GCN data:

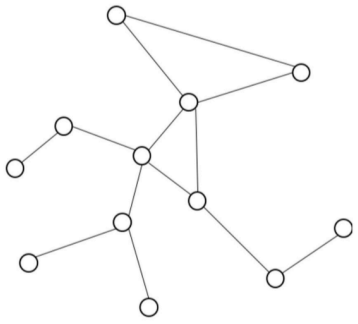
- Social Network
- Citation Network
- Molecules（分子结构）
- **Point Cloud**：
  - 严格来说，只有点没有连线，不是天生的图结构，但可以人工的表达成图的结构；
  - Edge的生成：不同人可以有不同的方法
    - Fix——NN查找或者radius NN；
    - Dynamic——DGCNN（见后）；
- 3D Mesh

GCN for Point Cloud

Graph 基础复习：见下图（图1）

- Graph - $G(V, E)$
  - V: a set of vertices
  - E: a set of edges
- Represent a point cloud by graph?
  - One point – one vertex
  - Edge
    - Fix – by coordinate based kNN/RadiusNN
    - Dynamic – **DGCNN**

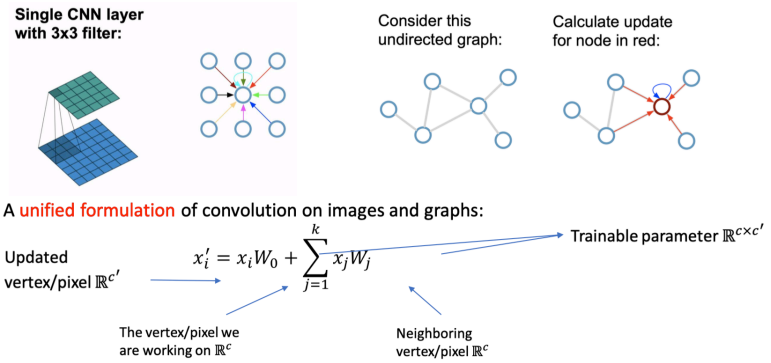
Vertex info:
Coordinate: $\mathbb{R}^3$
Feature: $\mathbb{R}^m$



Graph – $G(V, E)$

Convolution on Graph：

- 不像2D或者3D网格，不是定义在欧式空间，一个点周围的邻居不是固定的，导致卷积和pooling很麻烦！ 同时，也衍生出很多方法，所以GCN是一个family。
  - 图上定义卷积： 实际上kernel做卷积就是一个加权平均的矩阵化计算！ （图2）



- 问题：
  - 图上，每个点的邻居数量不是固定的，而以往kernel表示，实际的邻居数是固定的，比如3×3卷积，一个点有8个邻居；
  - 边的权重不是统一的，kernel表示下每个点的边权是一致的，而图结构下，每个邻居的边实际有不同的权重，上述方法没有考虑；

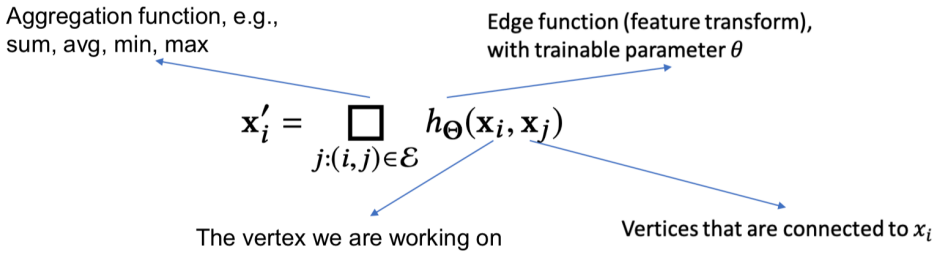
DGCNN

- DGCNN方案：
  - 问题1：因为实际点云是没有边的，是我们自己设置的，因此设置同样个数的连接即解决；
  - 问题2：实际上没有直接考虑这点，就假设边权是一样的，但由于输入包含坐标信息，而边权也可以用两点之间的相对距离表示，所以这个角度可以认为也考虑进去了；

- EdgeConv：主要贡献之一！
  - 是一个大一统的公式，把之前PointNet、PointNet++等方法都统一起来的，可以演化出不同的实现方法。

EdgeConv

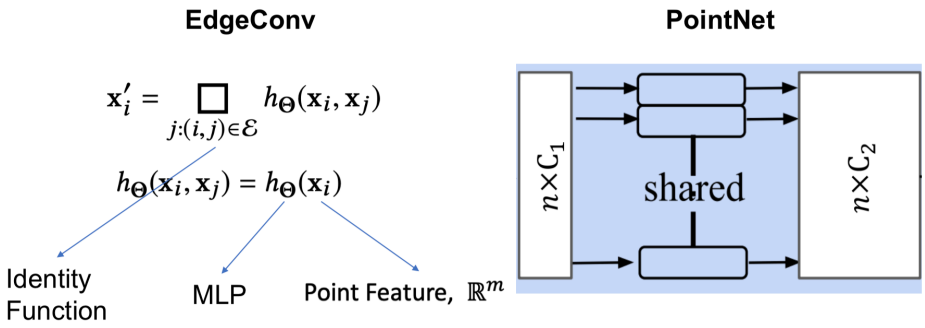
- Aggregation of neighboring vertex



  - 本质上，我们希望根据点x_i和其邻居x_j做feature transform（Edge function，这里通常是MLP，即 线性变化+ReLu），然后用aggregation function（例如sum、avg、min、max等）把点和其邻居组合起来变成一个输出vector；

  - x_i经过EdgeConv 从c维变换（encoding）c'维；

  - 例子：（图5）
    - PointNet：



- PointNet++:

EdgeConv

Maxpool

$$\mathbf{x}_i' = \square_{j:(i,j)\in\mathcal{E}} h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$$

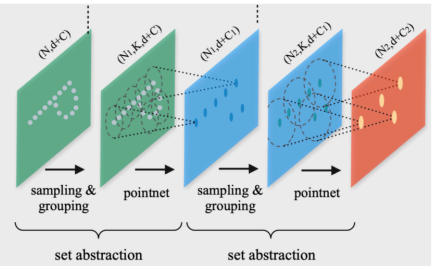
$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_j)$$

MLP — Features of dimension C, C_1, C_2

$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_j - \mathbf{x}_i)$$

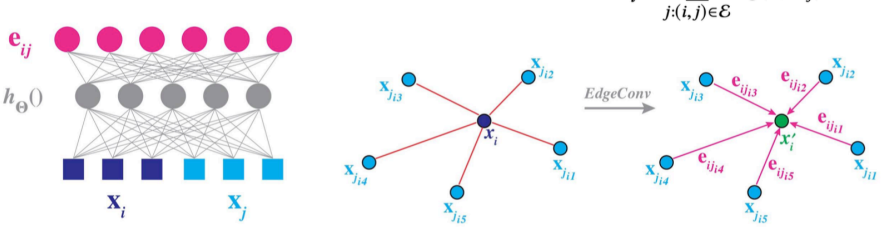
MLP — Coordinate of dimension d = 3

PointNet++



- DGCNN:

$$\mathbf{x}_i' = \square_{j:(i,j)\in\mathcal{E}} h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$$



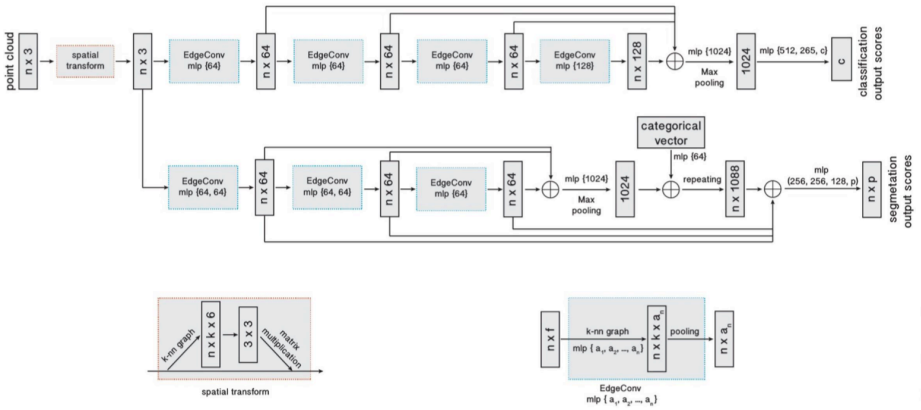
MLP with trainable param Θ — MLP with trainable param ϕ

$$h_\Theta(x_i, x_j) = h_\Theta(x_j - x_i) + h_\phi(x_i)$$

Neighboring vertex, $\mathbb{R}^m$ — The vertex we are working on, $\mathbb{R}^m$

$$x_i' = maxpool_j\left(h_\Theta(x_i, x_j)\right)$$

- DGCNN 网络结构：和PointNet极其相似，仅仅是把简单的MLP变成了通用的EdgeConv！



- Similar to T-Net of PointNet,
- NOT presented in the author's open-source code

n points, each is $x_i' = maxpool(h_\Theta(x_j - x_i) + h_\phi(x_i))$

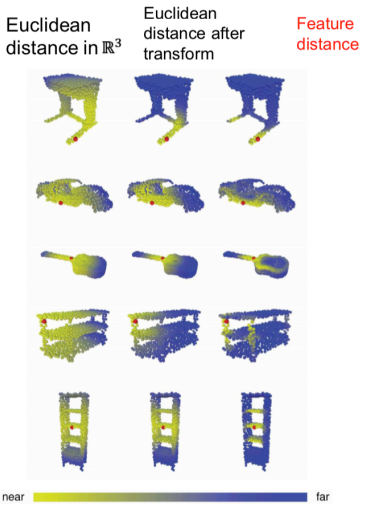
- DGCNN 图的构建：
  - KNN on coordinates：属于静态建图，DGCNN说不好！
  - KNN on features：动态建图 for subsequent EdgeConv；
    - 特别的，对第一层，携带的feature就是coordinates；
    - 神经网络拥有了** 调整边连接 **的能力，基于特征空间（学习出来的）；

  - 可视化：说明dynamic 基于语义特征的KNN建图方法有什么不一样（图6）
    - 神经网络如果要做分类、分割，更关心的是东西表达的语义而不是几何空间的距离；

Left
- Euclidean distance in $\mathbb{R}^3$

Middle
- Euclidean distance in $\mathbb{R}^3$
- After 3×3 matrix transform

Right
- Feature distance in $\mathbb{R}^3$
- From the last layer
- Semantic & dynamic



通用 GCN：
- 图结构表示：相似矩阵A（谱聚类那一章有学到）
  - 无连接-0
  - 有连接-1

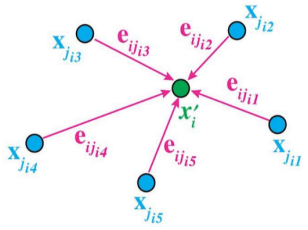
- 衍生概念：
  - Degree matrix D：n×n；
  - Laplacian matrix：L = D - A；
  - Normalized Laplacian matrix：Lsym = (D^-1/2)L(D^-1/2) = I - D^-1/2)A(D^-1/2)
  - 为什么需要这些衍生概念？（见后直观理解分析）

- 一种通用GCN layer的理解：主要包括下面两个步骤（图7）

Input: $X \in \mathbb{R}^{n \times C_{in}}$

Output: $H \in \mathbb{R}^{n \times C_{out}}$

A GCN layer consists of two steps
- Aggregation
  - Gather features from neighbors
- Update
  - Apply learnable layer to transform the aggregated features



- Aggregation：周围（邻居）信息（feature）的集合；
  - 例如：DGCNN里的(x_i-x_j)，只是一个减法收集信息，还没有函数处理；

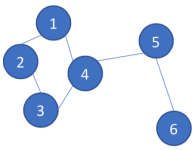
- Update：用函数处理aggregated features；
  - 例如：MLP处理；

- 直观理解衍生概念：
  - 简单的GCN方案如下图（图8）：加权求和收集信息 + 全连接层&ReLu 处理信息更新；

Input: $X \in \mathbb{R}^{n \times C_{in}}$

Output: $H \in \mathbb{R}^{n \times C_{out}}$

Graph



Connectivity / Similarity matrix A

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Aggregation
- Weighted sum of neighbors
- $N = A \cdot X$

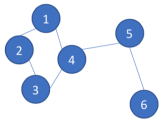
Update
- Linear (Fully Connected) Layer with learnable param $W \in \mathbb{R}^{C_{in} \times C_{out}}$
- Activation function σ, e.g, ReLU
- $H = \sigma(N \cdot W) = \sigma(AXW)$

  - 问题1：每个点没有考虑自己的信息（A的对角线元素是0）；
    - Laplacian matrix L解决；（图9）

Input: $X \in \mathbb{R}^{n \times C_{in}}$

Output: $H \in \mathbb{R}^{n \times C_{out}}$

Graph



Laplacian Matrix L

$$\begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ -1 & 0 & -1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Aggregation
- Weighted sum of neighbors and itself
- $N = (D - A) \cdot X = LX$
- In vector form: $N_i = \sum_j (A_{ij}(X_i - X_j))$

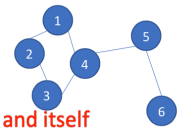
Update
- Linear (Fully Connected) Layer with learnable param $W \in \mathbb{R}^{C_{in} \times C_{out}}$
- Activation function σ, e.g, ReLU
- $H = \sigma(N \cdot W) = \sigma(LXW)$

  - 问题2：简单的加权求和，如果一个点连接的点特别多，求和/平均的结果会特别大，主导了神经网络，即A没有经过归一化；
    - 归一化Laplacian matrix Lsym解决；（图10）

Input: $X \in \mathbb{R}^{n \times C_{in}}$

Output: $H \in \mathbb{R}^{n \times C_{out}}$

Graph



Normalized Laplacian Matrix L_{sym}

$$\begin{bmatrix} 1 & -\frac{1}{\sqrt{2}\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}\sqrt{3}} & 0 & 0 \\ -\frac{1}{\sqrt{2}\sqrt{2}} & 1 & -\frac{1}{\sqrt{2}\sqrt{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}\sqrt{2}} & 1 & -\frac{1}{\sqrt{2}\sqrt{3}} & 0 & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & 0 & -\frac{1}{\sqrt{2}\sqrt{3}} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -\frac{1}{\sqrt{2}\sqrt{1}} \\ 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}\sqrt{1}} & 1 \end{bmatrix}$$

Aggregation
- Normalized weighted sum of neighbors and itself
- $N = D^{-1/2} L D^{-1/2} \cdot X = L_{sym} X$
- In vector form: $N_i = \sum_j \left(A_{ij}(X_i - X_j) \cdot \frac{1}{\sqrt{D_{ii} D_{jj}}} \right)$

Update
- Linear (Fully Connected) Layer with learnable param $W \in \mathbb{R}^{C_{in} \times C_{out}}$
- Activation function σ, e.g, ReLU
- $H = \sigma(N \cdot W) = \sigma(LXW)$

- 对比DGCNN：（图11）

DGCNN

- For each i
- MLP on neighbors → get multiple feature vectors
- Maxpool → get final feature vector for i

MLP with param Θ　　MLP with param ϕ

$$h_\Theta(x_i, x_j) = h_\Theta(x_j - x_i) + h_\phi(x_i) \qquad x_i' = maxpool_j\left(h_\Theta(x_i, x_j)\right)$$

Neighboring vertex, $\mathbb{R}^m$　　The vertex we are working on, $\mathbb{R}^m$

General GCN

- For each i
- Aggregation: Weighted sum of neighbors → get one feature vector
- Update: MLP on the feature vector → get final feature vector for i

- 不同点：区别很细微，DGCNN是GCN的特例；（随着网络的堆叠，先MLP还是后做，实际上没有区别）
  - Aggregation：
    - DGCNN先MLP（N×K×C_in --> N×K×C_out），GCN只是加权求和，没有MLP；
    - DGCNN固定了K邻居，K是个常数，GCN是根据拉普拉斯矩阵，不同数量的邻居连接；
  - update：GCN使用MLP；
  - DGCNN没有经过normalization，可以尝试改进作为一个点！