## LayerNorm 计算式

对于同样的输入 $X$，LayerNorm 的计算可以表示为：

1. 首先计算每个样本的特征值的均值和标准差：

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

2. 使用均值 $\mu$ 和标准差 $\sigma$ 对输入 $X$ 进行归一化：

$$X_{\text{norm}} = \frac{X - \mu}{\sigma + \epsilon}$$

3. 最后，将归一化的数据 $X_{\text{norm}}$ 乘以可学习的权重 $w$ 并加上偏置 $b$：

$$Y = w \cdot X_{\text{norm}} + b$$

```
import torch
import torch.nn as nn

class LayerNorm(nn.Module):
    def __init__(self, dim, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.eps = eps
        # 为每个输入特征学习一个独立的缩放参数和偏移参数
        self.weight = nn.Parameter(torch.ones(dim))
        self.bias = nn.Parameter(torch.zeros(dim))

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True, unbiased=False)
        return self.weight * (x - mean) / (std + self.eps) + self.bias
```

## RMSNorm 计算式

对于给定的输入 $X$（其中 $X$ 是一个 $n \times d$ 的矩阵，$n$ 是批次大小，$d$ 是特征维度），RMSNorm 的计算可以表示为：

1. 首先计算每个样本的特征平方的均值：

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i^2$$

其中 $x_i$ 是 $X$ 中的一个特征。

2. 接着计算均值的平方根的倒数，同时加上一个小的常数 $\epsilon$ 以避免除以零：

$$\text{RMS} = \frac{1}{\sqrt{\mu + \epsilon}}$$

3. 最后，使用得到的 $\text{RMS}$ 值对输入 $X$ 进行归一化，并乘以可学习的权重参数 $w$：

$$Y = X \cdot \text{RMS} \cdot w$$

```
class RMSNorm(torch.nn.Module):
    def __init__(self, dim: int, eps: float = 1e-6):
        super().__init__()
        self.eps = eps
        self.weight = nn.Parameter(torch.ones(dim))

    def _norm(self, x):
        return x * torch.rsqrt(x.pow(2).mean(-1, keepdim=True) + self.eps)

    def forward(self, x):
        output = self._norm(x.float()).type_as(x)
        return output * self.weight
```

相对于LayerNorm，RMSNorm的计算量更小，因此在处理大规模数据时更为高效。

# SwiGLU激活函数

$$\text{SwiGLU}(x) = (\text{Linear}(x) \cdot \text{Sigmoid}(\text{Linear}(x)))$$

结合了线性单元和门控机制，使其能够更复杂地表示非线性关系，能够有效地避免神经元死亡、缓解梯度消失和爆炸问题。相比之下，ReLU只是简单地将负值截断为零，表达能力相对有限。

# decoding过程中的采样策略

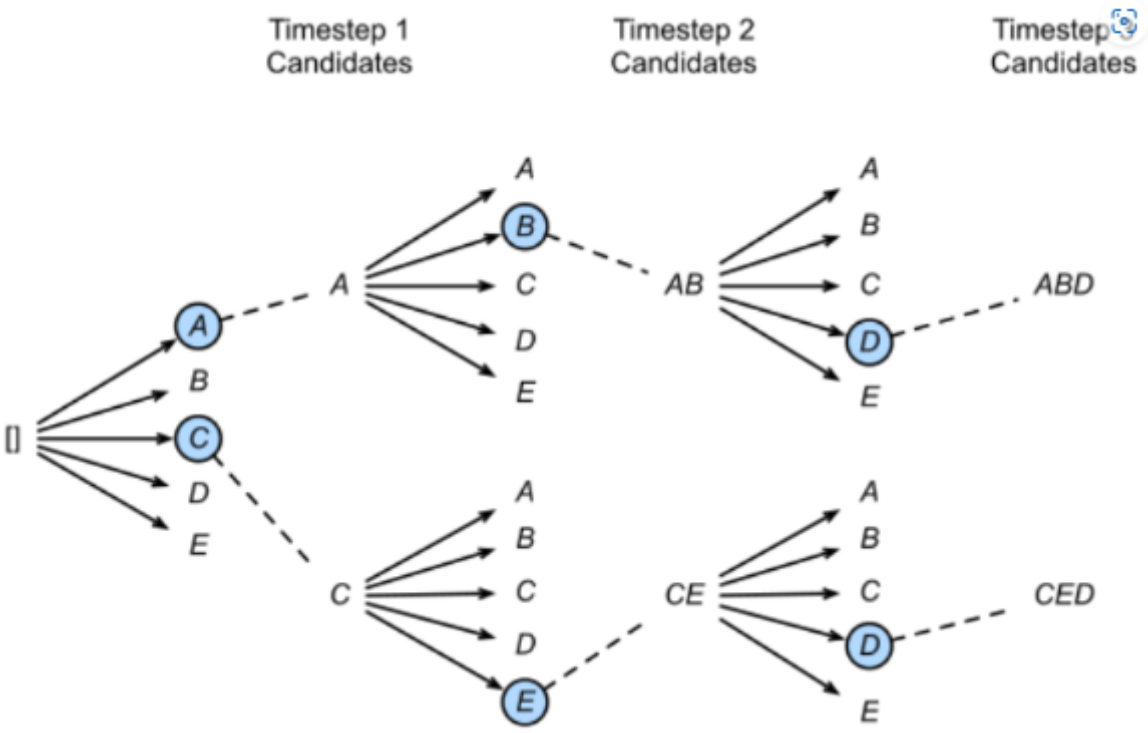
来自CSDN@自律也自由

## Greedy search

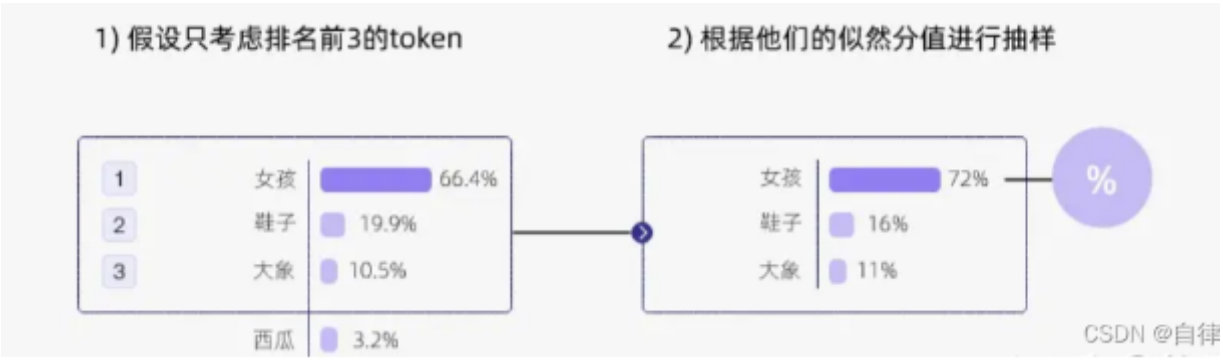
每次都选择当前时间步输出最大概率的词，贪婪搜索简单，计算量少，但是并不能找到全局最优的解。

## Beam Search

beam search是对贪心策略一个改进。思路也很简单，就是稍微放宽一些考察的范围。在每一个时间步，不再只保留当前分数最高的1个输出，而是保留num_beams个。当num_beams=1时集束搜索就退化成了贪心搜索。是一种启发式图搜索算法，通常用在图的解空间比较大的情况下，为了减少搜索所占用的空间和时间，在每一步深度扩展的时候，剪掉一些质量比较差的结点，保留下一些质量较高的结点。这样减少了空间消耗，并提高了时间效率，但缺点就是有可能存在潜在的最佳方案被丢弃，因此Beam Search算法是不完全的，一般用于解空间较大的系统中。




## Top-k抽样

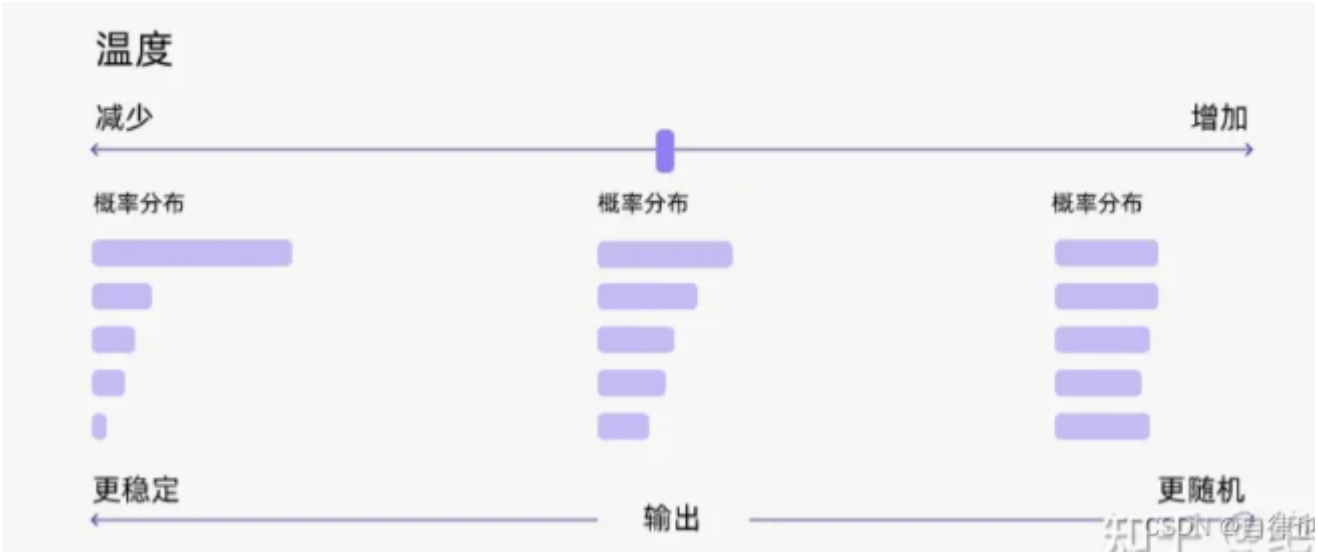



- 思路：从 tokens 里选择 k 个作为候选，然后根据它们的 likelihood scores 来采样模型从最可能的"k"个选项中随机选择一个，如果k=3，模型将从最可能的3个单词中选择一个。
- 优点：Top-k 采样是对前面“贪心策略”的优化，它从排名前 k 的 token 中进行抽样，允许其他分数或概率较高的 token 也有机会被选中。在很多情况下，这种抽样带来的随机性有助于提高生成质量。
- 缺点：在分布陡峭的时候仍会采样到概率小的单词，或者在分布平缓的时候只能采样到部分可用单词。k不太好选：k设置越大，生成的内容可能性越大；k设置越小，生成的内容越固定；设置为1时，和 greedy decoding 效果一样。

## Top-p抽样——核采样（Nucleus Sampling）


- 思路：候选词列表是动态的，从 tokens 里按百分比选择候选词，模型从累计概率大于或等于“p”的最小集合中随机选择一个，如果p=0.9，选择的单词集将是概率累计到0.9的那部分。
  - top-P采样方法往往与top-K采样方法结合使用，每次选取两者中最小的采样范围进行采样，可以减少预测分布过于平缓时采样到极小概率单词的几率。如果k和p都启用，则p在k之后起作用。
  - top-P越高，候选词越多，多样性越丰富。top-P越低，候选词越少，越稳定。
- 优点：top-k 有一个缺陷，那就是“k 值取多少是最优的？”非常难确定。于是出现了动态设置 token 候选列表大小策略——即核采样（Nucleus Sampling）。
- 缺点：采样概率p设置太低模型的输出太固定，设置太高，模型输出太过混乱。

## temperature




- 思路：通过温度，在采样前调整每个词的概率分布。温度越低，概率分布差距越大，越容易采样到概率大的字。温度越高，概率分布差距越小，增加了低概率字被采样到的机会。
- 参数：temperature(取值范围：0-1)设的越高，生成文本的自由创作空间越大，更具多样性。温度越低，生成的文本越偏保守，更稳定。
- 一般来说，prompt 越长，描述得越清楚，模型生成的输出质量就越好，置信度越高，这时可以适当调高 temperature 的值；反过来，如果 prompt 很短，很含糊，这时再设置一个比较高的 temperature 值，模型的输出就很不稳定了。

## 联合采样（top-k & top-p & Temperature）

思路：通常我们是将 top-k、top-p、Temperature 联合起来使用。使用的先后顺序是 top-k->top-p->Temperature。