LLM输出的确定性与随机性

1. LLM回答的确定性与随机性

在同样的prompt输入条件下，LLM输出回答既可以是确定性的也可以是随机性的

LLM在输入prompt后，系统生成回答文本分为两个阶段，

（1）第一阶段：LLM处理输入token,生成输出token的score分布，在输入不变的情况下，输出的score分布也是不变的

（2）第二阶段：多种解码策略处理score分布，如果采用确定性策略，那回答就是确定性的，如果采用随机性策略，那回答就具有随机性

2. LLM输入

基于next one token训练的模型属于因果语言模型(causal language modeling),LLM接受的只是文本，聊天消息,代码等只是一种特殊的文本形式,基于LLM的任何任务一定程度上都可以认为是对输入文本理解续写

（1）一般文本输入：一般输入的文本经过token化就可以输入给模型

（2）聊天消息输入：聊天消息经过聊天模板序列化为文本经token后输入给聊天模型,常用聊天模板ChatML格式,<I im_start I>标记消息开始,<I im_end I>标记消息结束

```
"{% for message in messages %}"
"{{'<|im_start|>'+message['role']+'\n'+message['content']+'<|im_end|>'+'\n'}}"
"{% endfor %}"
"{% if add_generation_prompt %}"
"{{ '<|im_start|>assistant\n' }}"
"{% endif %}"
```

Jinja模板语言ChatML格式聊天模板

```
messages = [
{"role": "system", "content": "You are a helpful assistant."}
{"role": "user", "content": "你是谁"}
]
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,add_generation_prompt=True)
```

```
Run: test_llm
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
你是谁<|im_end|>
<|im_start|>assistant
```

聊天消息经聊天模板转为text样例

3. LLM输出

LLM在输出token socre分布后需要解码策略确定采样的token,目前主要使用多策略联合采样(top-k,top-p,temperature)

LLM常用解码策略如下:

确定性策略:

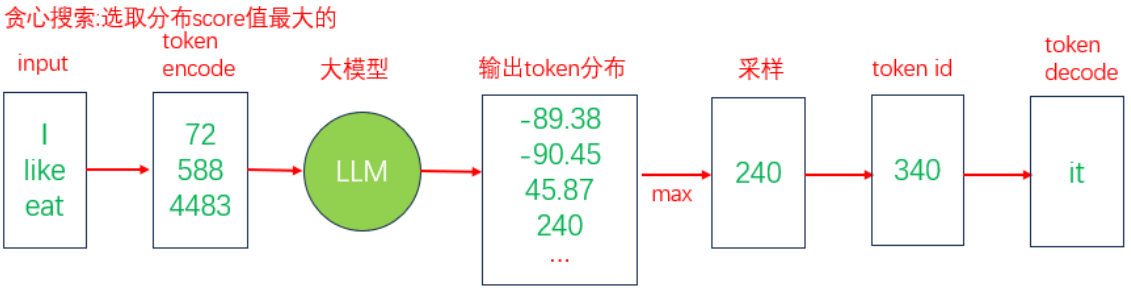
1.greedy search(贪心搜索)：每一步都取概率最大的token,直接选择模型输出token分布值最大的值,不用做softmax

优点：计算简单高效

缺点:

1.容易产生重复，文本不连贯,多样性不高

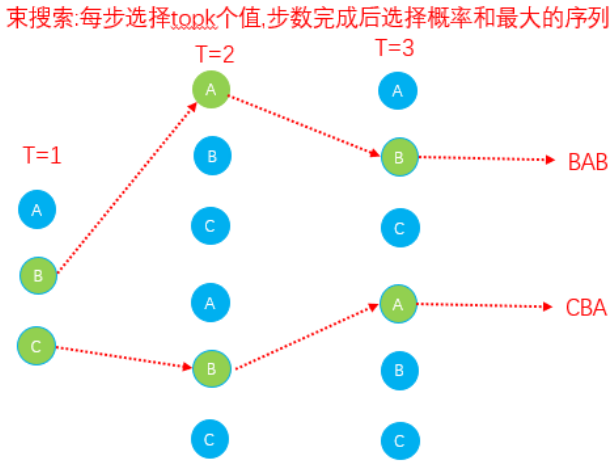
2.每步选择局部最优,可能错过全局最优



2.beam search(束搜索):每一步保留topk个概率值大的输出,模型输出token分布值经过log_softmax后保留,步数完成后最终选择概率和最大的路径序列,k=1就退化成贪心搜索

优点：属于启发式搜索,适用于解空间较大情况，一定程度可以保证最终序列概率最优

缺点：1.每步选择局部最优,可能错过全局最优 2.有可能出现重复,前后矛盾情况 3.计算量随序列长度指数级增长



随机性策略:

3.随机采样:随机选择一个token

优点：简单,多样性高

缺点：容易产生重复，文本不连贯

4.top-k采样：常用范围值5-50,每一步选取概率最高的k个token作为候选，再抽样选择一个，分布概率大的token采样到的概率越大

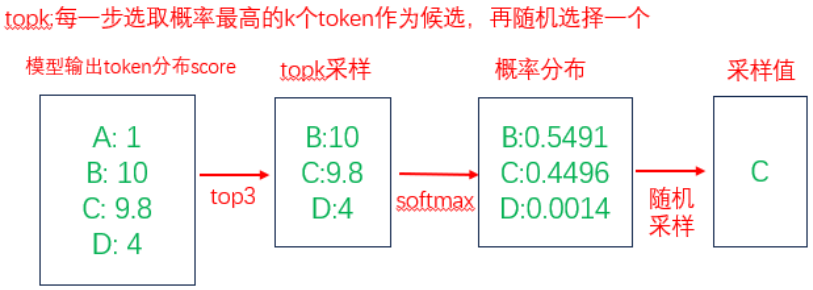
优点：多样性会提升,有助于增强文本的连贯性，减少出现不常见或者与上下文无关的词,随机性会有助于提高生成质量

缺点：

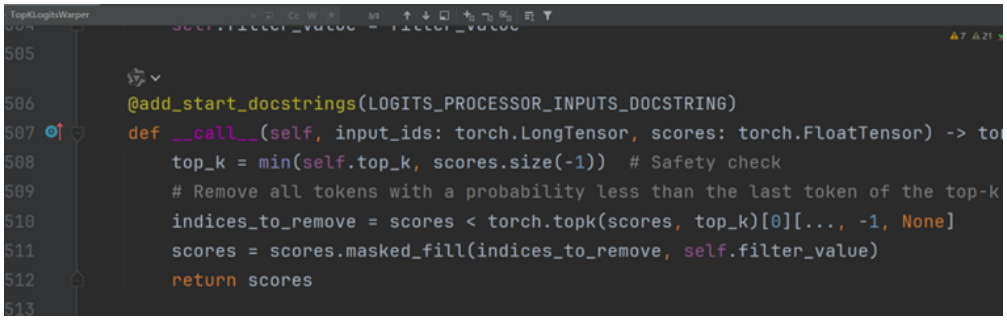
1.k值固定不动态，k值难确定

2.分布陡峭可能会采样到概率小的单词，分布平缓只能采样部分可用单词，

k=1时就退化成贪心搜索，可能会导致文本不符合常识逻辑或者简单无聊



相关代码：



5.top-p采样：常用范围值0.9-0.95,从累计概率和大于等于p的最小集合抽样选择一个，top-p常与top-k结合使用,如果k,p都启用,则p在k之后起作用,随机采样时分布概率大的token采样到的概率越大

优点：

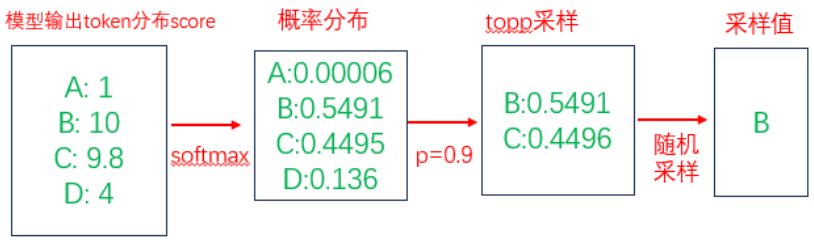
1.动态设置token候选列表大小

2.过滤掉低概率的token,

3.top-p越大多样性越丰富

缺点：p太小时模型输出越固定,低概率但有意义或创意的词被过滤掉

top-p:从累计概率大于等于p的最小集合抽样选择一个



相关代码：

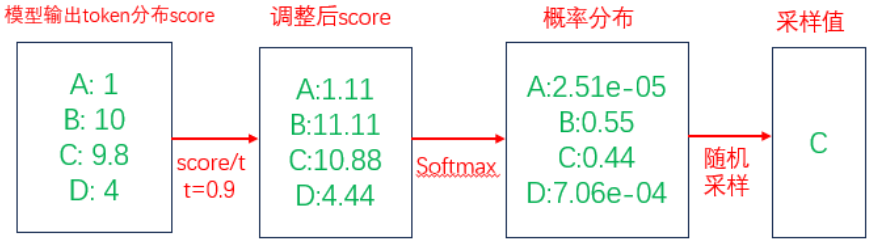
```
@add_start_docstrings(LOGITS_PROCESSOR_INPUTS_DOCSTRING)
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> torch.F
    sorted_logits, sorted_indices = torch.sort(scores, descending=False)
    cumulative_probs = sorted_logits.softmax(dim=-1).cumsum(dim=-1)

    # Remove tokens with cumulative top_p above the threshold (token with 0 are kept)
    sorted_indices_to_remove = cumulative_probs <= (1 - self.top_p)
    # Keep at least min_tokens_to_keep
    sorted_indices_to_remove[..., -self.min_tokens_to_keep_:] = 0

    # scatter sorted tensors to original indexing
    indices_to_remove = sorted_indices_to_remove.scatter(1, sorted_indices, sorted_in
    scores = scores.masked_fill(indices_to_remove, self.filter_value)
    return scores
```

6.temperature采样:常用范围值(0,1],通过温度调整token的score分布,温度越低，分布差距越大，越容易采样到概率大的token，温度越高，分布差距越小，低概率token采样机会增大,prompt越长越清晰，模型输出质量越好,可以适当提高温度值增加多样性,prompt越短越不清晰,高的温度值，输出就不稳定,随机采样时分布概率大的token采样到的概率越大

优点：可用调整分布概率控制多样性和稳定性

缺点：温度越高多样性,创意性越强,但也有可能产生错误或者不连贯，温度越低越保守稳定，容易出行重复

temperature:通过温度参数调节输出token的分布score



相关代码：

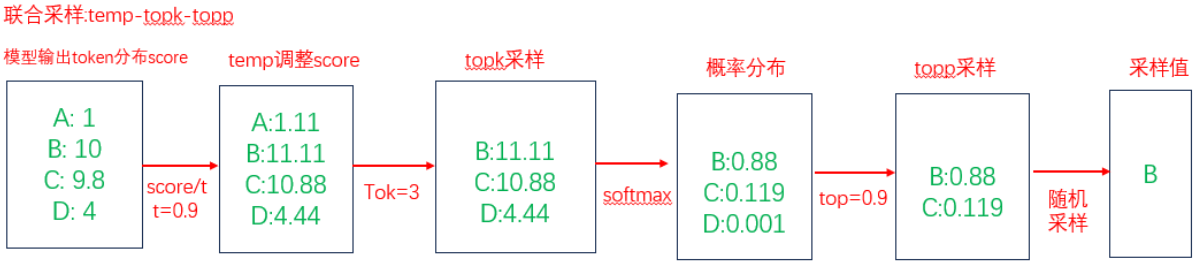
```
@add_start_docstrings(LOGITS_PROCESSOR_INPUTS_DOCSTRING)
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor)
    scores = scores / self.temperature
    return scores
```

7.联合采样(top-k,top-p,temperature):多策略并行,使用先后顺序temp-topk-topp

1.首先使用temperature调整模型输出的token score分布

2.再选取score大的topk个token

3.再从k个token中选择概率累计和达到top-p的token



相关代码:

```
# all samplers can be found in `generation_utils_samplers.py`
if generation_config.temperature is not None and generation_config.temperature != 1.0:
    warpers.append(TemperatureLogitsWarper(generation_config.temperature))
if generation_config.top_k is not None and generation_config.top_k != 0:
    warpers.append(TopKLogitsWarper(top_k=generation_config.top_k, min_tokens_to_keep=m
if generation_config.top_p is not None and generation_config.top_p < 1.0:
    warpers.append(TopPLogitsWarper(top_p=generation_config.top_p, min_tokens_to_keep=m
if generation_config.typical_p is not None and generation_config.typical_p < 1.0:
    warpers.append(
        TypicalLogitsWarper(mass=generation_config.typical_p, min_tokens_to_keep=min_to
```

常用惩罚策略:

重复token惩罚:为了控制减少重复token出现,主要策略如下:

1. repetition_penalty:惩罚系数,对输出的token 已经在input里面token进行score惩罚

   if repetition_penalty >1:减少重复词的生成概率

   if repetition_penalty =1:保持原有生成概率

   if repetition_penalty <1:增加重复词的生成概率

```
@add_start_docstrings(LOGITS_PROCESSOR_INPUTS_DOCSTRING)
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> t
    score = torch.gather(scores, 1, input_ids)

    # if score < 0 then repetition penalty has to be multiplied to reduce the t
    score = torch.where(score < 0, score * self.penalty, score / self.penalty)

    scores.scatter_(1, input_ids, score)
    return scores
```

2. no_repeat_ngram_size:生成文本时需要避免的ngram文本的大小,通过获取当前token的ngram,把模型输出scores对应的ngram token的值设置为无穷小,极大减低采样到ngram token的概率,所以基本ngram只能出现一次,谨慎使用

```
@add_start_docstrings(LOGITS_PROCESSOR_INPUTS_DOCSTRING)
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> torch.Float
    num_batch_hypotheses = scores.shape[0]
    cur_len = input_ids.shape[-1]
    #获取当前token的ngram元素list,并且把scores对应的token score设置为无穷小,排除这些ngram token
    banned_batch_tokens = _calc_banned_ngram_tokens(self.ngram_size, input_ids,
                                                    num_batch_hypotheses, cur_len)
    for i, banned_tokens in enumerate(banned_batch_tokens):
        scores[i, banned_tokens] = -float("inf")

    return scores
```

长度惩罚：

1. exponential_decay_length_penalty ： 当前长度超过设定的生成长度时,通过增加eos结束标记的score值，让采样到eos概率增大，达到生成结束的目的

```
@add_start_docstrings(LOGITS_PROCESSOR_INPUTS_DOCSTRING)
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) ->
    cur_len = input_ids.shape[-1]
    if cur_len > self.regulation_start:
        for i in self.eos_token_id:
            penalty_idx = cur_len - self.regulation_start
            # To support negative logits we compute the penalty of the absolu
            scores[:, i] = (scores[:, i] + torch.abs(scores[:, i]) *
                            (pow(self.regulation_factor, penalty_idx) - 1))
    return scores
```

4. LLM生成停止策略

停止主要使用eos_token特殊标记,由模型决定何时结束,max策略主要防止大模型停不下来

1.eos_token特殊标记：在生成过程中遇到停止特殊token,就结束生成

2.max_length:最大生成token长度

3.max_new_tokens:除input token长度之外新生成的最大长度,max_length= max_new_tokens + input_ids_length

4.max_time:生成token的最大时间限制范围内,比如设置为60s，那过了60s停止生成