Farthest Point Sampling (FPS)算法核心思想解析

不知道叫...
深度学习 入门级玩家

在点云研究中，采样算法举足轻重。目前很多流行的点云模型结构里面，都用到了FPS算法。本文将详细介绍FPS算法的流程以及代码实现里面的优化技巧，希望通过文本，读者对FPS的操作逻辑，时间、空间复杂度等，都能有一个清晰的理解，从而更好的理解相关文章。

1. 逻辑描述

假设有 n 个点，要从里面按照FPS算法，采样出 k 个点（ $k < n$ ）。逻辑上，可以将所有点归类到两个集合 A, B 里面。 A 表示选中的点形成的集合， B 表示未选中的点构成的集合。顾名思义，FPS做的事情是：每次从集合 B 里面选一个到集合 A 里面的点距离最大的点。

初始情况。 集合 A 为空，集合 B 包括所有点。

选第一个点。可以对所有点shuffle后，选第一个点即可。大部分实现也是这么做的。第一个点选完之后，将其移动到集合 A 中。此时，集合 A 包含 1 个点，集合 B 包含 $n-1$ 个点。

选第二个点。分别计算出集合 B 里面的每个点到集合 A 中的一个点的距离，选距离最大的点，将其移动到集合 A 中。此时，集合 A 包含 2 个点，集合 B 包含 $n-2$ 个点。

选第三个点。此时，如何定义集合 B 里面的点，到集合 A 里面的点的距离？因为集合 A 里面不止有一个点。这是理解FPS的核心。假设点 p_B 是集合 B 里面的一个点，计算 p_B 的距离的方式如下：

分别计算出 p_B 到集合 A 中每个点的距离。此时集合 A 里面有两个点，所以可以计算出两个距离值。
从计算出来的距离值里面，取最小的距离值，作为点 p_B 到集合 A 的距离值。

对于集合 B 里面的每个点，都可以计算出一个距离值： $\{p_B^1, p_B^2, \ldots, p_B^{n-2}\}$ 。选出最大的距离值对应的点，最为第3个点，移动集合 A 中。此时，集合 A 包含 3 个点，集合 B 包含 $n-3$ 个点。

之后可以按照选第三个点的方式，直到选出 k 个点为止。

2. 算法原理

假设此时集合 A 中包含 m 个点，集合 B 中包含 $n-m$ 个点。按照上面描述的逻辑，当选下一个点时，对于集合 B 里面的每个点，需要分别计算出到集合 A 中每个点的距离。大概估算一下，集合 B 共有 $n-m$ 个点，每个点要计算出 m 个距离值。所以，选下一个点的时间复杂度是 $(n-m)\times m$ 。

实际上，这里包含了一个重复计算的过程：

在选第 $m+1$ 个点时，对于集合 B 里面的一个点 p_B ，需要分别计算出到集合 A 里面每个点的距离。假设 d_B^1 表示点 p_B 到集合 A 中第一个点的距离，则需要计算的距离为：$\{d_B^1, d_B^2, \ldots, d_B^{m-1}, d_B^m\}$ ，然后取最小值 $min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}, d_B^m\})$ ，作为点 p_B 的距离值。
可以回退一步来看。在选第 m 个点时，对于集合 B 里面的点 p_B ，因为此时集合 A 里面有 $m-1$ 个点，所以需要计算的距离为：$\{d_B^1, d_B^2, \ldots, d_B^{m-1}\}$ 。

对比一下，可以看出，前后两步，点 p_B 重复计算的距离为：$\{d_B^1, d_B^2, \ldots, d_B^{m-1}\}$ 。

如何进行优化？

假设在选第 m 个点时，对于集合 B 里面的点 p_B ：

t_B^{m-1} 表示点 p_B 到集合 A 里面的 $m-1$ 个点的距离的最小值：

$$t_B^{m-1} = min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}\})$$

选出第 m 个点后，对于点 p_B ，继续选第 $m+1$ 个点时，按照逻辑，需要计算出：$min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}, d_B^m\})$ 。此时，并不需要再重复计算点 p_B 到集合 A 中前 $m-1$ 个点的距离，只需计算出点 p_B 到选出的第 m 个点的距离 d_B^m ，然后用 $min(min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}\}), d_B^m)$ ，即可计算出点 p_B 到集合A的距离。背后隐藏的数据公式是：

$$min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}, d_B^m\}) = min(min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}\}), d_B^m)$$

这个公式可以看作是一个简单的递推公式，如果 $t_B^{m-1} = min(\{d_B^1, d_B^2, \ldots, d_B^{m-1}\})$ ，则该公式可以写成：

$$t_B^m = min(t_B^{m-1}, d_B^m\})$$

如此以来，即可去掉重复计算的部分。代码实现上，只需要一个长度为 n 的数组，分别保存每个点到集合 A 的距离。每次新选出一个点后，按以上公式来更新这个数组即可。

此公式是大部分FPS代码实现里面的核心。

如果读者阅读过pointnet2里面fps的代码：

并且对里面的变量temp感到困惑，不妨试着按以上逻辑来理解。变量temp正是用来保存每个点到集合 A 的距离。

3. 算法分析

时间复杂度

每次选一个点，需要计算 n 个距离；选 k 个点，时间复杂度可以认为是： $\mathcal{O}(kn)$ ，由于 k 和 n 有一个常数比例关系，所以也可以认为是： $\mathcal{O}(n^2)$ 。

空间复杂度

需要一个长度为 n 的数组，来记录、更新每个点的距离值，所以复杂度为： $\mathcal{O}(n)$ 。