Learning task-specific features for 3D point cloud graph creation (May change)

Elías Abad-Rocamora
elias.abad@estudiantat.upc.edu

Javier Ruiz-Hidalgo
j.ruiz@upc.edu

Abstract

The ABSTRACT is to be in fully-justified italicized text, at the top of the left-hand column, below the author and affiliation information. Use the word "Abstract" as the title, in 12-point Times, boldface type, centered relative to the column, initially capitalized. The abstract is to be in 10-point, single-spaced type. Leave two blank lines after the Abstract, then begin the main text. Look at previous CVPR abstracts to get a feel for style and length.

1. Introduction

TODO

When learning 3D pointclouds representations, Graph Neural Networks are a common choice (explain why: Unordered operator, local operator, solves irregularity, unorderedness and unstructuredness.), but relationship between points is not explicit and therefore, edges have to be assigned with some rule. This is done usually with heuristics like KNN or Thresholded distance. We propose a generalization of KNN over a task-specific learnt point representation. We prove that this method is a generalization of KNN in the sense that KNN is a particular case of our method and that at least, same results should be achieved with it. We also empirically show that our method works by benchmarking it on a popular segmentation dataset (ModelNet40) against regular KNN over XYZ features.

Reference Challenges of 3D point cloud processing with deep learning [1]. (Irregularity, unorderedness and unstructuredness).

Mention different methods related to Deep Learning on 3D Point Clouds: Multiview based, Voxel based and applied directly to point clouds (PointNet and then Graph based).

PointNet [9] was the first architecture able to directly process unstructured 3D point clouds. This was done by applying 2 symmetric functions: a MLP with learnable parameters and a maxpooling function.

2. Related work

There are numerous works that rely on Graph Neural Networks to process unstructured 3D point clouds in

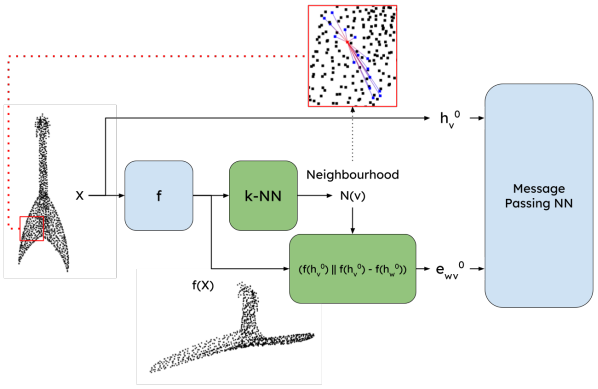

Figure 1. Graph creation pipeline: Each point is mapped from initial features to another space via f, where k-NN is performed to create the neighborhood. A combination of both nodes' transformed features is used to produce edge features, which along original node features are passed to a Message Passing Neural Network.

different manners [6], [12], [3], [11], [8]. All these architectures construct graphs by defining a node for each point in the point cloud and use some rule to define edges between them. [6] uses kd-tree graph and builds a feed-forward kd-network, where the leaves are the input of the network and the root is the output where the graph representation is encoded. [12] creates the edges by joining every node with the k-nearest neighbors in the point cloud, then uses edge convolution to update the node features and uses those learnt features to built the graph again, based on k-NN in the new feature space. At every layer [11] creates the edges by joining every node with k randomly chosen points that are at distance d $\leq \rho_l$, where ρ_l increases at every layer l, node features are updated by using an attention mechanism over adjacent nodes and furthest point sampling is used to reduce the dimensionality. [8] applies graph convolution at d-dilated k-nearest neighbors, in the first layer, xyz features are used and at consequent layers, learnt node features are used.

TODO: Mention newer work!!

3. Method

Motivated by how [12] and [8] use features at every layer to dynamically change the neighborhood, but all methods still use xyz features for the first layer graph, we designed our method to learn the best possible features to create the graph in the first layer.

3.1. Message Passing Neural Network

Our network relies on the Message Passing Neural Network (MPNN) framework [2]. This framework generalizes many Graph Neural Network methods with two phases: message passing phase, which runs for T timesteps and updates the node features, and a readout phase, which computes a feature vector for the whole graph.

3.1.1 Message passing phase

We base our work on the MPNN case defined in [4]. In this work, edge features are introduced into the MPNN. At each message passing stage, the node features of a node v are updated with an Multi Layer Perceptron (MLP) that takes as an input the concatenation $(. \parallel .)$ of the previous hidden state h_v^{t-1} and the message m_v^t:

$$h_v^t = H^t(h_v^{t-1} \parallel m_v^t) \tag{1}$$

Where H^t is an MPL and the message m_v^t is the sum of the features of its incoming edges:

$$m_v^t = \sum_{w \in N(v)} e_{wv}^t \tag{2}$$

The edge features e_{wv}^t are updated with an MLP E^t that takes as an input the concatenation of e_{wv}^{t-1}, h_w^{t-1} and h_v^{t-1}:

$$e_{wv}^t = E^t(e_{wv}^{t-1} \parallel h_w^{t-1} \parallel h_v^{t-1}) \tag{3}$$

Initial node hidden states h_v^0 are the pointcloud spatial features (xyz) and possibly other features like RGB color. We follow [12] and define starting edge features e_{wv}^0 as:

$$e_{wv}^0 = (f(h_v^0) \parallel f(h_w^0) - f(h_v^0)) \tag{4}$$

f can be any mapping.

3.1.2 Readout phase

Lastly, in the readout phase, we aggregate all the node features and obtain a global feature vector g of the graph. In order to do so, we need an order-invariant function like sum or max, in this work we apply max pooling after a skip connection and fully connected layer to aggregate features from each Message Passing stage:

$$g = \max_{v \in V} G(h_v^1 \parallel h_v^2 \parallel ... \parallel h_v^T) \tag{5}$$

Where G is an MLP. Finally, a set of fully connected layers can be applied to map g to the desired number of classes. If our task is to provide an output for each one of the nodes in the graph, the readout phase can be avoided and apply a shared weight MLP to each node features.

3.2. Latent features for k-NN edge creation

As mentioned in related work 2, many proposed architectures create directed graph edges by joining every node with its k-NN on input space. We propose learning a mapping $f : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{graph}}$ to another space, where we will perform k-NN on every node to create the edges of our graph. Our f mapping is defined by an MLP with learnable parameters F. Finally, the set of neighbors $N(v)$ of a node v is assigned as the k-NN in euclidean distance from the features $f(h_v^0)$ of the node v:

$$N(v) = \text{k-NN}_{f(h_w^0), w \in V}(f(h_v^0)) \tag{6}$$

3.3. Stress Loss

Stress is a well known metric in the statistics community. It was defined in [7] and it is the criterion to optimize when performing Multidimensional Scaling (MDS). We use Stress as a regularization of our mapping function, by jointly minimizing Stress, we restrict our MLP from mapping the input points to a complex space where the distances between points are very different. In this sense, Stress captures how well the transformed points represent the actual points. Let $d_{ij} = \|x_j - x_i\|_2$ and $\hat{d}_{ij} = \|f(x_j) - f(x_i)\|_2$ for $x_i, x_j \in \mathcal{X} = [x_1, x_2, ..., x_n] \in \mathbb{R}^3$ being the input pointcloud in xyz and $f : \mathbb{R}^3 \rightarrow \mathbb{R}^d$ the mapping represented by our MLP, Stress is defined as follows:

$$S(d, \hat{d}) = \sqrt{\frac{\sum_{i<j}(d_{ij} - \hat{d}_{ij})^2}{\sum_{i<j} d_{ij}^2}} \tag{7}$$

Then, to avoid computing the outer squared root, we can optimize squared stress, and do it jointly with the criterion of our task:

$$\min_\theta L = L_{task}(\theta) + \gamma S(d, \hat{d}(\theta))^2 \tag{8}$$

Where θ are the parameters of our model, L_{task} the appropriate loss function for our graph learning task (Classification, regression, etc.) and γ is a scalar fixed weight to control the importance of stress in the optimization of L.

4. Experiments

We test our work on two widely used 3D point cloud classification benchmarks: ModelNet40 [13] and ScanObjectNN [10]. For all of our experiments, we use an architecture with $T = 4$ Message passing stages. As defined in section 3.1.1, each Message Passing (MP) block is composed

of two MLPs: H^t and E^t, see eq. 1 and 3 respectively. The input size of E^t, depends on the number of output features of F^t, we denote this number as d_{graph}, see section 3.2. Both H^t and E^t depend on the number of input node features d_{in}. Finally, we use an MLP P with $d_{classes}$ output dimensions (40 for ModelNet40 and 15 for ScanObjectNN) after the readout phase to predict the pointcloud class, see table 4 for the dimensions of every MLP in the model. For the graph creation, we use $k = 16$ for selecting the k-NN. The whole architecture is trained through back-propagation for 100 epochs, with batch size equal to 16, a learning rate of 10^{-4} which is decayed in half every 20 epochs, ADAM optimizer [5] and Cross-entropy Loss for L_{task}, see equation 8.

In order to test the proposed architecture, we compare with a baseline model using $f(x_i) = xyz$, which is equivalent to using the Identity function for Modelnet40 and ScanObjectNN, as no other node features are provided in these datasets. Therefore, $d_{graph} = d_{in} = 3$. This baseline also allows us to be compared with other methods that create the neighborhood via k-NN over the spatial features.

4.1. Effect of d_{graph}

We investigate which is the best number of features to represent each point before performing k-NN and entering the first MP stage. In order to do so, we build F as a 2-layer MLP with a ReLU activation between them. The output dimensions of the first layer is fixed to 16 and 6 different values of d_{graph} (1,2,3,6,9 and 12) are tested in terms of average class accuracy and overall accuracy over the ModelNet40 dataset. For this experiment γ was set to 0 for every model.

Model	d_{graph}	Stress	mean Acc.	overall Acc.
baseline	3	0	**87.9**	91.6
mlp-3-16-1	1	0.496	86.6	90.1
mlp-3-16-2	2	0.482	86.4	90.3
mlp-3-16-3	3	0.674	87.7	90.9
mlp-3-16-6	6	1.12	**87.9**	**91.9**
mlp-3-16-9	9	1.89	87.0	90.8
mlp-3-16-12	12	2.61	87.4	91.4

Table 1. Classification results on ModelNet40 for different values of d_{graph}

NOTE: results of first run, still running 2 more runs of each experiment.

For output dimensions smaller than the input, the performance is clearly affected, providing around 1.5 points less in both average per-class accuracy and overall accuracy. This result is natural as F needs to find the configuration that provides best accuracy, and at the same time reduce the dimensionality of the data. In this case, the model cannot be able to learn the identity function as it is limited in the number of dimensions, the best it could do to be close to the baseline is to find a non-linear 1D/2D projection where distances in 3D are approximated.

For $d_{graph} \geq 3$, F is able to learn an identity mapping and therefore should be able to attain at least the same performance as our baseline. This is however not observed in our experiments, only for $d_{graph} = 6$ we are able to obtain same performance as the baseline in mean accuracy and outperform it just by just 0.3 points in overall accuracy 1. This makes us think that these models are not converging to the best solution possible.

4.1.1 Graph properties

In this section we analyze the properties of the graphs created with our model in comparison with the baseline. We intend to provide an explaination of which features is the network learning.

NOTE: Currently calculating Average Shortest path length and average number of weakly connected components. Need to analyze and see if it is interesting to include an analysis of those. Seems that the average shortest path length is greater than in the baseline.

4.2. Effect of γ

With the γ parameter, we can explicitly control how much importance has Stress in the optimization of F parameters, we argue that if k-NN over spatial features is the one of the most widely used graph creation method and has proven to provide excellent results, solutions that resemble well distances in the input space, should also provide good performance. Additionally, the fact that on the mlp-3-16-3 setup, F did not converge to the identity, but neither provided better or similar results to the baseline, made us think that F was not converging to the best possible solution. In order to analyze the effect of γ, we trained mlp-3-16-3 with γ values ranging from 0.0001 to 1 in x10 increments.

Model	γ	Stress	% shared edges	mean Acc.	overall Acc.
baseline				87.9	91.6
mlp-3-16-3	1	0.0071	97.2	**88.3**	**91.8**
	0.1	0.0233	93.3	88.0	91.4
	0.01	0.179	67.6	87.8	91.0
	0.001	0.331	57.8	86.5	90.9
	0.0001	0.578	58.9	87.5	91.4
	0	0.674	54.2	87.7	90.9

Table 2. Stress, percentage of edges shared with baseline's graph and Classification results on ModelNet40 for different values of γ

NOTE: results of first run, still running 2 more runs of each experiment.

The higher the values of γ, the closer F is to the identity mapping, this is observed in terms of Stress, which our experiments show to decrease with bigger γs and in terms of percentage of edges shared with baseline graph, which increases with γ until reaching a value of 97.2 at $\gamma = 1$, meaning an almost perfect match with baseline graph, see table 3 for details. This sense of proximity to the identity mapping is confirmed when looking at the plots of some of the transformed pointclouds. If one takes a look at figure 2, one can observe that in second and third columns, corresponding to $\gamma = 1$ and $\gamma = 0.1$ respectively, the pointclouds are very similar to the original one, while at the rightmost columns, one is not able to distinguish visually some of the shapes. The only thing that separates our mapping F from the identity is rotation. By minimizing Stress, we are only enforcing that the point-to-point distances are preserved. This means that for high γ values, F is just applying a rotation to the input pointcloud.

4.3. Comparison with other methods

Model	mean Acc.	overall Acc.
baseline	87.9	91.6
mlp-3-16-3	87.7	90.9

Table 3. Classification results on ModelNet40 for the 1024 point setup.

NOTE: By now we'll just compare ModelNet40, if there's time, we'll add ScanObjectNN.

5. Discussion

Comments on why it may not work (Or on why it does if we get it to do so in Christmas). New research lines on 3D point clouds task-specific graph learning.

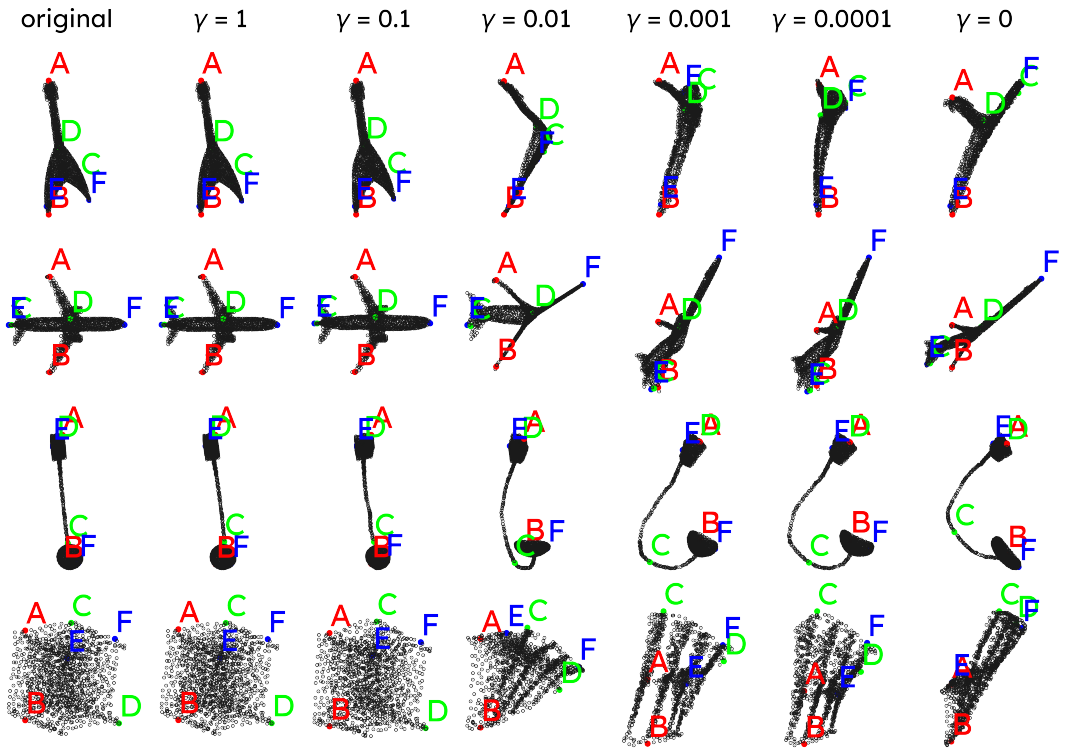

Figure 2. 2D projection of ModelNet40 test pointclouds number 29, 1, 230 and 512 after being transformed with F in mlp-3-16-3 setup trained with different γ values. "A" and "B","C" and "D" and "E" and "F" points correspond to argmax and argmin of x, y and z axis respectively in the original pointcloud. Every 3D pointcloud is projected into the hyperplane defined by "A", "B" and "F" in each space.

A. Architecture

References

[1] Saifullahi Aminu Bello, Shangshu Yu, and Cheng Wang. Review: deep learning on 3d point clouds. *CoRR*, abs/2001.06280, 2020.

[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[3] Wenkai Han, Chenglu Wen, Cheng Wang, Xin Li, and Qing Li. Point2node: Correlation learning of dynamic-node for point cloud feature modeling. *CoRR*, abs/1912.10775, 2019.

[4] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, Aug 2016.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[6] Roman Klokov and Victor S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *CoRR*, abs/1704.01222, 2017.

[7] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, Mar 1964.

[8] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Can gcns go as deep as cnns? *CoRR*, abs/1904.03751, 2019.

[9] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[10] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. *CoRR*, abs/1908.04616, 2019.

[11] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10288–10297, 2019.

[12] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

[13] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *CoRR*, abs/1406.5670, 2014.

MLP	Input size	Layer	Out	Act.	Norm.
F	d_{in}	Hidd Out*	16 d_{graph}	ReLU	
E^1	2*(d_{graph}+d_{in})	Hidd Out	32 64	ReLU ReLU	✓ ✓
E^2	3*64	Hidd Out	48 128	ReLU ReLU	✓ ✓
E^3	3*128	Hidd Out	96 256	ReLU ReLU	✓ ✓
E^4	3*256	Hidd Out	192 512	ReLU ReLU	✓ ✓
H^1	d_{in}+64	Hidd Out	32 64	ReLU ReLU	✓ ✓
H^2	128+64	Hidd Out	48 128	ReLU ReLU	✓ ✓
H^3	256+128	Hidd Out	96 256	ReLU ReLU	✓ ✓
H^4	515+256	Hidd Out	192 512	ReLU ReLU	✓ ✓
G	64+128+256+512	Out	512	ReLU	✓
P	512	Hidd1 Hidd2** Out	256 128 $d_{classes}$	ReLU ReLU	✓ ✓

Table 4. MLP dimensions, activations, and normalizations per layer. Input of second hidden layers and output layers, is the same as output of preceding layer. * This layer has no bias parameters. **This layer has a Dropout rate of 0.3.