Dynamic Graph CNN for Learning on Point Clouds

Marius Dufraisse

This paper proposes a new *EdgeConv* layer to process point clouds. This layer was developed to be similar to the 2D convolution used with great success in computer vision : it takes into account local neighborhood information and can be used to design neural networks for classification or semantic segmentation of point clouds.

1 Background

Hand-Crafted Features Point Clouds processing often requires a low-dimensional representation of the local geometry. Many descriptors have been proposed including those detailed during the course. Although such descriptors can be used to represent the geometry of neighborhoods its not straight forward to aggregate them to get global information.

Learning features The computer vision community has extensively shown that convolution can be used to learn hierarchy of features on images. Unfortunately this operation is not appropriate to process point clouds as, in this setting, data is not aligned on a grid. One way to circumvent this issue is to divide the space into a grid and apply the convolution on the resulting voxels. The main drawback of this method is that it requires large amount of memory as it has to store information even were there are no points in the cloud.

paragraphe sur pointnet ?

2 EdgeConv

The EdgeConv layer forward pass is done in 2 steps. The first one extract the local neighborhood and uses this to build a small graph around the processed point ; the second apply a pseudo convolution on the edges that link the points in this graph.

We will describe those in the general case were we consider a point cloud $X = \{x_1, \dots, x_n\} \subset \mathbf{R}^F$.

Building the graph EdgeConv uses the k-nearest neighbor (k-NN) graph $\mathcal{G} = (X, \mathcal{E})$ with $(x_i, x_j) \in \mathcal{E}$ if and only if x_i is one of the k nearest neighbors of x_j. Each edge is described with a feature $e_{ij} = h_\Theta(x_i, x_j)$ where h_Θ depends on some learnable parameters Θ. It is import to see that when stacking multiple EdgeConv layers this graph will be computed at each layer since points can be close in the 3D space but might be different in the feature space. This has the added benefit of creating non local features.

Aggregating the edges Once the k-NN graph is computed, the output of the layer is obtained as $x'_i = \bigoplus_{j:(i,j)\in\mathcal{E}} h_\Theta(x_i, x_j)$. Since point cloud are defined up to permutation of the points we want the EdgeConv layer to be invariant to this permutation. This means that $\bigoplus$ must be symmetric, in the paper they use the maximum , but the sum would work as well.

EdgeConv generalizes PointNet [1] as it can be obtained by taking $h_\Theta(x_i, x_j) = h_\Theta(x_i)$. The author of suggested to use a function of the form $h_\Theta(x_i, x_j - x_i)$ to keep both information about the position of the point in the feature space (via x_i) as well as more precise information about the local neighborhood (using $x_j - x_i$).

Properties Knowing the group of symmetries for which a neural network is interesting is important. We previously said that the layer is permutation invariant which is extremely important to process point clouds. It is also partially translation invariant. More precisely, if the input is translated by a vector T then the edge value will be $h_\Theta(x_i + T, (x_i + T) - (x_j + T)) = h_\Theta(x_i + T, x_i - x_j)$. This is not an issue as it means that global position information (first argument of h_Θ) isn't lost but the processing of the local neighborhood is invariant.

J'ai pas parlé du lien avec les graphCnn

1. This not actually true as PointNet reduces the cloud size and does not update the graph at each layer.