大模型面经之解码策略topk、topp（附代码）

瓦力算法学研所 2024年05月13日 18:41 北京

介绍

大模型文本生成中的解码策略是用来控制生成文本的多样性和连贯性的方法。当模型为下一个词生成概率分布时，这些策略决定了从概率分布中选择词的方式。

常用的解码方式有：

- 贪心解码（Greedy Decoding）：直接选择概率最高的单词。这种方法简单高效，但是可能会导致生成的文本过于单调和重复。
- 随机采样（Random Sampling）：按照概率分布随机选择一个单词。这种方法可以增加生成的多样性，但是可能会导致生成的文本不连贯和无意义。
- Beam Search：维护一个大小为 k 的候选序列集合，每一步从每个候选序列的概率分布中选择概率最高的 k 个单词，然后保留总概率最高的 k 个候选序列。这种方法可以平衡生成的质量和多样性，但是可能会导致生成的文本过于保守和不自然

Top-k、Top-p和Temperature是三种LLM常用的解码策略

Top-k

Top-k采样是一种简单的截断策略，它从一个限制为前k个最可能的词的集合中随机挑选下一个词。"k"是一个可以调整的参数，较小的k值倾向于生成更可预测的文本，而较大的k值则鼓励多样性。当k很大时，几乎等同于无约束采样；当k=1时，该策略退化为贪心解码，即永远选择最可能的下一个词。

```python
import torch
from labml_nn.sampling import Sampler

# Top-k Sampler
class TopKSampler(Sampler):
    # k is the number of tokens to pick
    # sampler is the sampler to use for the top-k tokens
    # sampler can be any sampler that takes a logits tensor as input and retu
    def __init__(self, k: int, sampler: Sampler):
        self.k = k
        self.sampler = sampler

    # Sample from logits
    def __call__(self, logits: torch.Tensor):
        # New logits filled with -∞; i.e. zero probability
```

```
        zeros = logits.new_ones(logits.shape) * float('-inf')
        # Pick the largest k logits and their indices
        values, indices = torch.topk(logits, self.k, dim=-1)
        # Set the values of the top-k selected indices to actual logits.
        # Logits of other tokens remain -∞
        zeros.scatter_(-1, indices, values)
        # Sample from the top-k logits with the specified sampler.
        return self.sampler(zeros)
```

Top-p (Nucleus) Sampling

Top-p采样，又称为nucleus采样，是另一种更加动态的截断策略。它先对所有可能的下一个词的概率进行排序，**然后累加这些概率，直到达到预先设定的阈值p**。这样选择的词集合将包含累计概率至少为p的集合，这通常意味着包含多数概率大词但只包含部分词汇。这样，生成的文本既能够保持多样性，又能保持一定的连贯性。

```
import torch
from torch import nn

from labml_nn.sampling import Sampler


class NucleusSampler(Sampler):
    """
    ## Nucleus Sampler
    """
    def __init__(self, p: float, sampler: Sampler):
        """
        :param p: is the sum of probabilities of tokens to pick $p$
        :param sampler: is the sampler to use for the selected tokens
        """
        self.p = p
        self.sampler = sampler
        # Softmax to compute $P(x_i | x_{1:i-1})$ from the logits
        self.softmax = nn.Softmax(dim=-1)

    def __call__(self, logits: torch.Tensor):
        """
        Sample from logits with Nucleus Sampling
        """

        # Get probabilities $P(x_i | x_{1:i-1})$
```

```
        probs = self.softmax(logits)

        # Sort probabilities in descending order
        sorted_probs, indices = torch.sort(probs, dim=-1, descending=True)

        # Get the cumulative sum of probabilities in the sorted order
        cum_sum_probs = torch.cumsum(sorted_probs, dim=-1)

        # Find the cumulative sums less than $p$.
        nucleus = cum_sum_probs < self.p

        # Prepend ones so that we add one token after the minimum number
        # of tokens with cumulative probability less that $p$.
        nucleus = torch.cat([nucleus.new_ones(nucleus.shape[:-1] + (1,)), nuc

        # Get log probabilities and mask out the non-nucleus
        sorted_log_probs = torch.log(sorted_probs)
        sorted_log_probs[~nucleus] = float('-inf')

        # Sample from the sampler
        sampled_sorted_indexes = self.sampler(sorted_log_probs)

        # Get the actual indexes
        res = indices.gather(-1, sampled_sorted_indexes.unsqueeze(-1))

        #
        return res.squeeze(-1)
```

Temperature

Temperature用来平滑或加剧模型的概率分布。温度参数T影响每个词被选中的概率。通过改变概率分布的锐度，可以控制生成的文本的随机性：

$$\rho_i = \frac{1}{Q} e^{-\epsilon_i/kT} = \frac{e^{-\epsilon i/kT}}{\sum j = 1^M e^{-\epsilon_j/kT}}$$

当T>1时，分布变得更平缓，小概率事件被选中的机会增大，这增加了生成文本的随机性和多样性。

当T=1时，保留模型原始的概率分布。

当T<1时，分布变得更尖锐，大概率事件更有可能被选中，这使得文本生成更加确定性，并且倾向于重复性。

```python
import torch
from torch.distributions import Categorical

from labml_nn.sampling import Sampler


class TemperatureSampler(Sampler):
    """
    ## Sampler with Temperature
    """
    def __init__(self, temperature: float = 1.0):
        """
        :param temperature: is the temperature to sample with
        """
        self.temperature = temperature

    def __call__(self, logits: torch.Tensor):
        """
        Sample from logits
        """

        # Create a categorical distribution with temperature adjusted logits
        dist = Categorical(logits=logits / self.temperature)

        # Sample
        return dist.sample()
```

配合使用这些策略可以极大影响生成文本的品质。实践中，人们经常根据具体应用调整这些参数，以便获得既丰富多样又逻辑连贯的文本。实际上，这些策略也经常被一起使用。例如，可以同时使用top-k和top-p来过滤候选词汇（topk >topp, 也即先选取topk个最大概率的候选项，再从这些候选项中概率和满足p的前n个选项），再通过调整temperature来进行最终的词选取。

文章来源：

https://www.zhihu.com/tardis/bd/art/647813179?source_id=1001

本系列将会持续更新，想要获取面经资料的同学欢迎关注公众号，进群一起交流~