

【手撕LLM-Generation】Top-K+重复性惩罚



小冬瓜AIGC

原创课程  公众号：手撕LLM

 来自专栏 · 手撕LLM >

163 人赞同了该文章 >

我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道：[手撕LLM+RLHF](#)
研究方向：LLM、RLHF、Safety、Alignment

本文从代码实现角度阐明LLM Generation

1. GPT-2 Top-K Sampling⁺

Radford, Alec, et al. "Language models are unsupervised multitask learners." *OpenAI blog* 1.8 (2019): 9.

GPT2论文使用"**Top-k random sampling**" 做Summarization的任务中，优于greedy decoding，具备更有概括性的summarization和**减少重复性"repetition"**

3.6. Summarization

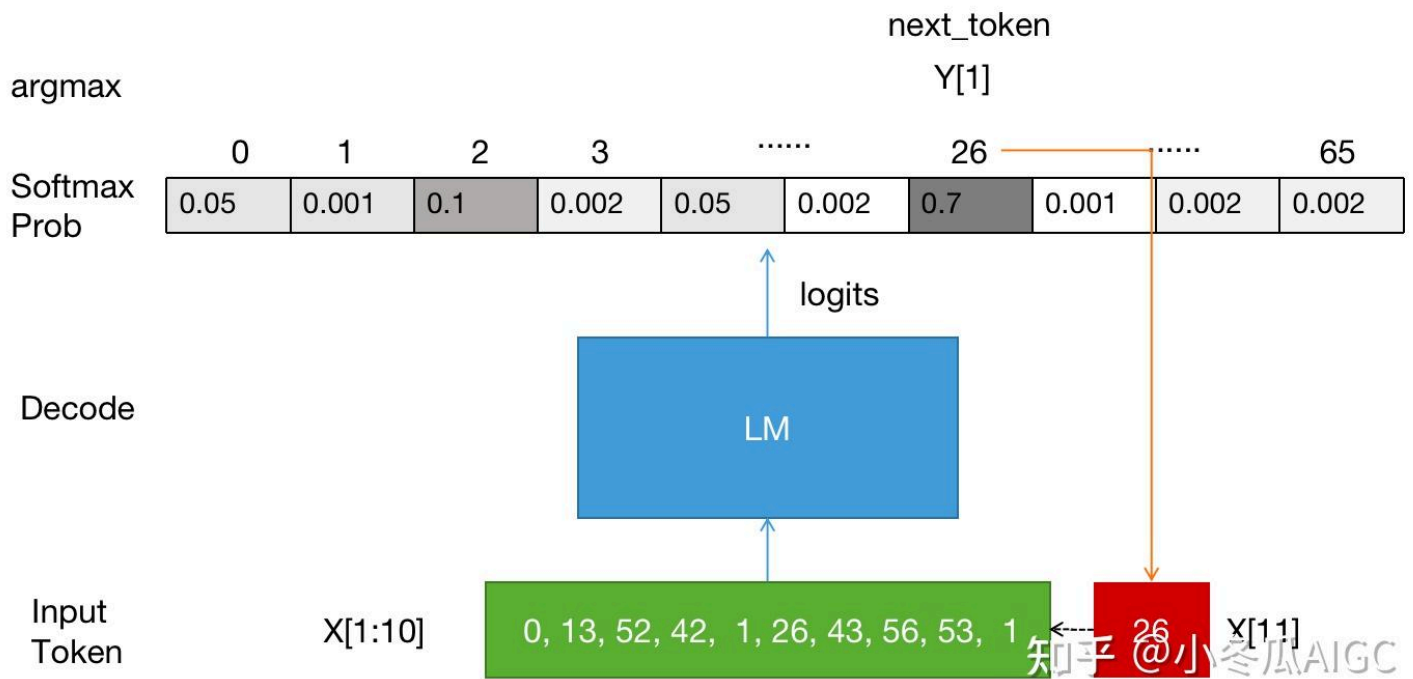
We test GPT-2's ability to perform summarization on the CNN and Daily Mail dataset (Nallapati et al., 2016). To induce summarization behavior we add the text `TL;DR:` after the article and generate 100 tokens with Top- k random sampling (Fan et al., 2018) with $k = 2$ which reduces repetition and encourages more abstractive summaries than greedy decoding. We use the first 3 generated sentences in these 100 tokens as the summary. While qualitatively the generations resemble summaries, as shown in Table 14, they often focus on recent content from the article or confuse specific details such as how many cars were involved in a crash or whether a logo was on a hat or shirt. On the commonly reported ROUGE 1,2,L metrics the generated summaries only begin to approach the performance of classic neural baselines and just barely outperforms selecting 3 random sentences from the article. GPT-2's performance drops by 6.4 points on the aggregate metric when the task hint is removed which demonstrates the ability to invoke task specific behavior in a language model with natural language. 知乎 @小冬瓜AIGC

2. Greedy Generation

2.1 Greedy Generation图解

通常Transformer预测时, 使用 `torch.argmax` 取最大概率的下标做为预测的Token

- 输入idx长度为10
- 计算 `logits = model(idx)`
- 计算 `next_token = argmax(softmax(logits))` # 1个下标
- 此时Token序列长度为11



2.2 Greedy Generation PyTorch代码

```
# Greedy Generation
idx = X[1,:10].reshape(1, 10) # generate 1
print("预测词表大小", model.config.vocab_size) # BPE词表
print("输入X:", idx)
print("输入X长度:", idx.shape)
for _ in range(256):
    logits, _ = model(idx)
    print(' 输出Logits:', logits.shape)
    probs = F.softmax(logits, dim=-1)
    print(' 输出Probs:', probs.shape)
    print(probs)
    idx_next = torch.argmax(probs, dim=2)
    print(' 输出下一个Token:', idx_next.shape)
    print(idx_next)
    idx = torch.cat((idx, idx_next), dim=1)
    print("当前的token长度:", len(idx[0]))
    print("当前的token序列:", idx)
    break
```

预测词表大小 65

输入X: tensor([[0, 13, 52, 42, 1, 26, 43, 56, 53, 1]])

输入X长度: torch.Size([1, 10])

输出Logits: torch.Size([1, 1, 65])

输出Probs: torch.Size([1, 1, 65])

tensor([[[[0.0062, 0.0275, 0.0166, 0.0274, 0.0175, 0.0077, 0.0087, 0.0104,
0.0084, 0.0044, 0.0213, 0.0083, 0.0175, 0.0155, 0.0071, 0.0060,
0.0087, 0.0103, 0.0327, 0.0079, 0.0203, 0.0128, 0.0102, 0.0126,
0.0119, 0.0353, 0.0505, 0.0114, 0.0113, 0.0139, 0.0036, 0.0428,
0.0260, 0.0215, 0.0159, 0.0027, 0.0152, 0.0133, 0.0159, 0.0050,
0.0094, 0.0107, 0.0136, 0.0114, 0.0143, 0.0114, 0.0144, 0.0191,
0.0242, 0.0160, 0.0087, 0.0128, 0.0150, 0.0392, 0.0149, 0.0042,
0.0122, 0.0142, 0.0141, 0.0113, 0.0131, 0.0109, 0.0324, 0.0200,
0.0104]]], grad_fn=<SoftmaxBackward0>)

输出下一个Token: torch.Size([1, 1])

tensor([[26]])

当前的token长度: 11

当前的token序列: tensor([[0, 13, 52, 42, 1, 26, 43, 56, 53, 1, 26]])

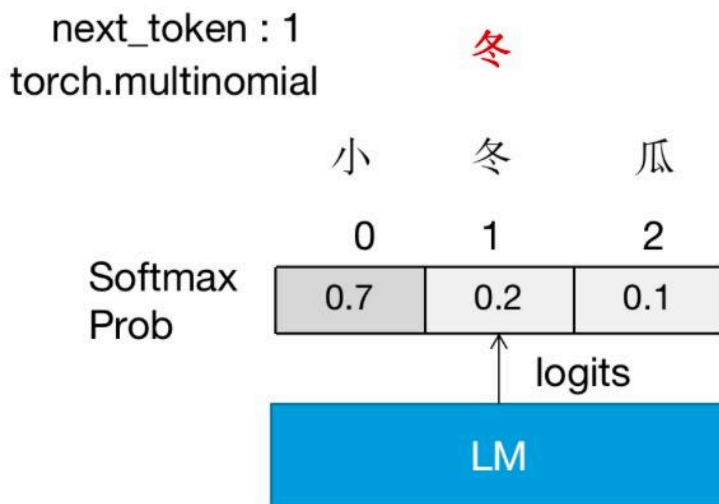
该图修正下，最大值是0.0353而是旁边0.0505

3. Sampling

3.1 Sample Multinomial

在Greedy方式中，会取最大概率值的token作为预测值。而使用sample generation

非最大概率值的token有概率能被采样到。"冬"的token的概率只有0.2，仍有几率被采样(torch.multinomial)到。




```

1 # top-k
2 dict_map = {0:"小", 1:"冬", 2:"瓜"}
3 prob = torch.tensor([0.7, 0.2, 0.1])
4 print("prob : ",prob)
5 |
6 print("根据概率权重所选择next token的下标")
7 for i in range(10):
8     next_token = torch.multinomial(prob, num_samples=1)[0]
9     print(f"第{i}次sample的next_token下标为{next_token}:",
10         dict_map[int(next_token)])

```

prob : tensor([0.7000, 0.2000, 0.1000])

根据概率权重所选择next token的下标

第0次sample的next_token下标为1: 冬

第1次sample的next_token下标为0: 小

第2次sample的next_token下标为2: 瓜

第3次sample的next_token下标为0: 小

第4次sample的next_token下标为0: 小

第5次sample的next_token下标为0: 小

第6次sample的next_token下标为0: 小

第7次sample的next_token下标为0: 小

第8次sample的next_token下标为2: 瓜

第9次sample的next_token下标为0: 小

知乎 @小冬瓜AIGC

此时只需要用 multinomial 替换 argmax 即可实现sampling

```

# Generation sample
idx = X[1, :10].reshape(1, 10) # generate 1
for _ in range(256):
    logits, _ = model(idx)
    probs = F.softmax(logits, dim=-1)
    # idx_next = torch.argmax(probs, dim=2)
    idx_next = torch.multinomial(probs, num_samples=1) # do sample
    idx = torch.cat((idx, idx_next), dim=1)

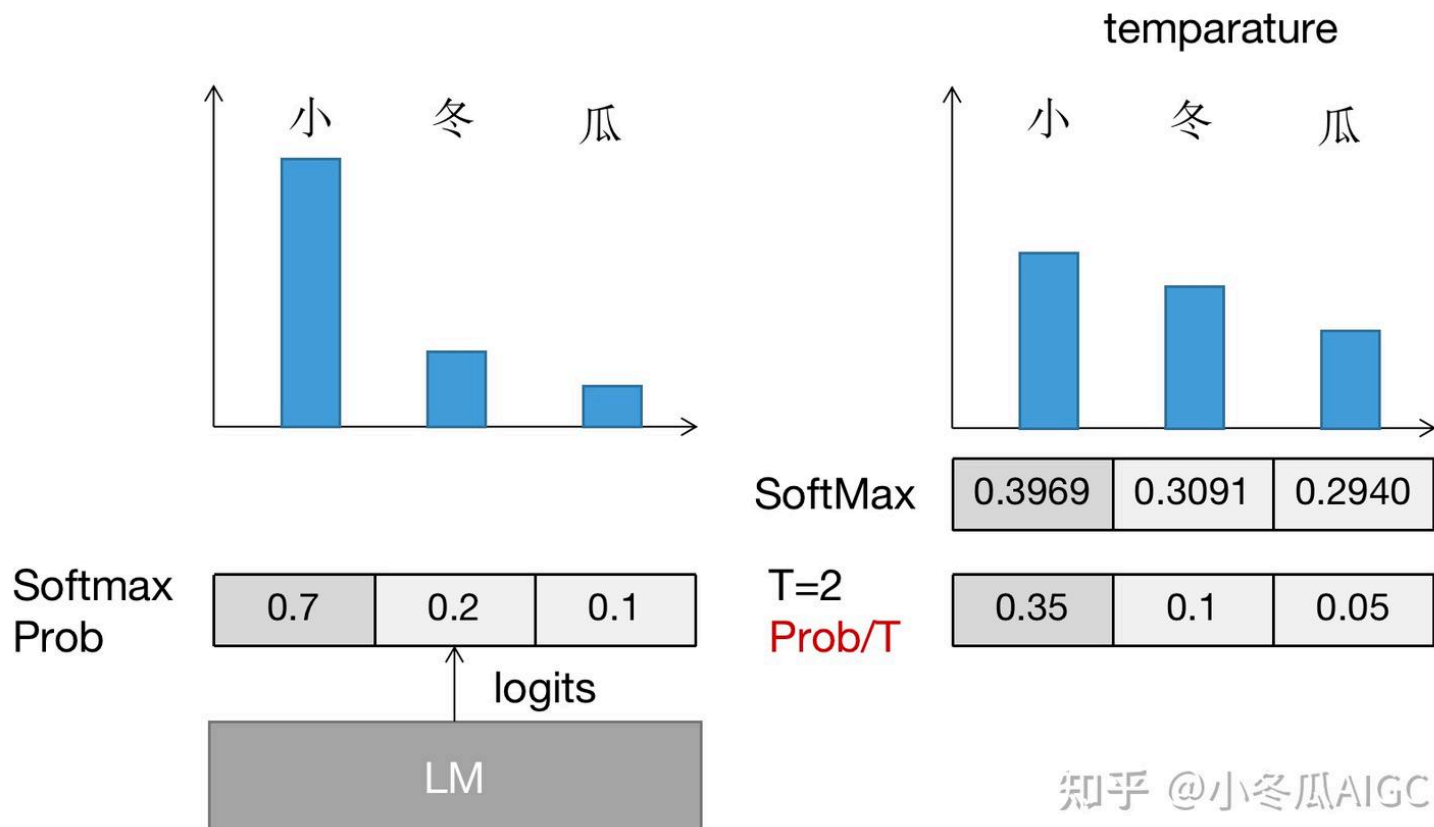
```

torch.multinomial的说明

Returns a tensor where each row contains `num_samples` indices sampled from the multinomial probability distribution located in the corresponding row of tensor `input`.

3.2 Temperature 平滑分布

在LM Generation过程中会得到较sharp的预测分布，此时通过Temperature可以使得预测的分布变得更加平滑，这样就增加了其他token预测概率，更有可能被选择到。



```

1 # Do Sample with temperature
2 temperature = 2.0
3 dict_map = {0:"小", 1:"冬", 2:"瓜"}
4 prob = torch.tensor([0.7, 0.2, 0.1])
5 print("prob : ", prob)
6 prob /= temperature
7 print("prob/T : ", prob)
8 prob = F.softmax(prob)
9 print("softmax(prob/T) : ", prob)
10
11 print("根据概率权重所选择next token的下标")
12 for i in range(10):
13     next_token = torch.multinomial(prob, num_samples=1)[0]
14     print(f"第{i}次sample的next_token下标为{next_token}:",
15           dict_map[int(next_token)])

```

```

prob : tensor([0.7000, 0.2000, 0.1000])
prob/T : tensor([0.3500, 0.1000, 0.0500])
softmax(prob/T) : tensor([0.3969, 0.3091, 0.2940])

```

根据概率权重所选择next token的下标

```

第0次sample的next_token下标为2: 瓜
第1次sample的next_token下标为2: 瓜
第2次sample的next_token下标为1: 冬
第3次sample的next_token下标为2: 瓜
第4次sample的next_token下标为1: 冬
第5次sample的next_token下标为1: 冬
第6次sample的next_token下标为2: 瓜
第7次sample的next_token下标为1: 冬
第8次sample的next_token下标为0: 小
第9次sample的next_token下标为0: 小

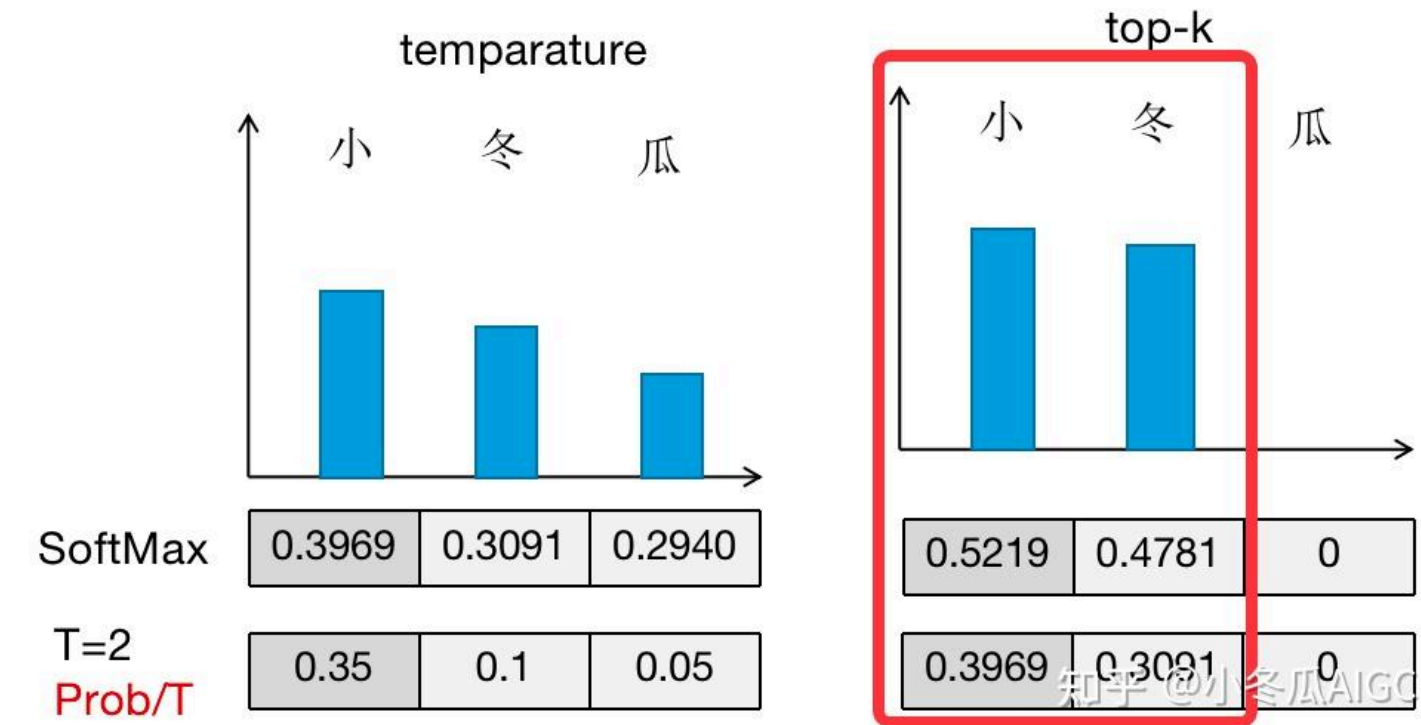
```

4 Top-K Sampling

4.1 Tok-K Sampling图解和代码

generation时，不希望概率过低的被采样到。则将top-k外的token概率置为0

以下示例仅取top-2时，token “瓜” 永远不会被采样到



```

1 # top-k
2 temperature = 2.0
3 top_k = 2
4 dict_map = {0:"小", 1:"冬", 2:"瓜"}
5 prob = torch.tensor([0.7, 0.2, 0.1])
6 print("prob : ", prob)
7 prob /= temperature
8 print("prob/T : ", prob)
9 prob = F.softmax(prob)
10 print("softmax(prob/T) : ", prob)
11 prob, _ = torch.topk(prob, top_k)
12 print("top-k:", prob)
13 prob = F.softmax(prob)
14 print("top-k softmax:", prob)
15 print("根据概率权重所选择next token的下标")
16 for i in range(10):
17     next_token = torch.multinomial(prob, num_samples=1)[0]
18     print(f"第{i}次sample的next_token下标为{next_token}:",
19           dict_map[int(next_token)])

```

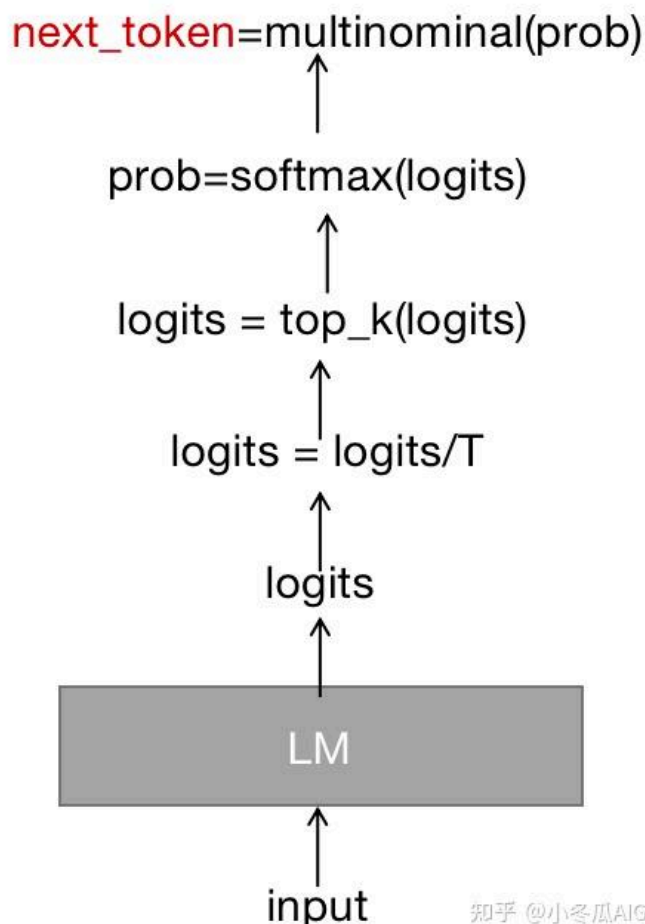
```

prob : tensor([0.7000, 0.2000, 0.1000])
prob/T : tensor([0.3500, 0.1000, 0.0500])
softmax(prob/T) : tensor([0.3969, 0.3091, 0.2940])
top-k: tensor([0.3969, 0.3091])
top-k softmax: tensor([0.5219, 0.4781])

```

知乎 @小冬瓜AIGC

4.2 Top-K完整生成流程+PyTorch代码



知乎 @小冬瓜AIGC


```
# GPT-2 Top-K generation
idx = X[1, :10].reshape(1, 10)
temperature = 2.0
top_k = 5
for _ in range(256):
    idx_cond = idx if idx.size(1) <= model.config.block_size else idx[:, -model.config.block_size:]
    logits, _ = model(idx_cond)
    logits = logits[:, -1, :] / temperature # 温度
    if top_k is not None:
        v, i = torch.topk(logits, min(top_k, logits.size(-1)))
        logits[logits < v[:, [-1]]] = -float('Inf')
    probs = F.softmax(logits, dim=-1)
    # sampling multinomial
    idx_next = torch.multinomial(probs, num_samples=1)
    idx = torch.cat((idx, idx_next), dim=1)
```

5. 重复性惩罚 Repetition Penalty

5.1 算法原理

重生成问题时LLM本身的弱点， 无论是否微调都可能出现。

对于重复性惩罚核心思想在于 "调整已出现过的tokens的logits"

generated 过程中重复性惩罚可追溯到以下文章。

Keskar N S, McCann B, Varshney L R, et al. Ctrl: A conditional transformer language model for controllable generation[J]. arXiv preprint arXiv:1909.05858, 2019.

To reconcile the two, we propose a new sampling scheme that trusts the model distribution through near-greedy sampling but prevents repetitions through a penalty. This penalized sampling works by discounting the scores of previously generated tokens.

$$p_i = \frac{\exp(x_i / (T \cdot I(i \in g)))}{\sum_j \exp(x_j / (T \cdot I(j \in g)))} \quad I(c) = \theta \text{ if } c \text{ is True else } 1$$

在Transformers库+中的 generate() 可设置惩罚系数

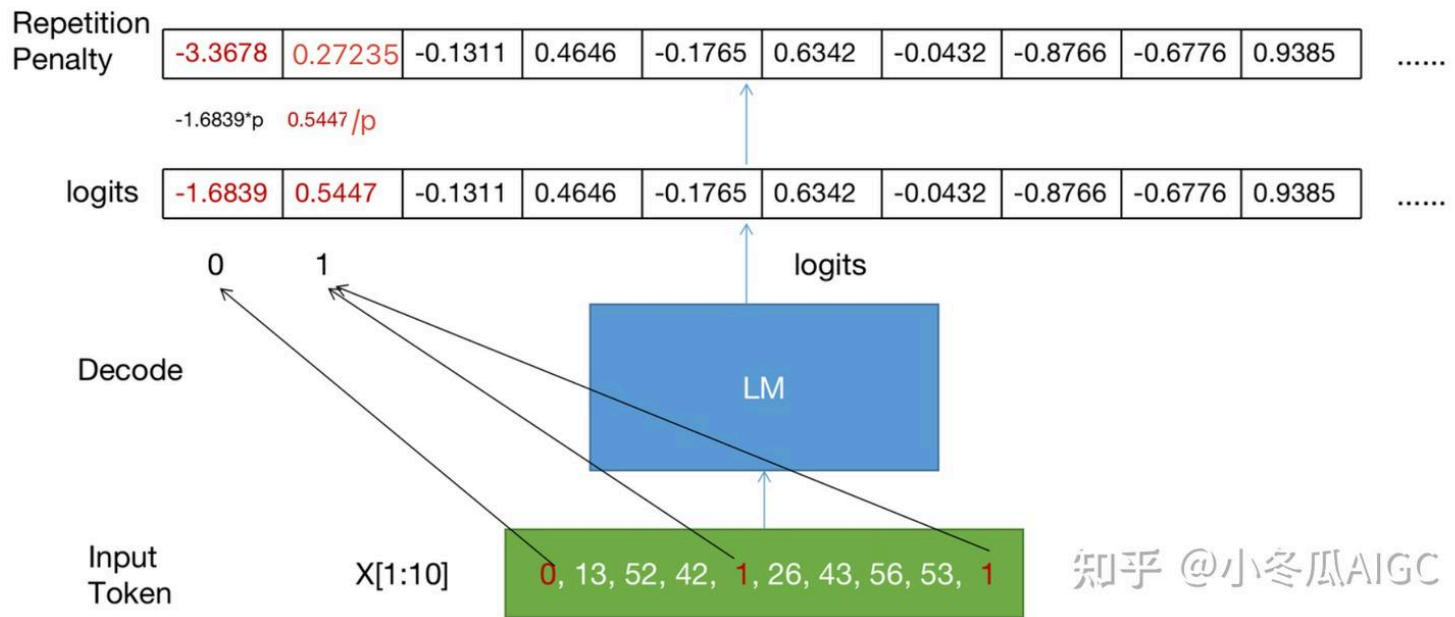
```
sample_outputs = model.generate(
    input_ids,
    top_k=50,
    repetition_penalty=1.2
)
```

5.2 重复性惩罚图解

重复性惩罚主要处理logits， 主要有以下处理步骤

1. input里出现的token挑选出来
2. 将input的token进行特定惩罚，目的是使对应token的logits数值尽可能小
3. 对于正负数需不同处理使得数值越小

$$RP(x) = \begin{cases} x/\theta, & \text{if } x > 0 \\ x \cdot \theta, & \text{if } x < 0 \end{cases}$$



知乎 @小冬瓜AIGC

5.3 Repetition Penalty 源码

以penalty为2.0举例，实际常用参数为1.2

```
# repetition penalty
idx = X[1, :10].reshape(1, 10)
print(idx)
penalty = 2.0
for _ in range(256):
    logits, _ = model(idx)
    logits = logits[:, -1, :]
    original_logits = logits.clone()

    #repetition penalty
    logits_idx = torch.gather(logits, 1, idx)
    logits_idx = torch.where(logits_idx < 0, logits_idx * penalty, logits_idx / penalty).clone()
    logits = logits.scatter_(1, idx, logits_idx)

    probs = F.softmax(logits, dim=-1)
    idx_next = torch.multinomial(probs, num_samples=1)
    idx = torch.cat((idx, idx_next), dim=1)
```

6. 其他

6.1 Transformers库generate示例

在Transformers库中可以轻松实现GPT中的Top-K采样生成。

```
sample_outputs = model.generate(
    input_ids,
    do_sample=True,
    max_length=50,
    top_k=50,
    top_p=0.95,
    temperature=0.7,
    repetition_penalty=1.2,
```

```
num_return_sequences=3
```

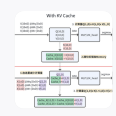
```
)
```

6.2 其他生成方法

- Top-P生成
- Beam-Search(TODO)

6.3 Generate 问题

- 从上述的生成过程中可见，每个Next-Token X_i 的预测都需要Input[0:i-1]的输入
- 此时会产生大量重复计算，导致 T^2 时间复杂度，效率太低
- 一种优化方法是增加KV-Cache⁺，而kv-cache是大模型的高效推理之基。推荐阅读：



小冬瓜AIGC：【手撕LLM-KVCache】显存刺客的前世今生--文末含代码

289 赞同 · 14 评论 文章

我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道。研究方向：LLM、RL、RLHF、AIGC和Agent。

《《手撕RLHF》解析如何系统的来做LLM对齐工程

小冬瓜AIGC：【手撕RLHF-Safe RLHF】带着脚镣跳舞的PPO

小冬瓜AIGC：【手撕RLHF-Rejection Sampling】如何优雅的从SFT过渡到PPO

《手撕LLM》系列文章+原创课程：LLM原理涵盖Pretrained/PEFT/RLHF/高性能计算

小冬瓜AIGC：【手撕LLM-QLoRA】NF4与双量化-源码解析

小冬瓜AIGC：【手撕LLM-RWKV】重塑RNN 效率完爆Transformer

小冬瓜AIGC：【手撕LLM-FlashAttention】从softmax说起，保姆级超长文！！

小冬瓜AIGC：【手撕LLM-Generation】Top-K+重复性惩罚

小冬瓜AIGC：【手撕LLM-KVCache】显存刺客的前世今生--文末含代码

小冬瓜AIGC：【手撕LLM-FlashAttention2】只因For循环优化的太美

《手撕Agent》从代码和工程角度，探索能够通向AGI的Agent方法

小冬瓜AIGC：【手撕Agent-ReAct】想清楚再行动、减轻LLM幻觉

我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道：手撕LLM+RLHF
研究方向：LLM、RLHF、Safety、Alignment

