉急道：“那我就没有。”黄蓉道：“你怎么算个法子？”黄蓉道：“妈，你就算爹爹不信，这小姑娘有甚么这般的干系？”洪七公道：“你爹爹只说黄老邪的功夫，只不过这小畜生只因我爹爹的功夫实在太快，但爹爹就算不肯饶你，可也不算七公未必能杀你，也不用饶你爹爹。”黄蓉道：“

Flag

微调模型回答能力评测：

选择模型

SFT

prompt

黄蓉是谁?

Temperature0.6

ClearSubmit

Output 1

黄药师的独生爱女，郭靖之妻。脸色侠摩首之妻，全真教第三代掌门人。全真教七怪之一，他们与郭靖之间有着深厚的感情，互相扶持。黄药师对郭靖心生嫉妒和建筑有着深厚的感情，一直生下他。在绝情谷中，郭靖遇到了两位主要角色如欧阳锋、杨康等人的故事。

Output 2

黄蓉是《射雕英雄传》中的女主角，她是郭靖的妻子，林中的一位顶级高手，擅长算计谋和谋略。她在小说中扮演着重要的角色，她的角色是郭靖的师父，也是郭靖的师父洪七公的女儿。黄蓉聪明机智，精通武功和草药医术。她是一位出身于江湖的少年，擅长打狗棒法，绝技艺。她在小说中展现出了坚强的意志和勇敢的精神，成为了一位武功高强的英雄人物。

Output 3

黄药师的独生爱女，郭靖是《射雕英雄传》中的女主角之一，她是一位武功高强的女侠，性格温婉娴良。她的真实身份是，被誉为“天下第一美女”。她在小说中扮演着重要的角色，与其他主要角色之间有着纠葛和情感纠葛。

Output 4

黄药师张黄药师的独生爱女，郭靖的师父，母亲是杨康的妻子。他们一起经历了许多冒险和挑战，最终成为了武林中的传奇人物。其性居住在桃花岛上，只有母亲的特点和其他势力的纽带。其次，黄药师对待朋友忠诚，对待敌人毫不留情。在与男主角杨过相遇后，二人相互倾心，经历了一系列的考验和困难，最终在一起。他们的爱情故事中起到了重要的推动作用，也成为了《倚天屠龙记》中最为动人的情节之一。

Output 5

黄药师的独生爱女，郭靖之妻。她母亲生她时难产而死，从小与父亲相依为命。肌肤如雪，娇美无比，容貌美绝，景色壮丽。皮肤，美丽高，双眼热爱，十绝美。爱酒的美貌和高尚的品德使得她在江湖上。在江湖中，她多次误以为她们郭靖交往事，为他不惜放弃自己的身份和家族，甚至不惜牺牲自己，自己的美貌才华绝世，才想相互扶持，共度过了许多艰难的时刻，黄蓉有智聪明，才情出众，善于算计和谐，她的美貌和高尚的品德使她成功地入育无比，拥有了超凡的武功和智慧。

Flag

在服务器上，每条512长度的回答能在2.5秒内生成。

从上面的回答中可以看出，续写模型具有语言组织能力，问答模型没有出现过拟合，而且回答的内容都和文本相关，具有一定的泛化能力。

模型效果分析

首先，使用大量中文语料对大模型进行训练能显著提升大模型对语法、句法的理解能力。在这样的训练后，大模型做续写或者回答相关问题时，可能会出现事实性错误，但是在语法和句法上不会出错。

同时，用上文所述的mask方式在保证文本长度的情况下让模型成功学会什么时候应该终止生成。在上图的示例中，大模型的回答均以句号作为结尾。

但是，模型纵然能拥有一定的泛化能力，但是并不拥有真正的智能。比如，在受过如此多的语料后如果问大模型“郭靖是男的还是女的”这样的对于人类非常简单的问题，大模型并不能给出好的答复。

还有一个发现：在续写的模式下，如果temperature过低，容易生成很多重复，这可能是由于模型参数小的同时受训语料有限、同时新用来训练的语料基本盖过旧语料训练结果造成的。

模型效果量化评估

evaluations.py中，我使用了困惑度和Rouge-L两种评估方式。具体的评估方法是，准备一个本地的问答对数据集，然后调用模型根据问题生成回答，然后比对模型生成的答案和数据集中的答案，并用两种方法进行评估。

因为我们使用的损失函数是交叉熵损失，因此能较为轻易地得到困惑度。计算方法如下：

给定一个真实分布 p 和一个模型预测分布 q ，对于单个样本的交叉熵损失为：

$$CE(p, q) = - \sum_i p_i \log q_i$$

平均交叉熵损失：

$$\text{average } CE(p, q) = - \frac{1}{N} \sum_{j=1}^N \sum_i p_{ij} \log q_{ij}$$

困惑度 (Perplexity, PP):

$$PP = e^{-\frac{1}{N} \sum_{j=1}^N \sum_i p_{ij} \log q_{ij}} = e^{\text{average } CE(p, q)}$$

对于Rouge-L:

$$\text{Recall} = \frac{LCS(R, C)}{|R|}$$

$$\text{Precision} = \frac{LCS(R, C)}{|C|}$$

$$F_\beta = \frac{(1+\beta^2) \times \text{Recall} \times \text{Precision}}{\beta^2 \times \text{Precision} + \text{Recall}}$$

代码层面，困惑度的计算直接使用`math.exp(loss)`实现，Rouge-L使用`rouge`包中的`Rouge`类实现。

其他探索性质的尝试

由于这个参数量的Transformer的效果甚至不如笔者原来使用LSTM做文本续写的效果，因此我写了一个十分简单的LSTM模型（`lstm_model.py`）和对应的训练代码（`lstm_train.py`）。训练代码可以直接运行。

模型类代码：

```
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers,
batch_first=True):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
                                batch_first=batch_first)

    def forward(self, x):
        x, _ = self.lstm(x)
        return x
```



```
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super().__init__()
        self.lstm1 = LSTM(input_size, hidden_size, num_layers)
        self.linear = nn.Linear(hidden_size, hidden_size)
        self.lstm2 = LSTM(hidden_size, hidden_size, num_layers)
        self.dropout = nn.Dropout(p=0.3)

    def forward(self, x):
        x = self.dropout(x)
        x = self.lstm1(x)
        x = self.linear(x)
        x = self.lstm2(x)
        return x
```

训练过程:

```
● (base) py2022011547@machine00:~/hw2/MiniGLM$ python lstm_train.py
Step 0, Loss 0.014772500842809677
Step 10, Loss 0.014788216911256313
Step 20, Loss 0.014756015501916409
Step 30, Loss 0.014838072471320629
Step 40, Loss 0.014955786056816578
Step 50, Loss 0.014582602307200432
Step 60, Loss 0.014842474833130836
Step 70, Loss 0.014225970953702927
Step 80, Loss 0.014698131009936333
Step 90, Loss 0.014557737857103348
```