PointNet++

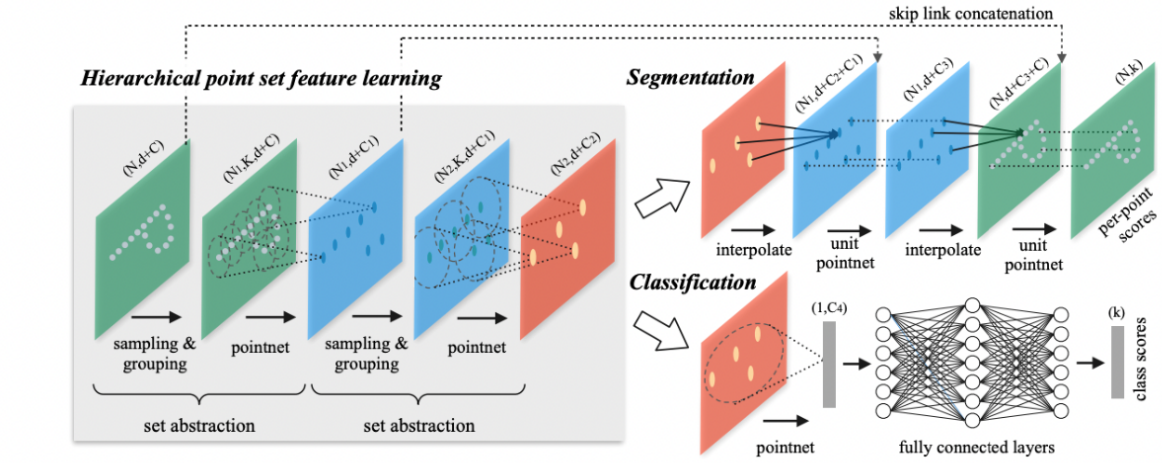
Created time	@September 21, 2022 2:28 PM
Last edited time	@November 27, 2022 12:12 AM
Date	@September 21, 2022 → September 22, 2022
Tags	
Status	Done

受到CNN多层级特征提取的的启发，将Hierarchical结构应用于PointNet

PointNet的几个问题：

1. point-wise MLP，仅仅是对每个点表征，对局部结构信息整合能力太弱 --> **PointNet++的改进：sampling和grouping整合局部邻域**
2. global feature直接由max pooling获得，无论是对分类还是对分割任务，都会造成巨大的信息损失 --> **PointNet++的改进：hierarchical feature learning framework，通过多个set abstraction逐级降采样，获得不同规模不同层次的local-global feature**
3. 分割任务的全局特征global feature是直接复制与local feature拼接，生成discriminative feature能力有限 --> **PointNet++的改进：分割任务设计了encoder-decoder结构，先降采样再上采样，使用skip connection将对应层的local-global feature拼接**

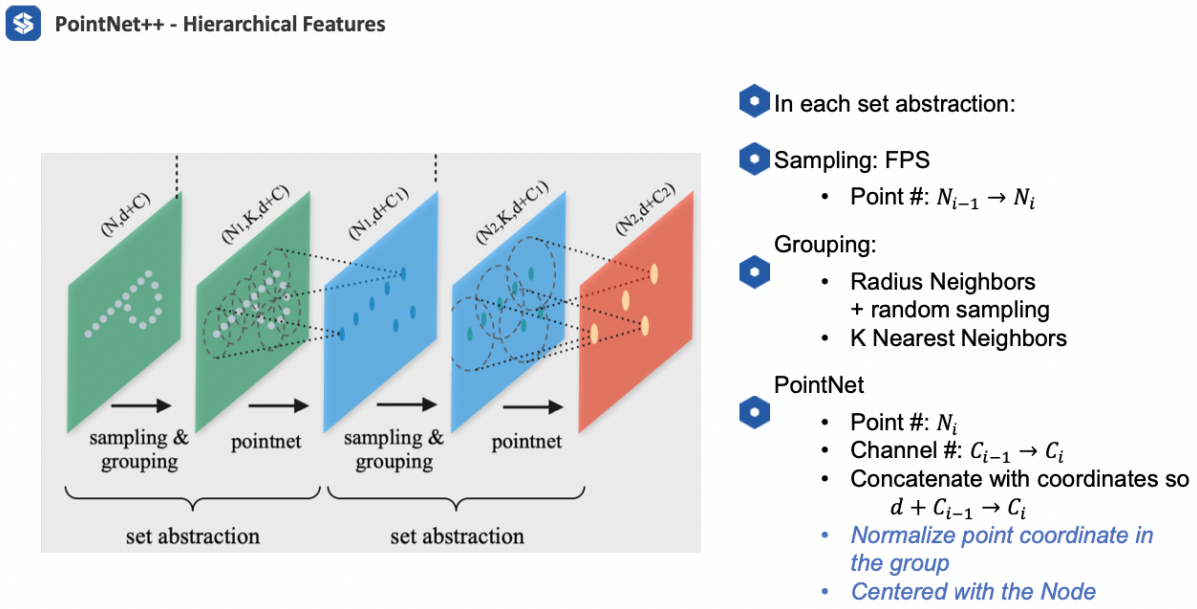
PointNet++ - Looks complicated?



PointNet++的网络大体是encoder-decoder结构

encoder为降采样过程，通过多个set abstraction结构实现多层次的降采样，得到不同规模的point-wise feature，最后一个set abstraction输出可以认为是global feature。其中set abstraction由sampling，grouping，pointnet三个模块构成。

decoder根据分类和分割应用，又有所不同。分类任务decoder比较简单，不介绍了。分割任务decoder为上采样过程，通过反向插值和skip connection实现在上采样的同时，还能够获得local+global的point-wise feature，使得最终的表征能够discriminative



Encoder部分

在PointNet的基础上增加了hierarchical feature learning framework的结构。这种多层次的结构由set abstraction层组成。

在每一个层次的set abstraction，点集都会被处理和抽象，而产生一个规模更小的点集，可以理解成是一个降采样表征过程，可参考上图左半部分。

Sampling

- 降采样 ： FPS 最远点采样

  输入规模为 **B**atch_size*__N__umber_of_points*(**d**im_xyz+**C**_extra_channels)

  一般 d = 3， c= 0

  输出规模为 ： **B**atch_size*__N_output__*(**d**im_xyz+**C**_extra_channels)

  N_output = **N1 < N** = Number_of_Points

```
def sample_and_group(npoint, radius, nsample, xyz, points, knn=False, use_xyz=True):
    '''
        points: (batch_size, ndataset, channel) tensor, if None will just use xyz as points
        knn: bool, if True use kNN instead of radius search
        use_xyz: bool, if True concat XYZ with local point features, otherwise just use point features
    Output:
        new_xyz: (batch_size, npoint, 3) TF tensor
        new_points: (batch_size, npoint, nsample, 3+channel) TF tensor
        idx: (batch_size, npoint, nsample) TF tensor, indices of local points as in ndataset points
        grouped_xyz: (batch_size, npoint, nsample, 3) TF tensor, normalized point XYZs
            (subtracted by seed point XYZ) in local regions
```

```
    '''
    new_xyz = gather_point(xyz, farthest_point_sample(npoint, xyz))
    # (batch_size, npoint, 3)

    if knn:
        _,idx = knn_point(nsample, xyz, new_xyz)
    else:
        idx, pts_cnt = query_ball_point(radius, nsample, xyz, new_xyz)
    grouped_xyz = group_point(xyz, idx) # (batch_size, npoint, nsample, 3)
    grouped_xyz -= tf.tile(tf.expand_dims(new_xyz, 2), [1,1,nsample,1]) # translation normalization
    if points is not None:
        grouped_points = group_point(points, idx) # (batch_size, npoint, nsample, channel)
        if use_xyz:
            new_points = tf.concat([grouped_xyz, grouped_points], axis=-1) # (batch_size, npoint, nample, 3+channel)
        else:
            new_points = grouped_points
    else:
        new_points = grouped_xyz
    return new_xyz, new_points, idx, grouped_xyz
```

farthest_point_sample输入输出非常明晰，输出的是降采样点在inp中的索引，因此是 B∗N1 int32类型的张量

```
def farthest_point_sample(npoint,inp):
    '''
input:
    int32
    batch_size * ndataset * 3   float32
returns:
    batch_size * npoint         int32
    '''
    return sampling_module.farthest_point_sample(inp, npoint)
```

gather_point的作用就是将上面输出的索引，转化成真正的点云

```
def gather_point(inp,idx):
    '''
input:
    batch_size * ndataset * 3   float32
    batch_size * npoints        int32
returns:
    batch_size * npoints * 3    float32
    '''
    return sampling_module.gather_point(inp,idx)
```

Grouping：

上一步sampling的过程是将 N∗(d+C) 降到 **N1∗(d+C)**。 实际上可以理解成是在 N 个点中选取 N1个中心点(key point)。

这一步的目的是以每个key point为中心，找其固定规模（令规模为 K ）的邻点，共同组成一个局部邻域(patch)。也就是会生成 N1 个局部邻域，输出规模为 **N1∗K∗(d+C)**

注意：

1)**找邻域的过程也是在坐标空间进行（也就是以上代码输入输出维度都是 d ，没有 C ， C 是在后面的代码拼接上的），而不是特征空间。**

2)找邻域这里有两种方式：KNN和query ball point.

```
if knn:
    _,idx = knn_point(nsample, xyz, new_xyz)
else:
    idx, pts_cnt = query_ball_point(radius, nsample, xyz, new_xyz)
grouped_xyz = group_point(xyz, idx) # (batch_size, npoint, nsample, 3)
```

其中前者KNN就是大家耳熟能详的K近邻，找K个坐标空间最近的点。后者query ball point就是划定某一半径，找在该半径球内的点作为邻点。

query ball point如何保证对于每个局部邻域，采样点的数量都是一样的呢？

事实上，如果query ball的点数量大于规模 K ，那么直接取前 K 个作为局部邻域；如果小于，那么直接对某个点重采样，凑够规模 K

摘自原文）Compared with kNN, ball query's local neighborhood guarantees a fixed region scale thus making local region feature more generalizable across space, which is preferred for tasks requiring local pattern recognition (e.g. semantic point labeling).也就是query ball更加适合于应用在局部/细节识别的应用上，比如局部分割。

sample和group操作都是在坐标空间进行的，因此如果还有特征空间信息（即point-wise feature），可以在这里将其与坐标空间拼接，组成新的point-wise feature，准备送入后面的unit point进行特征学习。

```
if points is not None: # points 含有特征信息的点云 C
    grouped_points = group_point(points, idx) # (batch_size, npoint, nsample, channel)
    if use_xyz:
        new_points = tf.concat([grouped_xyz, grouped_points], axis=-1) ## d+C (batch_size, npoint, nample, 3+channel)
    else:
        new_points = grouped_points ## C
else:
    new_points = grouped_xyz ## d
```

PointNet

输入 B*N1*K*(d+C) 输出 B*N1*(d+C1)

对B*N1个中心点，每个点代表的k-Patch做PointNet，实际使用二位卷积

1. point feature embedding B*N*K*(d+C1)

在每个Patch上作MaxPool

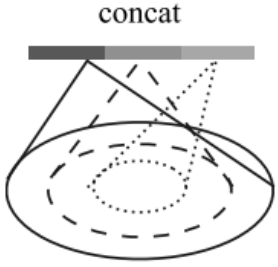
1. local pooling B*N1*1*C1

处理 Non-uniform Sampling Density

稀疏点云局部邻域训练可能不能很好挖掘点云的局部结构

解决方法：PointNet++做法：learn to combine features from regions of different scales when the input sampling density changes.

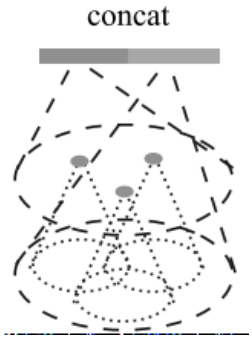
方案1: Multi-scale Grouping



对当前层的每个中心点，取不同radius的query ball，可以得到多个不同大小的同心球，也就是得到了多个相同中心但规模不同的局部邻域，分别对这些局部邻域表征，并将所有表征拼接。如上图所示。

代码层面其实就是加了个遍历radius_list的循环，分别处理，并最后concat

方案2：Multi-resolution grouping（MRG）



One vector (left in figure) is obtained by **summarizing the features at each subregion from the lower level** Li−1 using the set abstraction level.

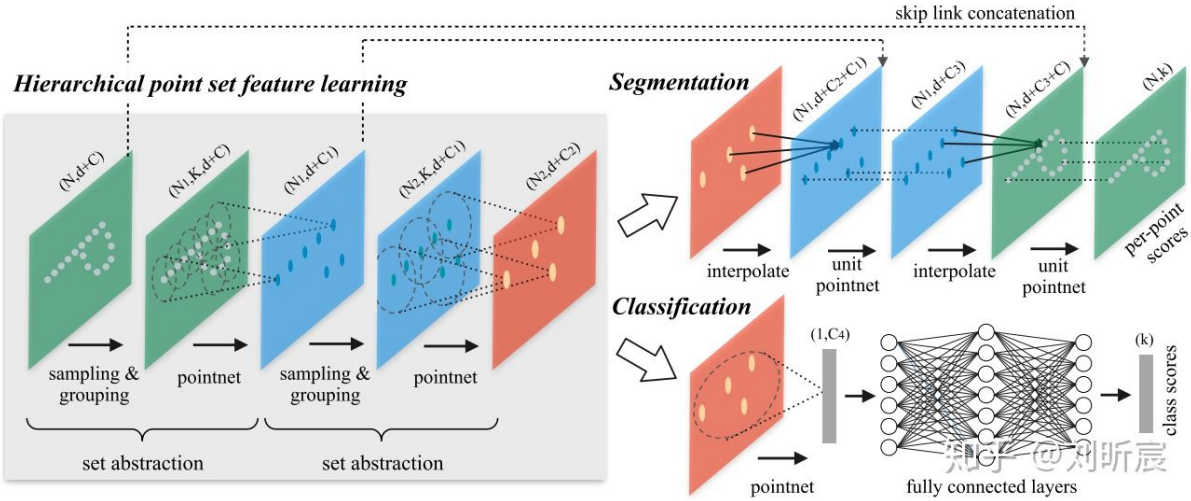
The other vector (right) is the feature that is obtained by **directly processing all raw points in the local region using a single PointNet**.

简单来说，就是当前set abstraction的局部邻域表征由两部分构成：

左边表征：对上一层set abstraction（还记得上一层的点规模是更大的吗？）各个局部邻域（或者说中心点）的特征进行聚合

右边表征：使用一个单一的PointNet直接在局部邻域处理原始点云

Decoder部分



分类 ：

B*N2*(d+C2) — PN—> **B*1*C4** —MLP+Softmax—> Scores

分割任务：

PointNet++设计了一种反向插值的方法来实现上采样的decoder结构，通过反向插值和skip connection来获得discriminative point-wise feature

设红色矩形点集 P1 : N1∗C ，蓝色矩形点集 P2 : N2∗C2 ，因为decoder是上采样过程，因此 N2>N1

1. 反向插值

对于 P2 中的每个点 x ，找在原始点云坐标空间下， P1 中与其最接近的 k 个点 x1,…,xk

当前我们想通过反向插值的方式用较少的点把更多的点的特征插出来，实现上采样

此时 x1,…,xk 的特征我们是知道的，我们想得到 x 的特征

如上公式，实际上就是将 x1,…,xk 的特征加权求和，得到x的特征。其中这个权重是与x和 x1,…,xk 的距离成反向相关的，意思就是距离越远的点，对x特征的贡献程度越小

2. Skip Connection

仍是不够的，回传得到的point-wise feature是从decoder的上一层得到的，因此算是global级别的信息，这对于想得到discriminative还是不够，因为我们还缺少local级别的信息！！！

上图中encoder蓝色矩形点集的 C1 表征是来自于规模更大的绿色矩形点集的表征，这在一定程度上其实是实现了local级别的信息

loss

无论是分类还是分割应用，本质上都是分类问题，因此loss就是分类任务中常用的交叉熵loss