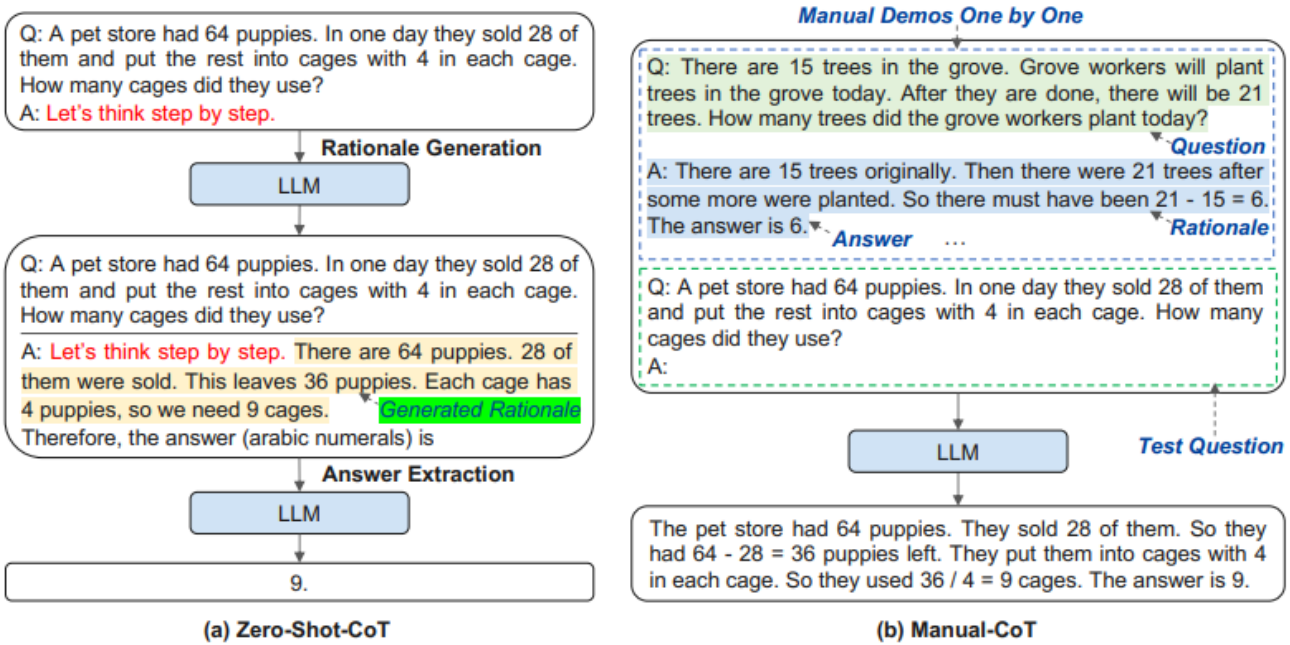
组会汇报

陈钶杰
专业:计算数学

August 8, 2023

目录

论文:Auto-CoT



- 背景:复杂的问题需要中间步骤来解决，这个中间步骤就需要用到我们的思维链的方法。
- Auto-CoT这种方法的想法就是自动产生Manual-CoT中的微调数据集。在问题后加入一个”let's think step by step”来推导出answer,然后将这个问题和回答当作微调数据集。

论文:Auto-CoT

- 如何选择的问题数据集?
  1. 检索式:检索特定的问题集:
  2. 随机式:随机寻找k个问题
  3. 使用聚类的方法，将n个问题通过聚类分成k个集群，然后在每个集群里面找出最优的s个问题,组成的问题集来进行zero-shot learning得到我们想要的微调数据。
- 结论：通过实验发现，在处理一些复杂问题时，使用Auto-CoT方法得到的结果准确性接近于Manual-CoT方法，明显优于Zero-Shot-CoT方法。

langchain

- LangChain是一个开源框架，允许从事人工智能的开发者将例如GPT-4的大语言模型与外部计算和数据来源结合起来。
- LangChain自身并不开发LLMs，它的核心理念是为各种LLMs实现通用的接口，把LLMs相关的组件“链接”在一起，简化LLMs应用的开发难度，方便开发者快速地开发复杂的LLMs应用。
- LangChain这是一个通用的框架，之后可以考虑用LangChain作为接口来做一些复杂的测试。

主要技术路线

方案1：Finetune

在基础大模型上，基于领域数据微调训练出一个**私有化部署的、数据安全的**领域模型。

优点：

- 私有化部署
- 数据安全
- 更擅长特定行业

缺点：

- 模型微调成本较高
- 数据更新的实时性较差
- 经验主义

专业化

方案2：LangChain+ChGLM-6B

利用输入的问题，依靠表征寻找到相似的知识段落文本。将相似知识段落文本拼接上问题输入对话式语言模型获取当前问题的文档问答结果。

优点：

- 灵活易扩展
- 私有化部署
- 数据安全

缺点：

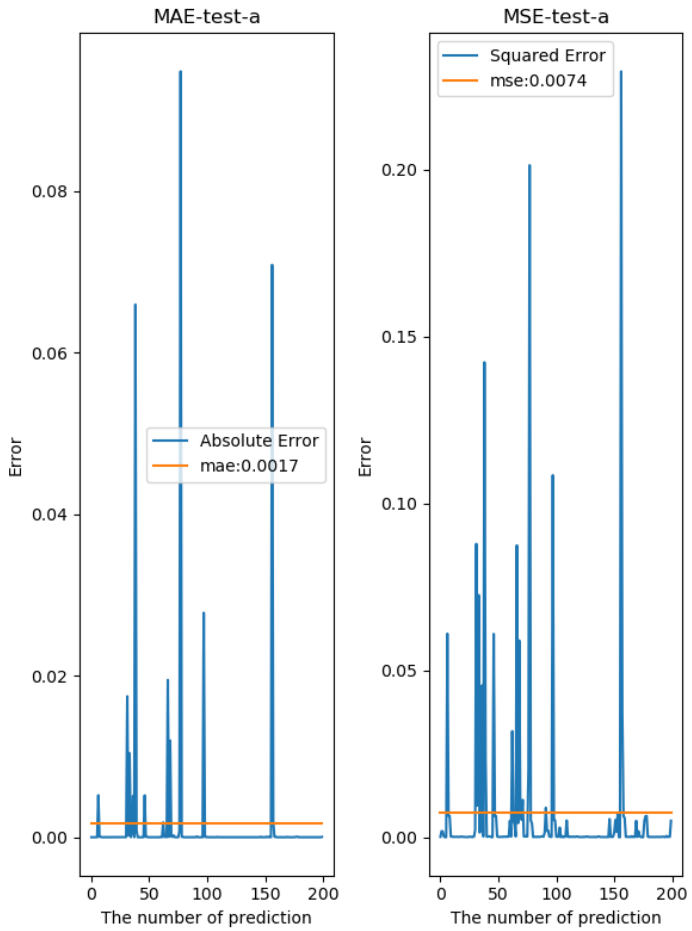
- 依赖知识库建设
- 依赖LLM的基础能力
- 依赖表征模型的能力

平民化

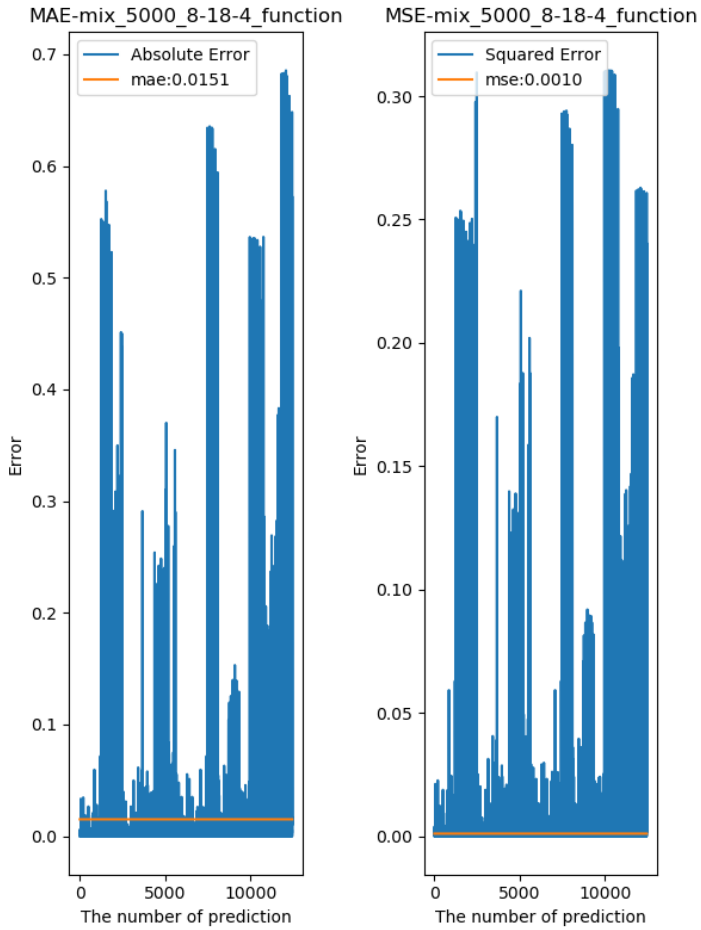
具体使用的例子

```python
openai_api_key="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"
import os
import openai

openai.api_key = openai_api_key

#自定义对话函数
def get_completion(prompt,model="gpt-3.5-turbo"):
    messages = [{"role":"user","content":prompt}]

    response = openai.ChatCompletion.create(
        model = model,
        messages=messages,
        temperature=0
    )
    return response.choices[0].message["content"]

get_completion("1+1是什么？")

"""
返回结果：

'1+1等于2.'
"""


#使用langchain
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage,SystemMessage,AIMessage

chat = ChatOpenAI(temperature=.7,openai_api_key="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")

chat(
    [
        SystemMessage(content="你是一个很棒的粤菜点餐机器人，可以帮助用户在一个简短的句子中弄清楚该吃什么"),
        HumanMessage(content="我喜欢西红柿，我应该吃什么")
    ]
)

"""
返回结果：

AIMessage(lc_kwargs=['content':'你可以试试叫做番茄炒蛋的菜品'],content='你可以试试叫做番茄炒蛋的菜品',additional_
kwargs={},example=False)
"""
```

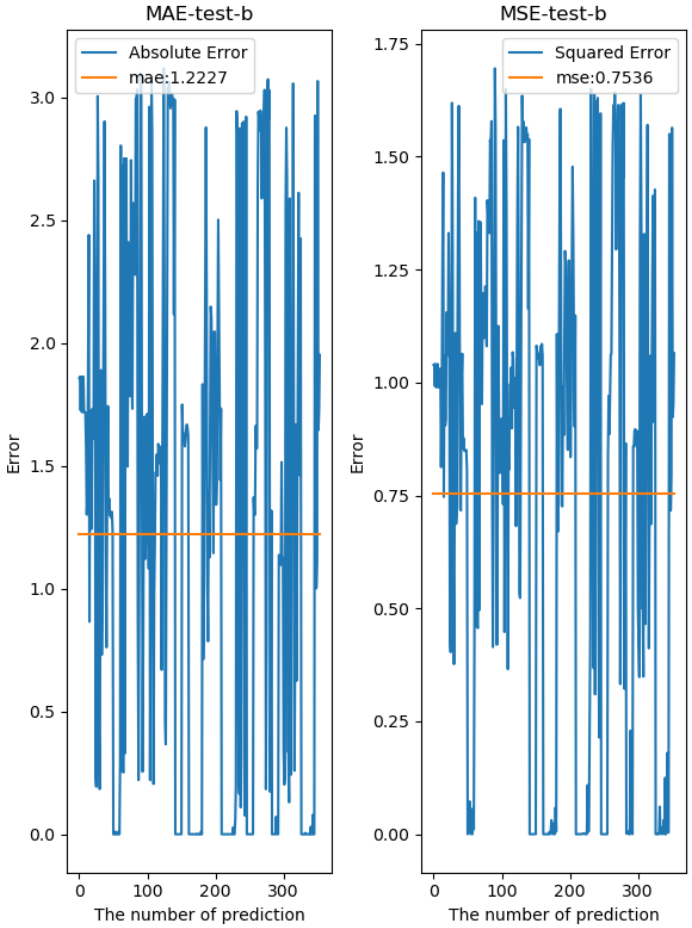
(数据样例1)在见过的任务中粗粒化的测试结果，通过率是100%



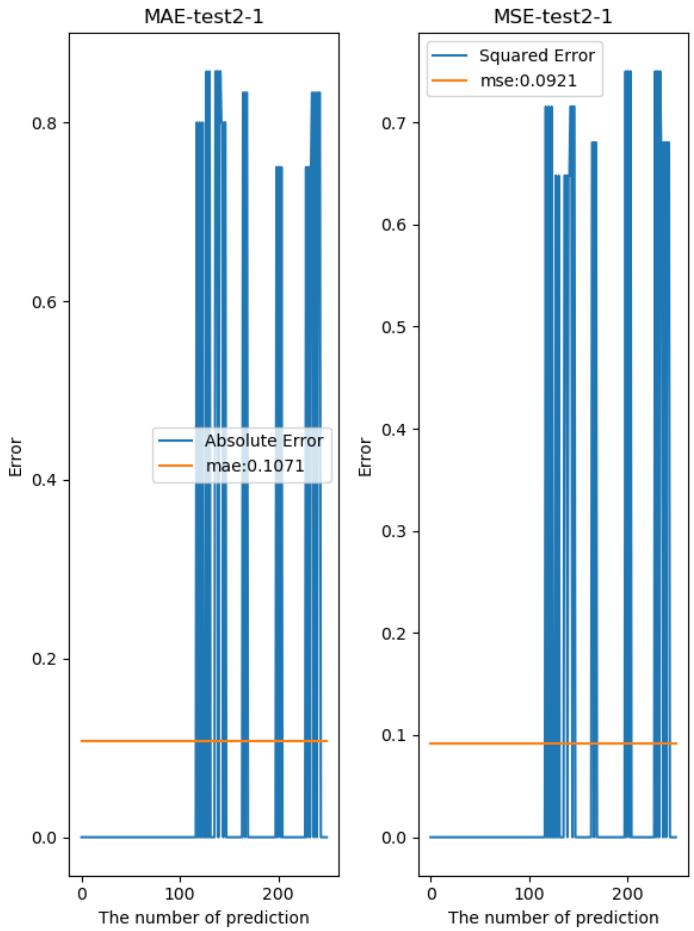
(数据样例1)在见过的任务上未进行粗粒化的测试结果



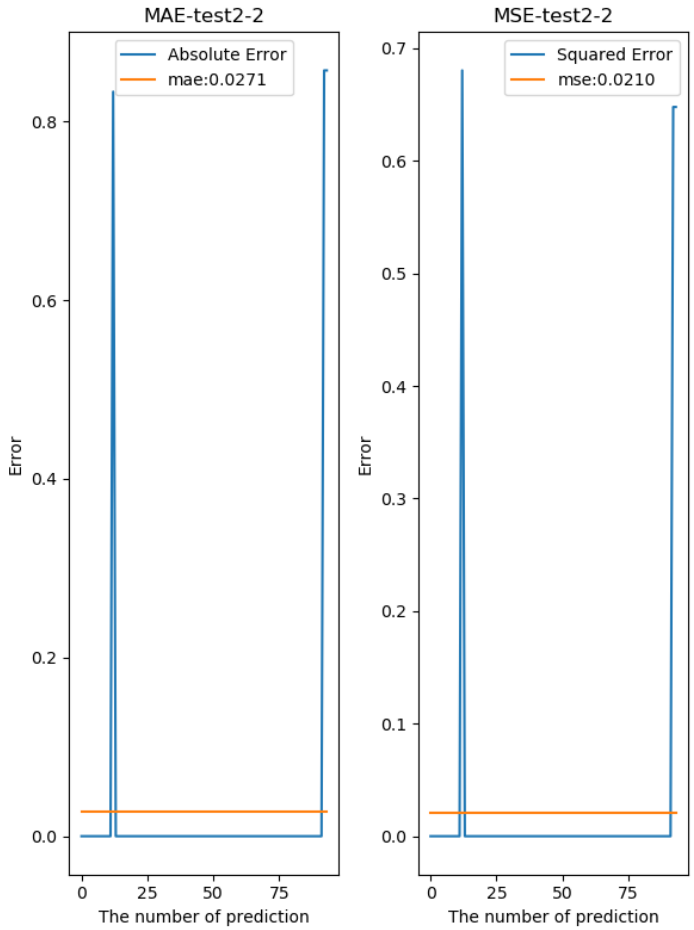
(数据样例1)在没见过的任务上使用粗粒化的测试结果，通过率是88.5%



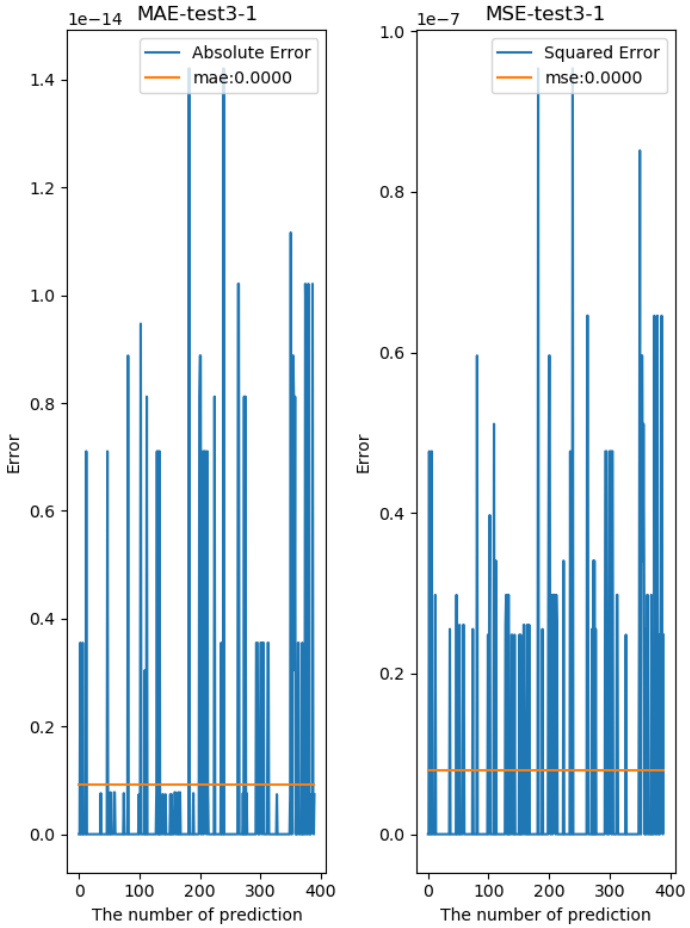
(数据样例2)在见过的任务上使用逐步推导的测试结果,通过率是73.53%



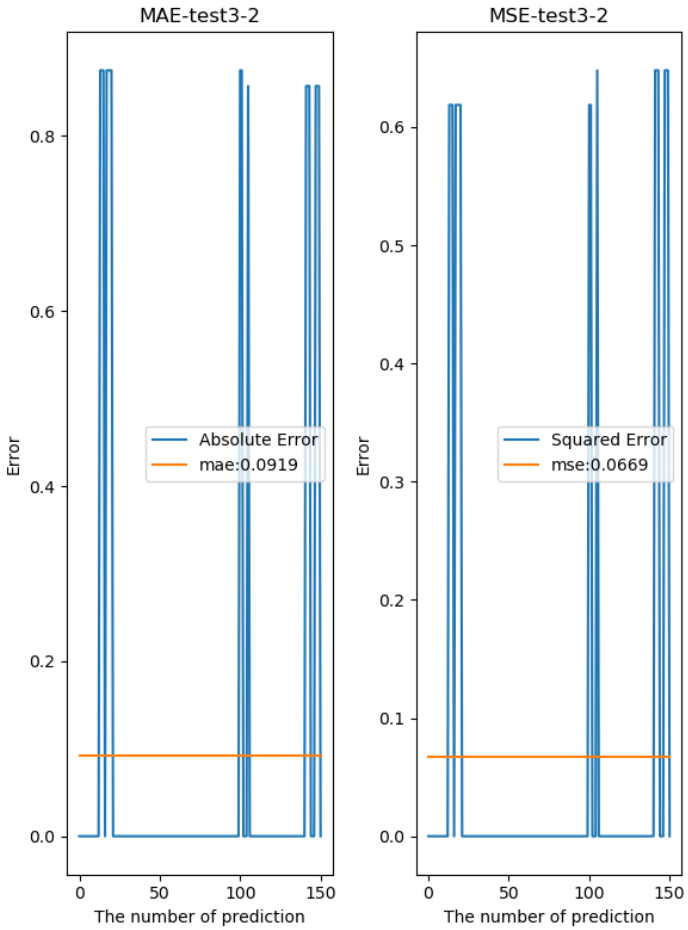
(数据样例2)在未见过的任务上使用逐步推导的测试结果，通过率是50.54%



(数据样例2)在见过的任务上使用粗粒化的测试结果,通过率是97.5%



(数据样例2)在未见过的任务上使用粗粒化的测试结果,通过率是75.5%



解决数学问题的能力

通过使用44个基本的数学问题，包括简单加减乘除，基本应用题等。构建从这44种基本数学问题中随机构造1000个数学问题以及标准答案的推导结果进行测试。将其中的700个数据作为训练集，剩下300个作为测试集。

- 从结果可以看出，chatglm1还是具有解决简单数学问题的能力，但是数的太大的话，结果就会不准确。
- 在一些简单的应用题目上，能够进行一步一步的推导过程，但是得到的结果不一定准确，中间步骤可能会出错。

实验结论

- 使用粗粒化之后可以看到通过率更高和误差更小了。不管对于见过还是没见过的任务,性能的都有提升。
- 但是另一种逐步推导的效果不太显著，可能是因为依旧受到语言模型无法进行前瞻探索的缘故
- 从解决数学能力的实验中可以看到使用Tree-Of-Thought的方式确实能提高解决问题的能力。
- chatglm2在最简单的回答问题上都有很大的问题，所以这个模型暂时无法使用。相比之下chatglm没有这个问题。
- 经过微调后的模型只能解决相关序列预测的问题，会影响回答其他的问题！

下一步计划及相关问题

- 暂时也没啥其他想法提高泛化能力，也许将数据量大幅度提高也能提高他的泛化能力。
- 如何更好的使用粗粒化来进行序列预测?
- 提取模型中的注意力权重，查看模型对于输入信息的处理细节。
- 问题:
  1. 使用llama2的代价是否可能会很大，当前即便是很小的数据集使用曙光运行一次要50元左右。而且如果是llama2的话就无法使用现在的自己的服务器了。
  2. 同一个模型对于已经见过的多个任务均能够进行精确的预测，是否需要在这方面更加完善实验?

谢谢老师和同学们的聆听!