大型语言模型(LLM)

陈钶杰
专业:计算数学

June 8, 2023

目录

LLM模型

- 解决一下几个问题
  - ChatGPT是否带来了NLP乃至AI领域的研究范式转换?
  - LLM从海量数据中学到了什么知识?
  - LLM又是如何存取这些知识的?
  - 随着LLM规模逐步增大，会带来什么影响?
  - 什么是In Context Learning?

ChatGPT是否带来了NLP乃至AI领域的研究范式转换？

目前规模最大的LLM模型，几乎都是类似GPT 3.0这种“自回归语言模型+Prompting”模式的.

原因如下:

1. 可以把分类问题转换成让LLM模型生成对应类别的字符串，这样理解和生成任务在表现形式就实现了完全的统一。同一个LLM生成模型，可以解决几乎所有NLP问题。而如果仍然采取Bert模式，则这个LLM模型无法很好处理生成任务。
2. 如果想要以零示例提示语（zero shot prompting）方式做好任务，则必须要采取GPT模式。有研究证明：如果是以fine-tuning方式解决下游任务，Bert模式的效果优于GPT模式；若是以zero shot/few shot prompting这种模式解决下游任务，则GPT模式效果要优于Bert模式。这说明了，生成模型更容易做好zero shot/few shot prompting方式的任务，而Bert模式以这种方式做任务，是天然有劣势的。

ChatGPT是否带来了NLP乃至AI领域的研究范式转换？

本来我们希望LLM能够用人类常用的命令方式来执行某个任务，但是目前技术还做不到，所以退而求其次，用Instruct技术来表达人类的任务需求。

1. ChatGPT的出现，改变了这个现状，用Instruct取代了Prompting，由此带来新的技术范式转换，并产生若干后续影响.
2. 基本实现了理想LLM的接口层，让LLM适配人的习惯命令表达方式，而不是反过来让人去适配LLM，绞尽脑汁地想出一个能Work的命令（这就是instruct技术出来之前，prompt技术在做的事情），而这增加了LLM的易用性和用户体验。相对之前的few shot prompting，它是一种更符合人类表达习惯的人和LLM进行交互的人机接口技术。

LLM相关的知识

LLM学到了什么知识

1. 语言类知识:词法、词性、句法、语义等有助于人类或机器理解自然语言的知识。
2. 世界知识: 在这个世界上发生的一些真实事件，以及一些常识性知识。

LLM如何存取知识?

1. 知识存储是在Transformer模型的参数里面的,1/3数据在多头注意力,2/3数据在FFN结构中.所以知识主体是存储在FFN结构里面的.

LLM的规模

1. “涌现能力”:指的是当模型参数规模未能达到某个阈值时，模型基本不具备解决此类任务的任何能力，体现为其性能和随机选择答案效果相当，但是当模型规模跨过阈值，LLM模型对此类任务的效果就出现突然的性能增长

人机接口:从in context learning到instruct理解

仔细区分这些不同的人机接口

1. In Context Learning只是拿出例子让LLM看了一眼，并没有根据例子，用反向传播去修正LLM模型参数的动作，就要求它去预测新例子。
2. 早期大家做zero shot prompting，实际上就是不知道怎么表达一个任务才好，于是就换不同的单词或者句子，反复在尝试好的任务表达方式，这种做法目前已经被证明是在拟合训练数据的分布.
3. Instruct的做法则是给定命令表述语句，试图让LLM理解它。所以尽管表面都是任务的表述，和zero shot prompting思路是不同的。

如果大语言模型具备越强的数据压缩能力，是否意味着它具备越强的AGI智能呢?

数据压缩:压缩即智能

- GPT模型训练的过程就是在进行数据压缩,将输入的语句等以参数的形式保存.比如Large Language Model Meta AI(LLAMA)模型的数据压缩率在14倍左右.
  1. "最小描述长度原理"可以用来解释这个观点.
  2. 比如我输入一个长度为1万的质数序列"2,3,5,11,...",那么对于这个输入的最佳解释即要尽可能短而准确的描述这个序列,比如输出"从2开始的一万个连续质数".
- LLM的这种数据压缩能力是无损的.
  1. 此模型能根据上文Context，给出的后续Next Token肯定会有错误,这些被预测错误的Token，其实就代表了LLM压缩数据的信息损失，这种损失是靠算术编码额外对信息进行编码来进行补偿，才达成数据的"无损压缩"效果.即有如下的公式:
  2. 数据无损压缩=LLM模型的有损数据压缩能力+算数编码的补偿能力

大LLM模型和小LLM模型的差异

- 小LLM模型建立了一个粗粒度的,模糊的世界图像,而随着模型规模越来越大,大LLM模型建立起能表征更多细节信息的清晰度越来越高的世界图像.

回路竞争视角下的tuning

- "回路竞争"猜想:如果从低向上激发过程中，我们希望的正确回路被激发，可以认为回路竞争胜利，则模型输出正确答案，而如果错误任务回路被激发，可以认为回路竞争失败，则模型输出错误答案。
- 可能通过Fine-tuning操作，在模型内部建立起了Shortcut捷径，导致输入信息后，信息传输直接走了捷径，而绕过了很多本该要走的通路。
- 对于instruct tuning就是创造了一条新的特殊的激活回路,输入命令自身形成的激活回路,建立起和对应任务回路的连接.

大模型参数高效微调(PEFT)

PEFT主要的两类方法,不同的方法对PLM 的不同部分进行下游任务的适配

- Prompt-Tuning(Prefix-Tuning):在模型的输入或隐层添加k个额外可训练的前缀tokens（这些前缀是连续的伪tokens，不对应真实的tokens），固定PLM 的所有参数，只训练这些前缀参数；
- Adapter-Tuning：则是在预训练模型内部的网络层之间添加新的网络层或模块来适配下游任务。比如在transformer中Multi-Head层和FFN层中间插入一个Adapter层,固定其他参数,只调整Adapter层的参数

代码生成大模型的挑战和机遇

- 理解能力:人类能够理解不同抽象层次的各种描述，相比之下，当前的LLM 往往对给定的上下文敏感，这可能会导致性能下降.
- 判断能力: 人类能够判定一个编程问题是否被解决。当前模型不论输入什么都会给出答案，而且该答案正确与否都不能确定，这在实际应用中会存在一定的问题。
- 解释能力:人类开发人员能够解释他们编写的代码，这对教育的和软件维护至关重要。
- 自适应学习能力:当前的大型语言模型与人类之间的一个根本区别是它们适应新知识和更新知识的能力。
- 多任务处理能力:代码大模型可以应用到各种各样和代码相关的任务中，例如代码修复，代码搜索，代码审核等。甚至代码大模型可以解决所有可以形式化为代码形式的下游任务。

经过微调的LLaMA在算术任务上的性能优于GPT-4

1. 我们的模型在各种基本算术任务上实现了最先进的性能，包括加法、减法、乘法，以及正整数的除法.我们展示了在综合生成的数据集上与GPT-4相比，在算术任务上微调的开源模型实现更高精度的潜力.
2. 证明监督的可行性,单独的微调可以使LLM为某些基本算术任务生成直接答案，例如大量加法和减法，而不应用任何特殊技术。令人印象深刻的性能主要归功于LLaMA数字的一致标记化。
3. 为了解决大量的乘法和除法问题，我们提出了一种新的分解方法方法基于任务的可学习性，利用基本的算术原理来确保人类的可解释性.
4. 我们系统地调查了分解方法及其证明有效性。我们对分解步骤进行了彻底的实验在一个完全合成的环境中，通过减轻自然的许多难以控制的方面语言我们的实验设置提供了研究CoT和中间监督.
5. 我们的端到端指令调优管道可以可以很容易地集成到现有的指令调优语言模型中，并可能增强他们对数学单词的数学推理问题。

代码

- 使用t5-base(2.8亿参数量,大小为1.3G)数据进行指令微调

```
train_texts = [
    ("Is apple used to mean the same thing in the next two sentences (see options)?\n\nI like apple.\n\nHe ate
    an apple.\n\n{options_}", "Yes"),
    ("What is the capital city of Germany?", "Berlin"),
    ("Calculate the sum of 5 and 3.", "8")
]

train_targets = [
    "No", "Yes", "Not sure",
    "Paris", "Berlin", "London",
    "sum", "product", "difference"
]
```

```
Input: Is apple used to mean the same thing in the next two sentences (see options)?

I like apple.

He ate an apple.

{options_}
Output: is apple used to mean the same thing in the next two sentences? Is apple used to mean the same th
ing in the next two sentences (see options)? I like apple. He ate an apple.
```

- 只是重复了两遍.

代码

- 时间序列的指令微调代码(t5_finetuning.py)
  1. 程序有一些bug没跑通.

```
        File "/tmp/__autograph_generated_filev6e6abma.py", line 64, in tf__call
            ag__.if_stmt(ag__.and_((lambda : (ag__.ld(input_ids) is not None)), (lambda : (ag__.ld(inputs
_embeds) is not None))), if_body_2, else_body_2, get_state_2, set_state_2, ('input_ids', 'input_shape'),
2)
        File "/tmp/__autograph_generated_filev6e6abma.py", line 61, in else_body_2
            ag__.if_stmt((ag__.ld(input_ids) is not None), if_body_1, else_body_1, get_state_1, set_state
_1, ('input_ids', 'input_shape'), 2)
        File "/tmp/__autograph_generated_filev6e6abma.py", line 58, in else_body_1
            ag__.if_stmt((ag__.ld(inputs_embeds) is not None), if_body, else_body, get_state, set_state,
('input_shape',), 1)
        File "/tmp/__autograph_generated_filev6e6abma.py", line 55, in else_body
            raise ag__.converted_call(ag__.ld(ValueError), (f'You have to specify either {ag__.ld(err_msg
_prefix)}input_ids or {ag__.ld(err_msg_prefix)}inputs_embeds',), None, fscope)

        ValueError: Exception encountered when calling layer 'decoder' (type TFT5MainLayer).

        in user code:

            File "/home/chenkejie/python_environment/t5_model/lib/python3.8/site-packages/transformers/mo
deling_tf_utils.py", line 1351, in run_call_with_unpacked_inputs  *
                return func(self, **unpacked_inputs)
            File "/home/chenkejie/python_environment/t5_model/lib/python3.8/site-packages/transformers/mo
dels/t5/modeling_tf_t5.py", line 677, in call  *
                raise ValueError(f"You have to specify either {err_msg_prefix}input_ids or {err_msg_prefi
x}inputs_embeds")

            ValueError: You have to specify either decoder_input_ids or decoder_inputs_embeds


        Call arguments received by layer 'decoder' (type TFT5MainLayer):
          • input_ids=None
          • attention_mask=None
          • encoder_hidden_states=tf.Tensor(shape=(None, 23, 512), dtype=float32)
          • encoder_attention_mask=tf.Tensor(shape=(None, 23), dtype=int32)
          • inputs_embeds=None
          • head_mask=None
          • encoder_head_mask=None
          • past_key_values=None
          • use_cache=True
          • output_attentions=False
          • output_hidden_states=False
          • return_dict=True
          • training=True


    Call arguments received by layer 'tft5_for_conditional_generation' (type TFT5ForConditionalGeneration
):
      • input_ids={'input_ids': 'tf.Tensor(shape=(None, 23), dtype=int32)', 'attention_mask': 'tf.Tensor(
shape=(None, 23), dtype=int32)'}
      • attention_mask=None
      • decoder_input_ids=None
      • decoder_attention_mask=None
      • head_mask=None
      • decoder_head_mask=None
      • encoder_outputs=None
      • past_key_values=None
      • inputs_embeds=None
```

代码

- 时间序列的指令微调代码(t5_finetuning.py)
  1. 在使用CUDA加速计算时，显存（GPU内存）不足以分配所需的张量。
  2.
```
  File "/media/data/chenkejie/python_environment/goat/lib/python3.8/site-packages/torch/nn/funct
ional.py", line 1845, in softmax
    ret = input.softmax(dim, dtype=dtype)
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 326.00 MiB (GPU 0; 10.75 GiB
total capacity; 8.65 GiB already allocated; 170.69 MiB free; 9.81 GiB reserved in total by PyTor
ch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentati
on.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

尝试运行如下几个程序:

1. Awesome-instruction-tuning:一个精心策划的开源指令调整数据集、模型、论文、资料库的列表。
   网址:https://github.com/zhilizju/Awesome-instruction-tuning
2. stanford_alpaca:该项目旨在建立和分享一个遵循指令的LLaMA模型。
   网址:https://github.com/tatsu-lab/stanford_alpaca
3. Chinese-Vicuna:该项目旨在建立和分享遵循指令的中文LLaMA模型调整方法
   网址:https://github.com/Facico/Chinese-Vicuna
4. FLAN:这个资源库包含了生成指令调谐数据集的代码。
   网址:https://github.com/google-research/FLAN

谢谢老师和同学的聆听!