12-11-2023

On the Analysis of Non-euclidean data: Sparsification, Classification and Generation

Yang Ye

On the Analysis of Non-Euclidean Data: Sparsification, Classification and Generation

by

Yang Ye

Under the Direction of Shihao Ji, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2023

ABSTRACT

In light of the groundbreaking achievements of convolutional neural networks (CNNs) in 2D image processing, there has been a pronounced interest in adapting these methods to non-Euclidean data, such as graphs and 3D geometric data. Point clouds, in particular, present unique challenges as they are sparse, unordered, and locality-sensitive, making the adaptation of CNNs to point cloud processing a non-trivial task. Similar challenges are encountered in the context of graph data. Consequently, the exploration of extending successful neural processing paradigms from 2D images to these non-Euclidean domains has emerged as a vibrant and dynamic research area.

This thesis focuses on advancing graph neural networks (GNNs) and analyzing 3D point clouds, emphasizing sparsification, classification and generation. For graph neural networks, a significant contribution is the introduction of Sparse Graph Attention Networks (SGAT), integrating a sparse attention mechanism into graph attention networks (GATs) through L_0-norm regularization. SGAT excels in edge removal (50%-80% on large graphs), enhancing interpretability without compromising performance on assortative graphs and improving it on disassortative graphs. In 3D point cloud analysis, an autoregressive approach, APSNet, formulates task-oriented point cloud sampling as a sequential generation process, and develops an attention-based point cloud sampling network that optimally samples 8 points out of 1024, tailoring the process for tasks like 3D point cloud classification, reconstruction, and registration. Extending into a non-autoregressive method, PTSNet, a point transformer, utilizes a transformer-based dynamic query generator. This innovation enables PTSNet to capture long-range correlations, mitigating issues like gradient vanishing and reducing duplicate samples compared to LSTM-based methods. Lastly, the thesis proposes GDPNet, first hybrid Generative and Discriminative PointNet, extending the Joint Energy-based Model (JEM) for point cloud generation and classification. GDPNet retains strong discriminative power of modern PointNet classifiers, while generating point cloud samples rivaling state-of-the-art generative approaches.

INDEX WORDS: Graph Neural Networks, Point cloud analysis, Sparsification, Classification, Attention, Energy-based model generation.

On the Analysis of Non-euclidean data: Sparsification, Classification and Generation

by

Yang Ye

DEDICATION

This dissertation is dedicated to my parents, Yonghai Ye and Huanyun Kang, for their unconditional love and endless support.

ACKNOWLEDGMENTS

I would like to express my profound gratitude to my advisor, Dr. Jonathon Shihao Ji, whose unwavering support has been the cornerstone of my Ph.D. journey. His wealth of knowledge, boundless patience, and insightful guidance were instrumental not only during the rigorous phases of research but also in shaping the narrative of this dissertation.

I extend sincere appreciation to my esteemed thesis committee members: Dr. Raj Sunderraman, Dr. Murray Patterson, and Dr. Xiaojing Ye. Their invaluable insights, constructive critiques, and thought-provoking questions compelled me to delve deeper into the intricacies of my research, ultimately enhancing its quality.

A special acknowledgment is reserved for my dedicated colleagues in the research group, including Dr. Xiang Li, Dr. Xiulong Yang, Yang Li, Qing Su, Hui Ye, and Mingchen Li. Additionally, heartfelt thanks to my friends Dr. Jiannan Ouyang, Mu Ge, and Dr. Jishen Yang for their unwavering support, both professionally and personally. Their collective encouragement played a pivotal role in the successful execution of this research.

I extend a special note of appreciation to my friend Wenjing Li for her exceptional support.

My gratitude also extends to everyone who contributed to the completion of this dissertation. While I regret not being able to individually mention each person involved, please accept my heartfelt thanks. This research would not have been possible without the collaborative support and encouragement of these individuals, and for that, I am truly grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Graph Neural Networks

Graph-structured data is ubiquitous in many real-world systems, such as social networks Tang et al. (2015), biological networks Zitnik & Leskovec (2017), and citation networks Sen et al. (2008), etc. Graphs can capture interactions (i.e., edges) between individual units (i.e., nodes) and encode data from irregular or non-Euclidean domains to facilitate representation learning and data analysis. Many tasks, from link prediction Van den Berg et al. (2017), graph classification Duvenaud et al. (2015) to node classification Yang et al. (2016), can be naturally performed on graphs, where effective node embeddings that can preserve both node information and graph structure are required. To learn from graph-structured data, typically an encoder function is needed to project high-dimensional node features into a low-dimensional embedding space such that "semantically" similar nodes are close to each other in the low-dimensional Euclidean space (e.g., by dot product) Hamilton et al. (2017b).

Recently, various Graph Neural Networks (GNNs) have been proposed to learn such embedding functions Scarselli et al. (2009); Bruna et al. (2014); Defferrard et al. (2016); Kipf & Welling (2017); Hamilton et al. (2017a,b); Veličković et al. (2018); Chen et al. (2020a). Traditional node embedding methods, such as matrix factorization Cao et al. (2015); Ou et al. (2016) and random walk Perozzi et al. (2014); Grover & Leskovec (2016), only rely on adjacent matrix (i.e., graph structure) to encode node similarity. Training in an unsupervised way, these methods employ dot product or co-occurrances on short random walks over graphs to measure the similarity between a pair of nodes. Similar to word embeddings Mikolov et al. (2013); Pennington et al. (2014); Ji et al. (2016), the learned node embeddings from these methods are simple look-up tables. Other approaches exploit both graph structure and node features in a semi-supervised training procedure for node embeddings Defferrard et al. (2016); Kipf & Welling (2017); Hamilton et al. (2017a); Veličković et al. (2018). These methods can be classified into two categories based on how they manipulate the adjacent matrix: (1) spectral graph convolution networks Scarselli et al. (2009); Bruna et al. (2014); Defferrard et al. (2016), and (2) neighbor aggregation or message passing algorithms Kipf & Welling (2017); Hamilton et al. (2017a); Veličković et al. (2018). Spectral graph convolution networks transform graphs to the Fourier domain, effectively converting convolutions over the whole graph into element-wise multiplications in the spectral domain. However, once the graph structure changes, the learned embedding functions have to be retrained or finetuned. On the other hand, the neighbor aggregation algorithms treat each node separately and learn feature representation of each node by aggregating (e.g., weighted-sum) over its neighbors' features. Under the assumption that connected nodes should share similar feature representations, these message passing algorithms leverage local feature aggregation to preserve the locality of each node, and is a generalization of classical convolution operation on images to irregular graph-structured data. For both categories of GNN algorithms, they can stack k layers on top of each other and aggregate features from k-hop neighbors.

1.2 3D Point Analysis

With the rapid development of 3D sensing devices (e.g., LiDAR and RGB-D camera), huge point cloud data are generated in the areas of robotics, autonomous driving and virtual reality Nüchter & Hertzberg (2008); Geiger et al. (2012); Park et al. (2008). A 3D point cloud, composed of the raw coordinates of scanned points in a 3D space, is an accurate representation of an object or shape and plays a key role in perception of the surrounding environment. Since point clouds lie in irregular space with variable densities, traditional feature extraction methods, such as convolutional neural networks (CNNs), designed for grid-structured 2D data do not perform well on 3D point clouds. Some methods attempt to first stiffly transform point clouds into grid-structured data and then take advantage of CNNs for feature extraction, such as projection-based methods Simony et al. (2018); Beltrán et al. (2018) and volumetric convolution-based methods Engelcke et al. (2017); Li (2017). Because placing a point cloud on a regular grid generates an uneven number of points in grid cells, applying the same convolution operation on such grid cells leads to information loss in crowded cells and wasting computation in empty cells. Recently, many methods of directly processing point cloud Qi et al. (2017b); Yu et al. (2018); Li et al. (2018d); Qi et al. (2019) have been proposed to enable efficient computation and performances in many applications, such as 3D point cloud classification Qi et al. (2017b); Li et al. (2018d); Thomas et al. (2019); Wu et al. (2019b), semantic segmentation Li et al. (2018b); Su et al. (2018); Liu et al. (2019); Wang et al. (2019b, 2018) and reconstruction Achlioptas et al. (2018); Yang et al. (2018); Han et al. (2019); Zhao et al. (2019), have been improved significantly. These methods take raw point clouds as input (without quantization) and aggregate local features at the last stage of the network, so the accurate data locations are kept intact but the computation cost grows linearly with the number of points.

Despite the significant progress of discriminative models for the tasks of 3D point cloud classification and segmentation, the research for generative models for point clouds are still far behind discriminative ones. Learning generative models for point clouds is crucial for point clouds analysis and characterizing the data distribution, which lays the foundation for various tasks such as shape completion, upsampling, synthesis and data augmentation. Although generative models such as variational auto-encoders (VAEs) Kingma & Welling (2014) and generative adversarial networks (GANs) Goodfellow et al. (2014) have shown great success in 2D image generation, it's quite challenging to extend these well-established methods to unordered 3D point clouds. Images are structured data but point clouds lie in irregular space with variable densities. Existing works on 3D generative models are mainly based on volumetric data, e.g., 3D ShapeNet Wu et al. (2015a), 3D GAN Wu et al. (2016), Generative VoxelNet Xie et al. (2018, 2020b), 3D-INN Huang et al. (2019), etc. While remarkable progress has been made, these methods have some inherent limitations for modeling point clouds. For instance, the training procedure could be unstable for GANs due to the adversarial losses. Auto-regressive models assume a generation ordering which is unnatural and might restrict the model's flexibility.

1.3 Dissertation Organization

The overall structure of this article is organized as below. To begin with, we briefly introduce Graph Neural Networks (GNNs) and 3d point analysis in Chapter 1. Chapter 2 introduces Sparse Graph Attention Networks (SGATs) that integrate a sparse attention mechanism into graph attention networks (GATs) via an L0-norm regularization to remove noisy edges and improve graph interpretability. Chapter 3 introduces an attention-based point cloud sampling network (APSNet), which formulates the task-oriented sampling 3D point clouds sampling as a sequential generation process and develop an attention-based auto-regressive network to solve this problem. Chapter 4 introduces an introduced transformer-based model, PTSNet, to capture long-range correlations, mitigating issues like gradient vanishing and reducing duplicate samples compared to LSTM-based methods. Chapter 5 introduces GDP-Net, the first hybrid Generative and Discriminative PointNet, extending the Joint Energy-based Model (JEM) for point cloud generation and classification. Finally, Chapter 6 concludes our work.

1.4 List of Publications

1. **Y. Ye**, S. Ji,”APSNet: Attention Based Point Cloud Sampling”.The 33rd British Machine Vision Conference (BMVC, Spotlight), 2022. Link

2. **Y. Ye**, S. Ji ” A Hybrid Generative and Discriminative PointNet on Unordered Point Sets”. Under review

3. **Y. Ye**, S. Ji "PTSNet: A Point Transformer for Task-oriented Point Cloud Sampling". Under review

4. **Y. Ye**, S. Ji, "Sparse graph attention networks". IEEE Transactions on Knowledge and Data Engineering (TKDE), Apr. 2021. Link

5. X. Yang, H. Ye, **Y. Ye**, X. Li, S. Ji, "Generative Max-Mahalanobis Classifiers for Image Classification, Generation and More". ECML, 2021 Link

6. K. Li, G. Luo, **Y. Ye** , W. Li, S. Ji, Z. Cai "Adversarial privacy-preserving graph embedding against inference attack". IEEE Internet of Things, Nov. 2020. Link

7. S. Srimath, **Y. Ye**, K. Sarker, R. Sunderraman, S. Ji "Human Activity Recognition from RGB Video Streams Using 1D-CNNs", IEEE Internet of People. 2021. Link

CHAPTER 2
Sparse Graph Attention Networks

2.1 Introduction

Among all the GNN algorithms, the neighbor aggregation algorithms Kipf & Welling (2017); Hamilton et al. (2017a); Veličković et al. (2018) have proved to be more effective and flexible. In particular, Graph Attention Networks (GATs) Veličković et al. (2018) use attention mechanism to calculate edge weights at each layer based on node features, and attend adaptively over all neighbors of a node for representation learning. To increase the expressiveness of the model, GATs further employ multi-head attentions to calculate multiple sets of attention coefficients for aggregation. Although multi-head attentions improve prediction accuracies, our analysis of the learned coefficients shows that multi-head attentions usually learn very similar distributions of attention coefficients (see Sec. 2.3.1 for details). This indicates that there might be a significant redundancy in the GAT modeling. In addition, GATs cannot assign an unique attention score for each edge because multiple attention coefficients are generated (from multi-heads) for an edge per layer and the same edge at different layers might receive different attention coefficients. For example, for a 2-layer GAT with 8-head attentions, each edge receives 16 different attention coefficients. The redundancy in the GAT modeling not only adds significant overhead to computation and memory usage but also increases the risk of overfitting. To mitigate these issues, we propose to simplify the architecture of GATs such that only one single attention coefficient is assigned to each edge across all GNN layers. To further reduce the redundancy among edges or remove noisy edges, we incorporate a sparsity constraint into the attention mechanism of GATs. Specifically, we optimize the model under an L_0-norm regularization to encourage model to use as fewer edges as possible. As we only employ one attention coefficient for each edge across all GNN layers, what we learn is an **edge-sparsified** graph with noisy/task-irrelevant edges removed. Our Sparse Graph Attention Networks (SGATs), as shown in Fig. 2.1, outperform the original GATs in two aspects: (1) SGATs simplify the architecture of GATs, and this reduces the risk of overfitting, and (2) SGATs can identify noisy/task-irrelevant edges[1] of a graph such that an edge-sparsified graph structure can be discovered, which is more robust for downstream classification tasks. As a result, SGAT is a robust graph learning algorithm that can learn from both assortative and disassortative graphs, while GAT fails on disassortative graphs.

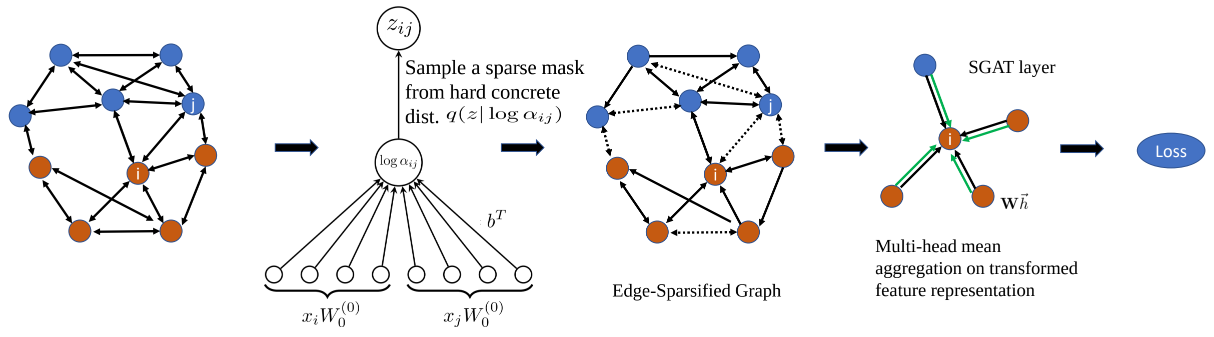
z_{ij}
Sample a sparse mask from hard concrete dist. $q(z|\log\alpha_{ij})$
$\log\alpha_{ij}$
b^T
$x_iW_0^{(0)}$
$x_jW_0^{(0)}$
Edge-Sparsified Graph
SGAT layer
$\mathbf{W}\vec{h}$
Multi-head mean aggregation on transformed feature representation
Loss

Figure 2.1 The overview of SGATs. By attaching a binary mask to each edge, SGATs utilize a sparse attention mechanism (as the output of the mask generator) to guide model to remove noisy/task-irrelevant edges and yield an edge-sparsified graph. In the plot above, the dashed lines denote removed edges. More details are described in Sec. 2.3.

[1]We call an edge task-irrelevant or noisy if removing it from graph incurs a similar or improved accuracy for downstream predictive tasks.

2.2 Background and Related Work

In this section, we first introduce our notation and then review prior works related to the neighbor aggregation methods on graphs. Let $G = (V, E)$ denote a graph with a set of nodes $V = \{v_1, \cdots, v_N\}$, connected by a set of edges $E \subseteq V \times V$. Node features are organized in a compact matrix $X \in \mathbb{R}^{N \times D}$ with each row representing the feature vector of one node. Let $A \in \mathbb{R}^{N \times N}$ denote the adjacent matrix that describes graph structure of G: $A_{ij} = 1$ if there is an edge e_{ij} from node i to node j, and 0 otherwise. By adding a self-loop to each node, we have $\tilde{A} = A + I_N$ to denote the adjacency matrix of the augmented graph, where $I_N \in \mathbb{R}^{N \times N}$ is an identity matrix.

For a semi-supervised node classification task, given a set of labeled nodes $\{(v_i, y_i), i = 1, \cdots, n\}$, where y_i is the label of node i and $n < N$, we learn a function $f(X, A, W)$, parameterized by W, that takes node features X and graph structure A as inputs and yields a node embedding matrix $H \in \mathbb{R}^{N \times D'}$ for all nodes in V; subsequently, H is fed to a classifier to predict the class label of each unlabeled node. To learn the model parameter W, we typically minimize an empirical risk over all labeled nodes:

$$\mathcal{R}(W) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f_i(X, A, W), y_i\right), \tag{2.1}$$

where $f_i(X, A, W)$ denotes the output of $f(X, A, W)$ for node i and $\mathcal{L}(\cdot)$ is a loss function, such as the cross-entropy loss that measures the compatibility between model predictions and class labels. Although there exist many different GNN algorithms that can solve Eq. 2.1, the main difference among them is how the encoder function $f(X, A, W)$ is defined.

2.2.1 Neighbor Aggregation Methods

The most effective and flexible graph learning algorithms so far follow a neighbor aggregation mechanism. The basic idea is to learn a parameter-sharing aggregator, which takes feature vector x_i of node i and its neighbors' feature vectors $\{x_j, j \in \mathcal{N}_i\}$ as inputs and outputs a new feature vector for node i. Essentially, the aggregator function aggregates lower-level features of a node and its neighbors and generates high-level feature representations. The popular Graph Convolution Networks (GCNs) Kipf & Welling (2017) fall into the category of neighbor aggregation. For a 2-layer GCN, its encoder function can be expressed as:

$$f(X, A, W) = \text{softmax}\left(\hat{A}\sigma(\hat{A}XW^{(0)})W^{(1)}\right), \tag{2.2}$$

where $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $W^{(\cdot)}$s are the learnable parameters of GCNs. Apparently, GCNs define the aggregation coefficients as the symmetrically normalized adjacency matrix $\hat{A}$, and these coefficients are shared across all GCN layers. More specifically, the aggregator of GCNs can be expressed as

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i} \hat{A}_{ij} h_j^{(l)} W^{(l)}\right), \tag{2.3}$$

where $h_j^{(l)}$ is the hidden representation of node j at layer l, $h^{(0)} = X$, and $\mathcal{N}_i$ denotes the set of all the neighbors of node i, including itself.

Since a fixed adjacency matrix $\hat{A}$ is used for feature aggregation, GCNs can only be used for the transductive learning tasks, and if the graph structure changes, the whole GCN model needs to be retrained or fine-tuned. To support inductive learning, GraphSage Hamilton et al. (2017a) proposes to learn parameterized aggregators (e.g., mean, max-pooling or LSTM aggregator) that can be used for feature aggregation on unseen nodes or graphs. To support large-scale graph learning tasks, GraphSage uniformly samples a fixed number of neighbors per node and performs computation on a sampled sub-graph at each iteration. Although it can reduce computational cost and memory usage significantly, its accuracies suffer from random sampling and partial neighbor aggregation.

2.2.2 Graph Attention Networks

Recently, attention networks have achieved state-of-the-art results in many computer vision and natural language processing tasks, such as image captioning Xu et al. (2015b) and machine translation Bahdanau et al. (2015). By attending over a set of inputs, attention mechanism can decide which parts of inputs to attend to in order to gather the most useful information. Extending the attention mechanism to graph-structured data, Graph Attention Networks (GATs) Veličković et al. (2018) utilize an attention-based aggregator to generate attention coefficients over all neighbors of a node for feature aggregation. In particular, the aggregator function of GATs is similar to that of GCNs:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} a_{ij}^{(l)} h_j^{(l)} W^{(l)}\right), \tag{2.4}$$

except that (1) $a_{ij}^{(l)}$ is the attention coefficient of edge e_{ij} at layer l, assigned by an attention function other than by a predefined $\hat{A}$, and (2) different layers utilize different attention functions, while GCNs share a predefined $\hat{A}$ across all layers. To increase the capacity of attention mechanism, GATs further exploit multi-head attentions for feature aggregation: each head works independently to aggregate information, and all the outputs of multi-heads are then concatenated to form a new feature representation for the next layer. In principle, the learned attention coefficient can be viewed as an importance score of an edge. However, since each edge receives multiple attention coefficients at a layer and the same edge at a different layer has a different set of attention coefficients, GATs cannot assign an unique importance score to quantify the significance of an edge.

Built on the basic framework of GATs, our SGATs introduce a sparse attention mechanism via an L_0-norm regularization for feature aggregation. Furthermore, we only assign one attention coefficient (or importance score) to each edge across all layers. As a result, we can identify important edges of a graph and remove noisy/task-irrelevant ones while retaining similar or sometimes even higher predictive performances on downstream classification tasks. Our results demonstrate that there is a significant amount of redundancies in graphs (e.g., 50%-80% of edges in assortative graphs like PPI and Reddit, and over 88% edges in disassortative graphs) that can be removed to achieve similar or improved classification accuracies.

2.2.3 Graph Sparsification

There are some prior works related to SGATs in terms of graph sparsification Calandriello et al. (2018); Chakeri et al. (2016); Rong et al. (2020); Hasanzadeh et al. (2020); Chen et al. (2020b); Zheng et al. (2020); Luo et al. (2021); Kim & Oh (2021). Spectral graph sparsification Calandriello et al. (2018); Chakeri et al. (2016) aims to remove unnecessary edges for graph compression. Specifically, it identifies a sparse subgraph whose Laplacian matrix can approximate the original Laplacian matrix well. However, these algorithms do not utilize node representations for graph compression and are not suitable for semi-supervised node classification tasks considered in the paper. DropEdge Rong et al. (2020) (and its Bayesian treatment Hasanzadeh et al. (2020)) propose to stochastically drop edges from graphs to regularize the training of GNNs. Specifically, DropEdge randomly removes a certain number of edges from an input graph at each training iteration to prevent the overfitting and over-smoothing issues Li et al. (2018c). At validation or test phase, DropEdge is disabled and the full input graph is utilized. This method shares the same spirit of Dropout Srivastava et al. (2014) and is an intuitive extension to graph structured data. However, DropEdge does not induce an edge-sparsified graph since different subsets of edges are removed at different training iterations and the full graph is utilized for validation and test, while SGAT learns an edge-sparsified graph by removing noisy/task-irrelevant edges permanently from input graphs. Because of these discrepancies, these methods are not directly comparable to SGAT. Recently, Chen et al. (2020b) propose LAGCN to add/remove edges based on the predictions of a trained edge classifier. It assumes the input graph is almost noisy free (e.g., assortative graphs) such that an edge classifier can be trained reliably from the existing graph topology. However, this assumption does not hold for very noisy (disassortative) graphs that SGAT can handle. NeuralSparse Zheng et al. (2020) learns a sparsification network to sample a k-neighbor subgraph (with a pre-defined k), which is then fed to GCN, GraphSage or GAT for node classification. Again, it does not aim to learn an edge-sparsified graph as the sparsification network produces a different subgraph sample each time and multiple subgraphs are used to improve accuracy. PTDNet Luo et al. (2021) proposes to improve the robustness and generalization performance of GNNs by learning to drop task-irrelevant edges. It samples a subgraph for each layer and applies a denoising layer before each GNN layer. Therefore, it cannot induce an edge-sparsified graph either. SuperGAT Kim & Oh (2021) improves GAT with an edge self-supervision regularization. It assumes that ideal attention should give all weights to label-agreed neighbors and introduces a layer-wise regularization term to guild attention with the presence or absence of an edge. However, when the graph is noisy, the regularization term will still push connected nodes to have same labels, and may generate suboptimal results.

Overall, none of these prior works induce an edge-sparsified graph while retaining similar or improved classification accuracies. Moreover, all of these algorithms are evaluated on assortative graphs with improved performance. But none of them (except SuperGAT Kim & Oh (2021)) has been evaluated on noisy disassortative graphs. As we will see when we present results, SGAT outperforms all of these state-of-the-arts on disassortative graphs and demonstrates its robustness on assortative and disassortative graphs.

2.3 Sparse Graph Attention Networks

The key idea of our Sparse Graph Attention Networks (SGATs) is that we can attach a binary gate to each edge of a graph to determine if that edge shall be used for neighbor aggregation or not. We optimize the SGAT model under an L_0 regularized loss function such that we can use as fewer edges as possible to achieve similar or better classification accuracies. We first introduce our sparse attention mechanism, and then describe how the binary gates can be optimized via stochastic binary optimization.

2.3.1 Formulation

To identify important edges of a graph and remove noisy/task-irrelevant ones, we attach a binary gate $z_{ij} \in \{0,1\}$ to each edge $e_{ij} \in E$ such that z_{ij} controls if edge e_{ij} will be used for neighbor aggregation or not[2]. This corresponds to attaching a set of binary masks to the adjacent matrix A:

$$\bar{A} = A \odot Z, \qquad Z \in \{0,1\}^M, \tag{2.5}$$

where M is the number of edges in graph G. Since we want to use as fewer edges as possible for semi-supervised node classification, we train model parameters W and binary masks Z by minimizing the following L_0-norm regularized empirical risk:

$$\begin{aligned} \mathcal{R}(W,Z) &= \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f_i(X, A \odot Z, W), y_i\right) + \lambda \|Z\|_0 \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f_i(X, A \odot Z, W), y_i\right) + \lambda \sum_{(i,j)\in E} 1_{[z_{ij}\neq 0]}, \end{aligned} \tag{2.6}$$

where $\|Z\|_0$ denotes the L_0-norm of binary masks Z, i.e., the number of non-zero elements in Z (edge sparsity), $1_{[c]}$ is an indicator function that is 1 if the condition c is satisfied, and 0 otherwise, and λ is a regularization hyperparameter that balances between data loss and edge sparsity. For the encoder function $f(X, A \odot Z, W)$, we define the following attention-based

[2]Note that edges e_{ij} and e_{ji} are treated as two different edges and therefore have their own binary gates z_{ij} and z_{ji}, respectively.

aggregation function:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} a_{ij} h_j^{(l)} W^{(l)}\right), \tag{2.7}$$

where a_{ij} is the attention coefficient assigned to edge e_{ij} across all layers. This is in a stark contrast to GATs, in which a layer-dependent attention coefficient $a_{ij}^{(l)}$ is assigned for each edge e_{ij} at layer l.

To compute attention coefficients, we simply calculate them by a row-wise normalization of $A \odot Z$, i.e.,

$$a_{ij} = \text{normalize}\left(A_{ij} z_{ij}\right) = \frac{A_{ij} z_{ij}}{\sum_{k \in \mathcal{N}_i} A_{ik} z_{ik}}. \tag{2.8}$$

Intuitively, the center node i is important to itself; therefore we set z_{ii} to 1 so that it can preserve its own information. Compared to GAT, we do not use softmax to normalize attention coefficients since by definition $z_{ij} \in \{0, 1\}$ and typically $A_{ij} \geq 0$ such that their product $A_{ij} z_{ij} \geq 0$.

Similar to GAT, we can also use multi-head attentions to increase the capacity of our model. We thus formulate a multi-head SGAT layer as:

$$h_i^{(l+1)} = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} a_{ij} h_j^{(l)} W_k^{(l)}\right), \tag{2.9}$$

where K is the number of heads, $\|$ represents concatenation, a_{ij} is the attention coefficients computed by Eq. 2.8, and $W_k^{(l)}$ is the weight matrix of head k at layer l. Note that only one set of attention coefficients a_{ij} is calculated for edge e_{ij}, and they are shared among all heads and all layers. With multi-head attention, the final returned output, $h_i^{(l+1)}$, consists of KD' features (rather than D') for each node.

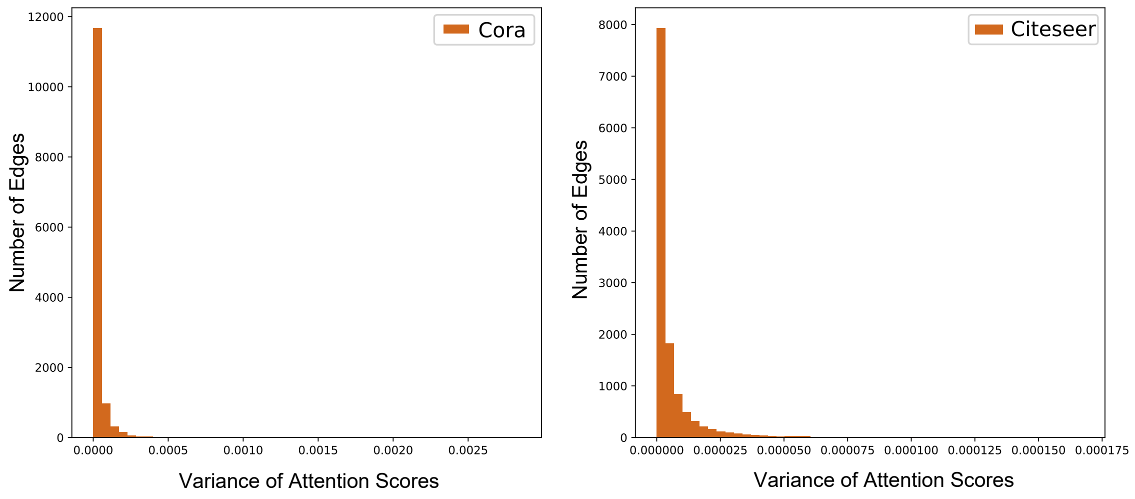

Figure 2.2 Histogram of variance of attention coefficients of a 2-layer GAT with a 8-head attention on the Cora and Citeseer datasets. The variances of attention coefficients of majority of edges are close to 0, indicating GAT learns similar distributions of attention scores from all heads and all layers.

Why can we use one set of coefficients for multi-head attention? This is based on our observation that all GAT heads tend to learn attention coefficients with similar distributions, indicating significant redundancy in the GAT modeling. For example, given a 2-layer GAT with a 8-head attention, each edge receives 16 attention coefficients, on which the variance can be calculated. Fig. 2.2 shows the histograms of variance of attention coefficients over all the edges in Cora and Citeseer, respectively[3]. As we can see, the variances of attention coefficients of majority of edges are close to 0, indicating GAT learns similar distributions of attention coefficients from different heads and from different GAT layers. This means using one set of attention coefficients might be enough for feature aggregation. In addition, using one set of attention coefficients isn't rare in GNNs as GCNs use a shared $\hat{A}$ across all layers and are very competitive to GATs in terms of classification accuracies. While

[3]Similar patterns are observed on the other datasets used in our experiments.

GCNs use one set of predefined aggregation coefficients, SGATs learn the coefficients from a sparse attention mechanism. We believe it is the learned attention coefficients instead of multi-set attention coefficients that leads to the improved performance of GATs over GCNs, and the benefit of multi-set attention coefficients might be very limited and could be undermined by the risk of overfitting due to increased complexity. Therefore, the benefits of using one set of attention coefficients over the original multi-set coefficients are at least twofold: (1) one set of coefficients is computationally K times cheaper than multiple sets of coefficients and is less prone to overfitting; and (2) one set of coefficients can be interpreted as edge importance scores such that they can be used to identify important edges and remove noisy/task-irrelevant edges for robust learning from real-world graph-structured data.

2.3.2 Model Optimization

Stochastic Variational Optimization To optimize Eq. 2.6, we need to compute its gradient w.r.t. binary masks Z. However, since Z is a set of binary variables, neither the first term nor the second term is differentiable. Hence, we resort to approximation algorithms to solve this binary optimization problem. Specifically, we approximate Eq. 2.6 via an inequality from stochastic variational optimization Bird et al. (2018): Given any function $\mathcal{F}(\boldsymbol{z})$ and any distribution $q(\boldsymbol{z})$, the following inequality holds:

$$\min_{\boldsymbol{z}} \mathcal{F}(\boldsymbol{z}) \leq \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z})}[\mathcal{F}(\boldsymbol{z})], \tag{2.10}$$

i.e., the minimum of a function is upper bounded by its expectation.

Since $z_{ij}\ \forall(i,j) \in E$ is a binary random variable, we assume z_{ij} is subject to a Bernoulli distribution with parameter $\pi_{ij} \in [0,1]$, i.e. $z_{ij} \sim \text{Ber}\left(z_{ij};\pi_{ij}\right)$. Thus, we can upper bound Eq. 2.6 by its expectation:

$$\tilde{\mathcal{R}}(W,\pi) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{q\left(Z|\pi\right)}\mathcal{L}\left(f_i(X, A \odot Z, W), y_i\right) + \lambda\sum_{(i,j)\in E}\pi_{ij}. \tag{2.11}$$

Now the second term of Eq. 2.11 is differentiable w.r.t. the new model parameters π. However, the first term is still problematic since the expectation over a large number of binary random variables Z is intractable, and thus its gradient does not allow for an efficient computation.

The Hard Concrete Gradient Estimator We therefore need further approximation to estimate the gradient of the first term of Eq. 2.11 w.r.t. π. Fortunately, this is a well-studied problem in machine learning and statistics with many existing gradient estimators for this discrete latent variable model, such as REINFORCE Williams (1992), Gumble-Softmax Jang et al. (2017), REBAR Tucker et al. (2017), RELAX Grathwohl et al. (2018) and the hard concrete estimator Louizos et al. (2018). We choose the hard concrete estimator due to its superior performance in our experiments and relatively straightforward implementation. Specifically, the hard concrete estimator employs a reparameterization trick to approximate the original optimization problem Eq. 2.11 by a close surrogate function:

$$\hat{\mathcal{R}}(W, \log\boldsymbol{\alpha}) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\boldsymbol{u}\sim\mathcal{U}(0,1)}\mathcal{L}\Big(f_i(X, A\odot g(f(\log\boldsymbol{\alpha},\boldsymbol{u})), W), y_i\Big) + \lambda \sum_{(i,j)\in E} \sigma\Big(\log\alpha_{ij} - \beta\log\frac{-\gamma}{\zeta}\Big) \tag{2.12}$$

with

$$f(\log\alpha, u) = \sigma((\log u - \log(1-u) + \log\alpha)/\beta)(\zeta-\gamma)+\gamma,$$

$$g(\cdot) = \min(1, \max(0, \cdot)),$$

where $\mathcal{U}(0,1)$ is a uniform distribution in the range of $[0,1]$, $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function, and $\beta = 2/3, \gamma = -0.1$ and $\zeta = 1.1$ are the typical parameter values of the hard concrete distribution. We refer the readers to Louizos et al. (2018) for more details of the hard concrete gradient estimator.

During training, we optimize $\log\alpha_{ij}$ for each edge e_{ij}. At the test phrase, we generate a deterministic mask $\hat{Z}$ by employing the following formula:

$$\hat{Z} = \min(\mathbf{1}, \max(\mathbf{0}, \sigma((\log\boldsymbol{\alpha})/\beta)(\zeta-\gamma)+\gamma)), \tag{2.13}$$

which is the expectation of Z under the hard concrete distribution $q(Z|\log\boldsymbol{\alpha})$. Due to the hard concrete approximation, $\hat{z}_{ij}$ is now a continuous value in the range of $[0,1]$. Ideally, majority of elements of $\hat{Z}$ will be zeros, and thus many edges can be removed from the graph.

Inductive Model of $\log \boldsymbol{\alpha}$ The learning of binary masks Z discussed above is transductive, by which we can learn a binary mask z_{ij} for each edge e_{ij} in the training graph G. However, this approach cannot generate new masks for edges that are not in the training graph. A more desired approach is inductive that can be used to generate new masks for new edges. This inductive model of $\log \boldsymbol{\alpha}$ can be implemented as a generator, which takes feature vectors of a pair of nodes as input and produces a binary mask as output. We model this generator simply as

$$\log \alpha_{ij} = (x_i W_0^{(0)} \| x_j W_0^{(0)}) b^T \tag{2.14}$$

where $b \in \mathcal{R}^{D'}$ is the parameter of the generator and $W_0^{(0)}$ is the weight matrix of head 0 at layer 0. To integrate this generator into an end-to-end training pipeline, we define this generator to output $\log \alpha_{ij}$. Upon receiving $\log \alpha_{ij}$ from the mask generator, we can sample a mask $\hat{z}_{ij}$ from the hard concrete distribution $q(z|\log \alpha_{ij})$. The set of sampled mask $\hat{Z}$ is then used to generate an edge-sparsified graph for the downstream classification tasks. Fig. 2.1 illustrates the full pipeline of SGATs. In our experiments, we use this inductive SGAT pipeline for semi-supervised node classification.

2.4 Evaluation

To demonstrate SGAT's ability of identifying important edges for feature aggregation, we conduct a series of experiments on synthetic and real-world (assortative and disassortative) semi-supervised node classification benchmarks, including transductive learning tasks and inductive learning tasks. We compare our SGATs with the state-of-the-art GNN algorithms:

GCNs Kipf & Welling (2017), GraphSage Hamilton et al. (2017a), GATs Veličković et al. (2018), SuperGAT Kim & Oh (2021), DropEdge Rong et al. (2020) and PTDNet Luo et al. (2021). For a fair comparison, our experiments closely follow the configurations of the competing algorithms. Our code is available at: `https://github.com/Yangyeeee/SGAT`.

Table 2.1 Summary of the graph datasets used in the experiments

	Type	Task	Nodes	Edges	Features	Classes	#Neighbors	$H(G)$
Cora	assortative	transductive	2,708	13,264	1,433	7	2.0	0.83
Citeseer	assortative	transductive	3,327	12,431	3,703	6	1.4	0.71
Pubmed	assortative	transductive	19,717	108,365	500	3	2.3	0.79
Amazon computers	assortative	transductive	13,381	505,474	767	10	18.4	0.79
Amazon photo	assortative	transductive	7,487	245,812	745	8	15.9	0.84
Actor	disassortative	transductive	7,600	60,918	931	5	4.4	0.24
Cornell	disassortative	transductive	183	737	1,703	5	1.6	0.11
Texas	disassortative	transductive	183	741	1,703	5	1.7	0.06
Wisconsin	disassortative	transductive	251	1,151	1,703	5	2.0	0.16
PPI	assortative	inductive	56,944	818,716	50	121	6.7	-*
Reddit	assortative	inductive	232,965	114,848,857	602	41	246.0	0.81

* PPI is a multi-label dataset, whose $H(G)$ can not be calculated.

2.4.1 Graph Datasets

Assortative and Disassortative Graphs Graph datasets can be categorized into assortative and disassortative ones Zachary (2002); Ribeiro et al. (2017) according to the node homophily in terms of class labels as introduced by Pei et al. (2020),

$$H(G) = \frac{1}{|V|}\sum_{v \in V} \frac{\text{Number of } v\text{'s neighbors of the same label}}{\text{Number of } v\text{'s neighbors}}.$$

Assortative graphs refer to ones with a high node homophily, where nodes within the local neighborhood provide useful information for feature aggregation. Common assortative graphs include citation networks and community networks. On the other hand, disassortative graphs contain nodes of the same label but not in their direct neighborhood, and therefore nodes within the local neighborhood provide more noises than useful information. Example disassortative graph datasets are co-occurrence networks and webpage hyperlink networks. We evaluate our algorithm on both types of graphs to demonstrate the robustness of SGATs on pruning redundant edges from clean assortative graphs and noisy edges from disassortative graphs. In our experiments, we consider seven established assortative and four disassortative graphs, whose statistics are summarized in Table 2.1.

Transductive Learning Tasks Three citation network datasets: Cora, Citeseer and Pubmed Sen et al. (2008) and two co-purchase graph datasets: Amazon Computers and Amazon Photo Shchur et al. (2018) are used to evaluate the performance of our algorithm in the transductive learning setting, where test graphs are *included* in training graphs for feature aggregation and thus facilitates the learning of feature representations of test nodes for classification. The citation networks have low degrees (e.g., only 1-2 edges per node), while the co-purchase datasets have higher degrees (e.g., 15-18 edges per node). Therefore, we can demonstrate the performance of SGAT on sparse graphs and dense graphs. For the citation networks, nodes represent documents, edges denote citation relationship between two documents, and node features are the bag-of-words representations of document contents; the goal is to classify documents into different categories. For the co-purchase datasets, nodes represent products, edges indicate two products are frequently purchased together, and node features are the bag-of-words representations of product reviews; similarly, the goal is to classify products into different categories. Our experiments closely follow the transductive learning setup of Kipf & Welling (2017); Shchur et al. (2018). For all these datasets, 20 nodes per class are used for training, 500 nodes are used for validation, and 1000 nodes are used for test.

For the four disassortative graphs, *Actor* Tang et al. (2009) is an actor co-occurrence network, where nodes denote actors and edges indicate co-occurrence of two actors on the same Wikipedia web page. Node features are the bag-of-word representation of keywords in the actors' Wikipedia pages. Each node is labeled with one of five classes according to the topic of the actor's Wikipedia page. *Cornell*, *Texas*, and *Wisconsin* come from the WebKB dataset collected by Carnegie Mellon University. Nodes represent web pages and edges denote hyperlinks between them. Node features are the bag-of-word representation of the corresponding web page. Each node is labeled with one of the five categories {student, project, course, staff, and faculty}. We follow Pei et al. (2020) to randomly split nodes of each class into 60%, 20%, and 20% for training, validation, and test. The experiments are repeatedly run 10 times with different random splits and the average test accuracy over these 10 runs are reported. Test is performed when validation accuracy achieves maximum on each run.

Inductive Learning Tasks Two large-scale graph datasets: PPI Zitnik & Leskovec (2017) and Reddit Hamilton et al. (2017a) are also used to evaluate the performance of SGAT in the inductive learning setting, where test graphs are *excluded* from training graphs for parameter learning, and the representations of test nodes have to be generated from trained aggregators for classification. In this case, our inductive experiments closely follow the setup of GraphSage Hamilton et al. (2017a). The protein-protein interaction (PPI) dataset consists of graphs corresponding to different human tissues. Positional gene sets, motif gene sets and immunological signatures are extracted as node features and 121 gene ontology categories are used as class labels. There are in total 24 subgraphs in the PPI dataset with each subgraph containing 3k nodes and 100k edges on average. Among 24 subgraphs, 20 of them are used for training, 2 for validation and the rest of 2 for test. For the Reddit dataset, it's constructed from Reddit posts made in the month of September, 2014. Each node represents a reddit post and two nodes are connected when the same user commented on both posts. The node features are made up with the embedding of the post title, the average embedding of all the post's comments, the post's score and the number of comments made on the post. There are 41 different communities in the Reddit dataset corresponding to 41 categories. The task is to predict which community a post belongs to. We use the same data split as in GraphSage Hamilton et al. (2017a), where the first 20 days for training and the remaining days for test (with 30% used for validation). This is a large-scale graph learning benchmark that contains over 100 million edges and about 250 edges per node, and therefore a high edge redundancy is expected.

2.4.2 Models and Experimental Setup

Models A 2-layer SGAT with a 2-head attention at each layer is used for feature aggregation, followed by a softmax classifier for node classification. We use ReLU Nair & Hinton (2011) as the activation function and optimize the models with the Adam optimizer Kingma & Ba (2015) with the learning rate of $lr = 1e-2$. We compare SGAT with the state-of-the-arts in terms of node classification accuracy. Since SGAT induces an edge-sparsified graph, we also report the percentage of edges removed from the original graph.

To investigate the effectiveness of sparse attention mechanism induced by the L_0-norm regularization, we also introduce a GAT with *top-k* attention baseline (named as GAT-2head-top-k). This baseline has the same architecture of SGAT-2head, which only uses one head's attentions to do neighbor aggregation. However, instead of using sparse attention coefficients induced by Eq. 2.8, we remove the *top-k* smallest attention coefficients calculated by GAT's dense attention function, with k set to remove the same percentage of edges induced by SGAT-2head. We report the best performance of GAT-2head-top-k from two training procedures: (1) removing the *top-k* smallest attention coefficients from the beginning of the training, (2) removing *top-k* smallest attention coefficients after the validation accuracies have converged. This GAT-2head-top-k baseline also produces an edge-sparsified graph, and thus serves as a fair comparison to SGAT-2head.

Hyperparameters We tune the performance of SGAT and its variants based on the hyperparameters of GAT since SGAT is built on the basic framework of GAT. For a fair comparison, we also run 1-head and 2-head GAT models with the same architecture as SGATs to illustrate the impact of sparse attention models vs. standard dense attention models. To prevent models from overfitting on small datasets, L_2 regularization and dropout Srivastava et al. (2014) are used. Dropout is applied to the inputs of all layers and the attention coefficients. For the large-scale datasets, such as PPI and Reddit, we do not use L_2 regularization or dropout as the models have enough data for training. We implemented our SGAT and its variants with the DGL library Wang et al. (2019a).

2.4.3 Experiments on Synthetic Dataset

To illustrate the idea of SGAT, we first demonstrate it on a synthetic dataset – Zachary's Karate Club Zachary (1977), which is a social network of a karate club of 34 members with links between pairs of members representing who interacted outside the club. The club was split into two groups later due to a conflict between the instructor and the administrator. The goal is to predict the groups that all members of the club joined after the split. This is a semi-supervised node classification problem in the sense that only two nodes: the instructor (node 0) and the administrator (node 33) are labeled and we need to predict the labels of all the other nodes. We train a 2-layer SGAT with a 2-head attention at each layer on

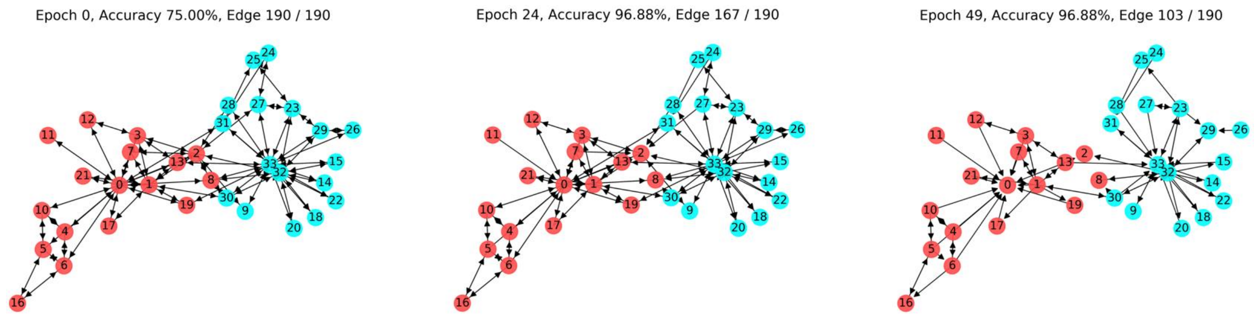
Epoch 0, Accuracy 75.00%, Edge 190 / 190
Epoch 24, Accuracy 96.88%, Edge 167 / 190
Epoch 49, Accuracy 96.88%, Edge 103 / 190

Figure 2.3 The evolution of the graph of Zachary's Karate Club at different training epochs. SGAT can remove 46% edges from the graph while retaining almost the same accuracy at 96.88%. Nodes 0 and 33 are the labeled nodes, and the colors show the ground-truth labels. The video can be found at `https://youtu.be/3Jhr26lXRl8`.

the dataset. Fig. 2.3 illustrates the evolution of the graph at different training epochs, the corresponding classification accuracies and number of edges kept in the graph. As can be seen, as the training proceeds, some insignificant edges are removed and the graph is getting sparser; at the end of training, SGAT removes about 46% edges while retaining an accuracy of 96.88% (i.e., only one node is misclassified), which is the same accuracy achieved by GCN and other competing algorithms that utilize the full graph for prediction. In addition, the removed edges have an intuitive explanation. For example, the edge from node 16 to node 6 is removed while the reversed edge is kept. Apparently, this is because node 6 has 4 neighbors while node 16 has only 2 neighbors, and thus removing one edge between them doesn't affect node 6 too much while may be catastrophic to node 16. Similarly, the edges between node 27, 28 and node 2 are removed. This might be because node 2 has an edge to node 0 and has no edge to node 33, and therefore node 2 is more like to join node 0's group and apparently the edges to nodes 27 and 28 are insignificant or might be due to noise.

Table 2.2 Classification accuracies on seven assortative graphs for semi-supervised node classification. Results of GCNs on PPI and Reddit are trained in a transductive way. The results annotated with * are from our experiments, and the rest of results are from the corresponding papers. OOM indicates "out of memory". Results are the averages of 10 runs.

Datasets	**Cora**	**Citeseer**	**Pubmed**	**Amazon computer**	**Amazon Photo**	**PPI**	**Reddit**
GCN Kipf & Welling (2017)	81.5%	70.3%	79.0%	81.5%*	91.2%*	50.9%*	94.38%*
GraphSage Hamilton et al. (2017a)	-	-	-	-	-	61.2%	95.4%
GAT Veličković et al. (2018)*	82.5%	70.9%	78.6%	81.5%	89.1%	98.3%	OOM
GAT-1head*	82.1%	70.8%	77.4%	81.3%	89.7%	85.6%	92.6%
GAT-2head*	83.5%	70.8%	78.3%	82.4%	90.4%	97.6%	93.5%
GAT-2head-top-k*	82.8%	70.9%	78.2%	77.5%	85.6%	95.5%	93.3%
SGAT-1head*	82.3%	70.6%	76.1%	81.1%	89.5%	93.6%	94.9%
SGAT-2head*	83.0%	71.5%	78.3%	81.8%	89.9%	97.6%	95.8%
Edge Removed*	2.0%	1.2%	2.2%	**63.6%**	**42.3%**	**49.3%**	**80.8%**

*From our experiments.
Note that DropEdge Rong et al. (2020) and PTDNet Luo et al. (2021) and SuperGAT Kim & Oh (2021) have reported improved accuracies on assortative graphs. Hence, we do not include their results in this table since only SGAT induces edge-sparsified graphs and we only claim SGAT achieves similar accuracies as GAT on these graphs.

2.4.4 Experiments on Assortative Graphs

Next we evaluate the performance of SGAT on seven assortative graphs, where nodes within the local neighborhood provide useful information for feature aggregation. In this case, some redundant or task-irrelevant edges may be removed from the graphs with no or minor accuracy losses. For a fair comparison, we run each experiments 10 times with different random weight initializations and report the average accuracies.

The results are summarized in Table 2.2. Comparing SGAT with GCN, we note that SGAT outperforms GCN on the PPI dataset significantly while being similar on all the other six benchmarks. Comparing SGAT with GraphSage, SGAT again outperforms GraphSage on PPI by a significant margin. Comparing SGAT with GAT, we note that they achieve very competitive accuracies on all six benchmarks except Reddit, where the original GAT is "out of memory" and SGAT can perform successfully due to its simplified architecture. Another advantage of SGAT over GAT is the regularization effect of the L_0-norm on the edges. To demonstrate this, we test two GAT variants: GAT-1head and GAT-2head that have the similar architectures as SGAT-1head and SGAT-2head but with different attention mechanisms (i.e., standard dense attention vs. sparse attention). As we can see, on the Reddit dataset, the sparse attention-based SGATs outperform GATs by 2-3% while sparsifying the graph by 80.8%. As discussed earlier, to evaluate the effectiveness of sparse attention mechanism of SGAT further, we also introduce a baseline (GAT-2head-top-k), which has the same architecture of SGAT-2head but removes the *top-k* smallest coefficients calculated by GAT's dense attention function, with k set to remove the same number of redundant edges induced by SGAT. As we can see from Table 2.2, SGAT-2head outperforms GAT-2head-top-k by 1%-4% accuracies on larger benchmarks (Amazon computer, Amazon Photo, PPI and Reddit) when a large percentage of edges is removed, demonstrating the superiority of

SGAT's sparse attention mechanism over the naive top-k attention coefficients selection as used in GAT-top-k.

Overall, SGAT is very competitive against GCN, GraphSage and GAT in terms of classification accuracies, while being able to remove certain percentages of redundant edges from small and large assortative graphs. Specifically, on the three small citation networks: Cora, Citeseer and Pubmed, SGATs learn that majority of the edges are critical to maintain competitive accuracies as the original graphs are already very sparse (e.g., numbers of edges per node are 2.0, 1.4, 2.3, respectively. See Table 2.1), and therefore SGATs remove only 1-2% of edges. On the other hand, on the rest of large or dense assortative graphs, SGATs can identify significant amounts of redundancies in edges (e.g., 40-80%), and removing them incurs no or minor accuracy losses.

2.4.5 Experiments on Disassortative Graphs

As shown in Table 2.1, the $H(G)$ of disassortative graphs are around 0.1-0.2. This means the graphs are very noisy, i.e., a node and majority of its neighbors have different labels. In this case, the neighbor aggregation mechanism of GAT, GCN and GraphSage would aggregate noisy features from neighborhood and fail to learn good feature representations for the downstream classification tasks, while SGAT may excel due to its advantage of pruning noisy edges in order to achieve a high predictive performance.

To verify this, we compare SGAT with GAT, Geom-GCN Pei et al. (2020), MLP, DropEdge, SuperGAT and PTDNet on the four disassortative graphs. Geom-GCN is a variant of GCN that utilizes a complicated node embedding method to identify similar nodes and

Table 2.3 Classification accuracies of different node classification algorithms on four disassortative graphs. Results are the averages of 10 runs.

Datasets	**Actor**	**Cornell**	**Texas**	**Wisconsin**
MLP*	35.1	81.6	81.3	84.9
GAT Veličković et al. (2018)*	34.6	55.9	55.4	53.5
SuperGAT Kim & Oh (2021)*	30.4	57.6	61.1	60.1
DropEdge-GCN Rong et al. (2020)*	30.6	54.5	61.5	59.8
Geom-GCN Pei et al. (2020)	31.6	60.8	67.6	64.1
PTDNet-GCN Luo et al. (2021)*	35.6	80.3	82.2	84.9
SGAT-2head*	**35.7**	**82.4**	**86.2**	**86.1**
Edge Removed*	88.1%	93.9%	95.0%	91.9%

*From our experiments.

create an edge between them, such that it can aggregate features from informative nodes and outperform GCN on the disassortative graphs. We also consider an MLP model as baseline, which makes prediction solely based on the node features without aggregating any local information. For a fair comparison, the GAT and MLP have a similar model capacity as that of SGAT-2head. We also compare SGAT with the state-of-the-art robust GNN models that we discussed in related work: SuperGAT Kim & Oh (2021)[4], DropEdge Rong et al. (2020)[5], and PTDNet Luo et al. (2021)[6]. Since none of them reported the performance on these disassortative graphs, we follow the same experimental settings discussed above and run their open source implementations.

The results are shown in Table 2.3. It can be observed that MLP outperforms GAT and the majority of algorithms considered, indicating that local aggregation methods fail to get a good performance due to the noisy neighbors. The robust GNN algorithms: SuperGAT and DropEdge achieve better performance than GAT in general but are still worse than MLP

[4]https://github.com/dongkwan-kim/SuperGAT
[5]https://github.com/DropEdge/DropEdge
[6]https://github.com/flyingdoog/PTDNet

since the extremely noisy neighbors violate the label-agreement assumption of SuperGAT or beyond the noise level that simple drop edge can handle. On the other hand, PTDNet achieves a competitive performance with MLP, demonstrating that PTDNet's denoising layers and layer-wise subgraph sampling are indeed very effective. Among all the algorithms considered, SGAT achieves the best accuracies on the disassortative graphs. As shown in the last row of Table 2.3, on all the dissassortative graphs SGAT tends to remove majority of edges from the graphs, and only less than 10% edges are kept for feature aggregation, which explains its superior performance on these noisy disassortative graphs.

Overall, SGAT is a much more robust algorithm than GAT (and in many cases other competing methods) on assortative and disassortative graphs since it can detect and remove noisy/task-irrelevant edges from graphs in order to achieve similar or improved accuracies on the downstream classification tasks.

2.4.6 Analysis of Removed Edges

We further analyze the edges removed by SGAT. Fig. 2.4 illustrate the evolution of classification accuracy and number of edges kept by SGAT as a function of training epochs on the Cora, PPI and Texas test datasets. As we can see, SGAT removes 2% edges from Cora slowly during training (as Cora is a sparse graph), while it removes 49.3% edges from PPI and over 88.1% edges from Texas rapidly, indicating a significant edge redundancy in PPI and Texas.

To demonstrate SGAT's accuracy of identifying important edges from a graph, Fig. 2.5 shows the evolution of classification accuracies on the PPI test dataset when different per-

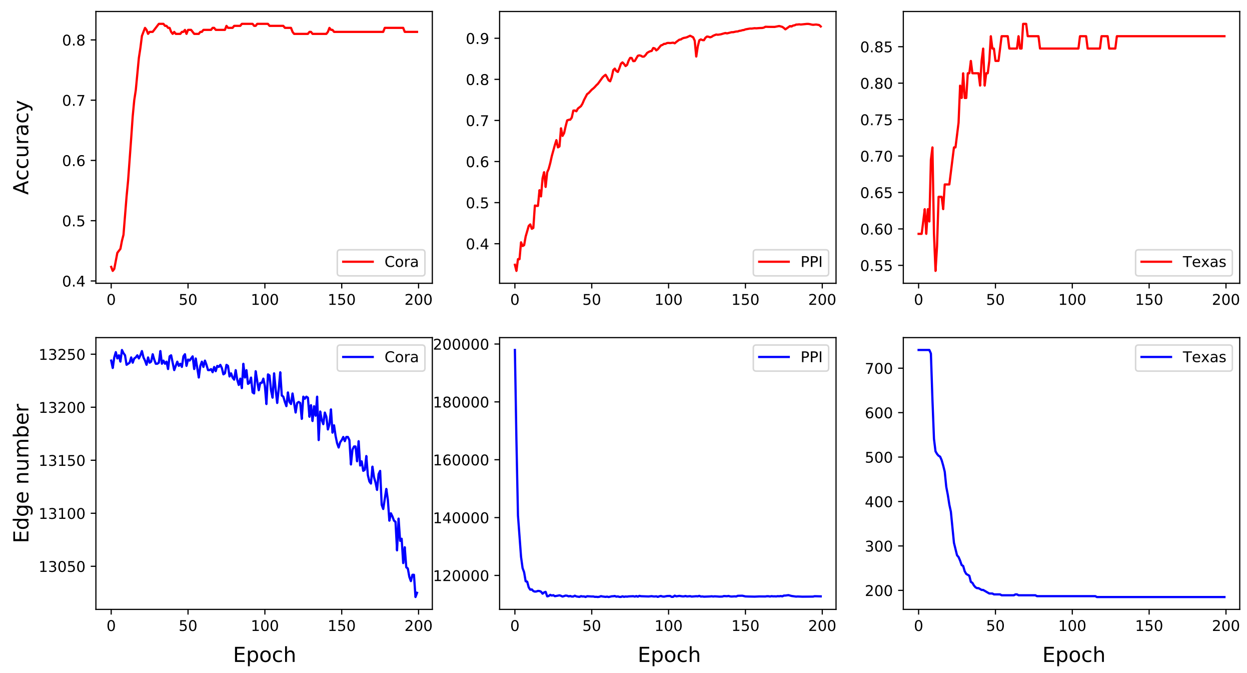

Figure 2.4 The evolution of classification accuracy (top) and number of kept edges (bottom) as a function of training epochs on the Cora, PPI and Texas test datasets.

centages of edges are removed from the graph. We compare three different strategies of selecting edges for removal: (1) top-k% edges sorted descending by $\log \alpha_{ij}$, (2) bottom-k% edges sorted descending by $\log \alpha_{ij}$, and (3) uniformly random k%. As we can see, SGAT identifies important edges accurately as removing them from the graph incurs a dramatically accuracy loss as compared to random edge removal or bottom-k% edge removal.

2.4.7 Hyperparameter Tuning

SGAT has a few important hyperparameters, which affect the performance of SGAT significantly. In this section, we demonstrate the impact of them and discuss how we tune the hyperparameters for performance trade-off. One of the most important hyperparameters of SGAT is the λ in Eq. 2.6, which balances the classification loss (the first term) and edge

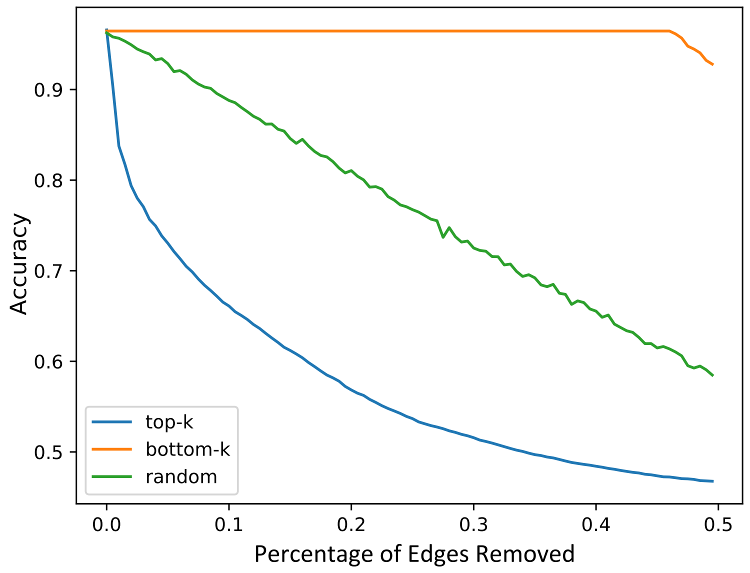

Figure 2.5 The evolution of classification accuracies on the PPI test dataset when different percentages of edges are removed from the graph. Three different strategies of selecting edges for removal are considered.

sparsity (the second term). As λ increases, the L_0 sparsity regularization gets stronger. As a result, a large number of edges will be pruned away (i.e., z=0), but potentially it will incur a lower classification accuracy if informative edges are removed (i.e., over-pruning). We therefore select a λ to yield the highest edge prune rate, while still achieving a good predictive performance on the downstream classification tasks. Fig. 2.6 shows the results of tuning λ on the PPI (top) and Texas (bottom) validation datasets. It can be observed that as λ increases, more edges are removed from the PPI and Texas datasets. However, the classification accuracies have different trends on PPI and Texas. As more edges are removed from PPI, the accuracy retains almost no changes when $\lambda \leq 2e-6$, and drops significantly when $\lambda > 2e$–6. This is because PPI is an assortative graph, in which local neighborhood provides useful information for feature aggregation, and pruning them from the graph in general incurs no or minor accuracy loss until a large λ that leads to over-pruning. In contrast, as more edges are removed from Texas, the accuracy increases at beginning when $\lambda \leq 5e{-}3$ and plateaus afterwards. This is because Texas is a disassortative graph, in which local neighborhood provides more noise than useful information for feature aggregation, and pruning noisy edges from the graph typically improves classification accuracy until a large λ that leads to over-pruning. Similar patterns are observed on the other assortative and disassortative graphs used in our experiments. Based on the results in Fig. 2.6, we choose $\lambda = 2e-6$ for PPI and $\lambda = 5e{-}3$ for Texas as they achieve the best balance between classification accuracy and edge sparsity.

Another important hyperparameter of SGAT is the number of heads K in Eq. 2.9. As K increases, SGAT has more capacity to learn from the data, but is more prone to overfitting. This is demonstrated in Fig. 2.7, where we present the classification accuracies of SGAT on PPI and Texas as K increases. As we can see, when $K = 2$ SGAT achieves the best (or close to best) accuracies on both datasets. Similar trends are also observed on the other datasets. Therefore, in our experiments we choose $K = 2$ as the default.

2.4.8 Visualization of Learned Features

Finally, we visualize the learned feature representations from the penultimate layer[7] of GAT and SGAT with t-SNE Maaten & Hinton (2008). The results on Cora and Texas are shown in Fig. 2.8. It can be observed that SGAT and GAT learn similar representations on Cora when the graph is nearly noisy-free (e.g., assortative graphs), while SGAT learns a better

[7]The layer before the final FC layer for classification.

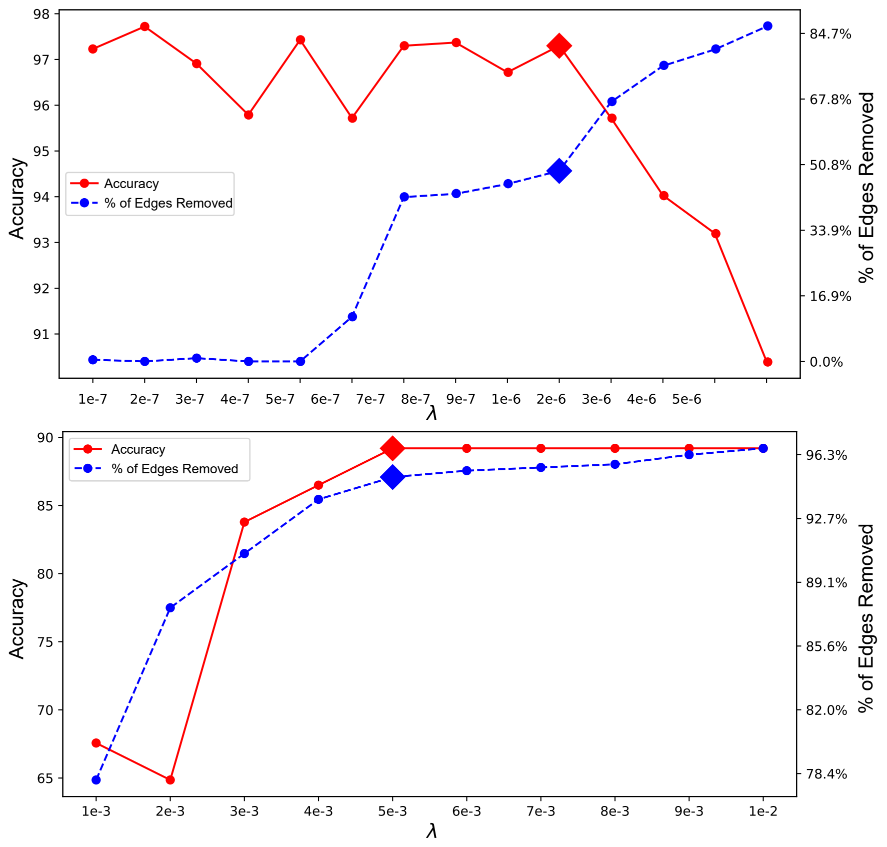

Figure 2.6 The impact of λ to the classification accuracy and edge sparsity on the PPI (top) and Texas (bottom) validation dataset.

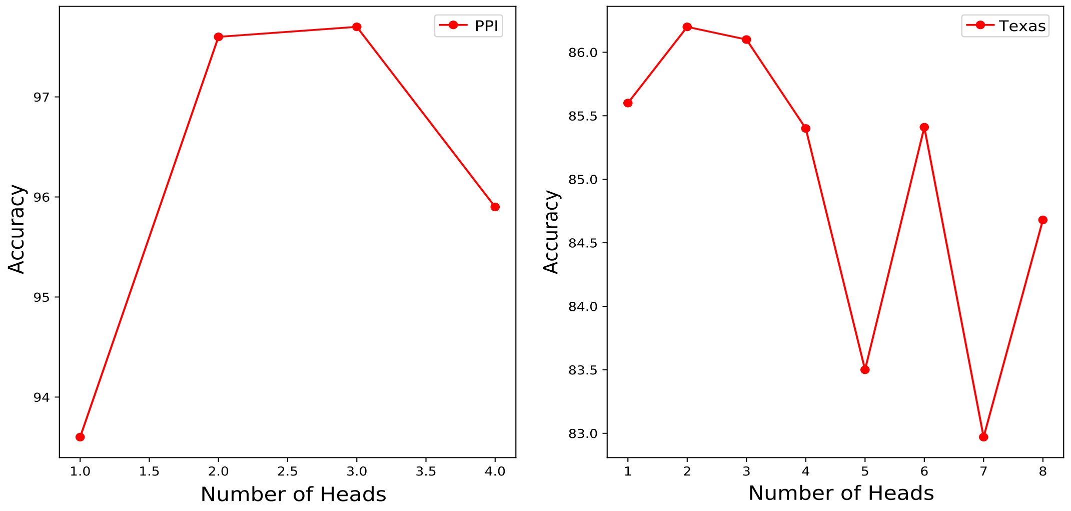

Figure 2.7 The evolution of classification accuracy of SGAT as a function of number of heads on the PPI and Texas datasets.

representation with a higher class separability than GAT on Texas when the graph is very noisy (e.g., disassortative graphs), demonstrating the robustness of SGAT on learning from assortative and disassortative graphs.

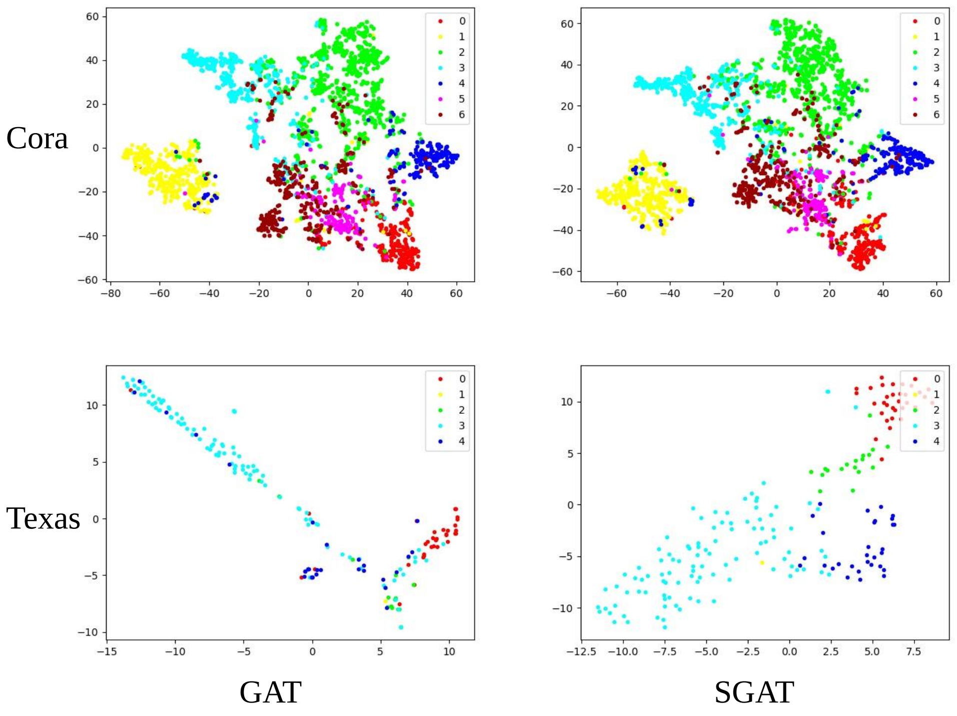

Figure 2.8 t-SNE visualization of learned feature representations on Cora and Texas.

2.4.9 Discussion

Given a similar architecture and the same number of heads, one may expect that SGAT would be faster and more memory efficient than GAT since a large portion of edges can be removed by SGAT. However, our empirical study shows that both algorithms have a similar overall runtime and memory consumption. This is because learning sparse attention coefficients has the similar complexity as learning standard dense attention coefficients and storing feature representations (other than A and Z) consumes most of memory. Therefore, even though SGAT can remove a large portion of edges from a graph, it isn't faster or more memory efficient than GAT.

One potential speed up of SGAT is that we can skip the computation associated with edges of $z \approx 0$ during training. However, this heuristic will be an approximation because an edge with $z \approx 0$ may be reactivated in later iterations during stochastic optimization, and this potentially will cause accuracy drop. We leave this as our future work.

In summary, the main advantage of SGAT is that it can identify noisy/task-irrelevant edges from both assortative and disassortative graphs to achieve a similar or improve classification accuracy, while the conventional GAT, GCN and GraphSage fail on noisy disassortative graphs due to their local aggregation mechanism. The robustness of SGAT is of practical importance as real-world graph-structured data are often very noisy, and a robust graph learning algorithm that can learn from both assortative and disassortative graphs is very critical.

2.5 Conclusion

In this chapter, we propose sparse graph attention networks (SGATs) that integrate a sparse attention mechanism into graph attention networks (GATs) via an L_0-norm regularization on the number of edges of a graph. To assign a single attention coefficient to each edge, SGATs further simplify the architecture of GATs by sharing one set of attention coefficients across all heads and all layers. This results in a robust graph learning algorithm that can detect and remove noisy/task-irrelevant edges from a graph in order to achieve a similar or improved accuracy on downstream classification tasks. Extensive experiments on seven assortative graphs and four disassortative graphs demonstrate the robustness of SGAT.

As for future extensions, we plan to investigate the applications of SGATs on detecting superficial or malicious edges injected by adversaries. We also plan to explore the application of sparse attention network of SGATs in unsupervised graph domain adaption (e.g. Wu et al. (2020)) to improve inter-graph attention.

CHAPTER 3
APSNet: Attention Based Point Cloud Sampling

3.1 Introduction

Processing large-scale dense 3D point clouds is challenging due to the high cost of storing, transmitting and processing these data. Point cloud sampling, a task of selecting a subset of points to represent the original point clouds at a sparse scale, can reduce data redundancy and improve the efficiency of 3D data processing. So far, there are a few heuristics-based sampling methods, such as random sampling (RS), farthest point sampling (FPS) Eldar et al. (1997); Moenning & Dodgson (2003), and grid (voxel) sampling Wu et al. (2015b); Qi et al. (2016). However, all of these methods are task-agnostic as they do not take into account the subsequent processing of the sampled points and may select non-informative points to the downstream tasks, leading to suboptimal performance. Recently, S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020) are proposed, which demonstrate that better sampling strategies can be learnt via a task-oriented sampling network. These sampling networks can generate a small number of samples that optimize the performance of a downstream task, and outperform traditional task-agnostic samplers significantly in various applications Dovrat et al. (2019a); Lang et al. (2020).

We argue that point cloud sampling can be considered as a sequential generation process, in which points to be sampled next should depend on the points that have already been sampled. However, existing task-oriented sampling methods, such as S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020), do not pay enough attention to the sample dependency and generate all samples in one shot (without parameter reusing or sharing when generating samples of different sizes). In this chapter, we propose an attention-based point cloud sampling network (APSNet) for task-oriented sampling, which enables a FPS-like sequential sampling but with a task-oriented objective. Specifically, APSNet employs a novel LSTM-based sequential model to capture the correlation of points with a global attention. The feature of each point is extracted by a simplified PointNet architecture, followed by an LSTM Hochreiter & Schmidhuber (1997) with attention mechanism to capture the relationship of points and select the most informative ones for sampling. Finally, the sampled point cloud is fed to a (frozen) task network for prediction. The whole pipeline is fully differentiable and the parameters of APSNet can be trained by optimizing a task loss and a sampling loss jointly (See Fig. 3.1).

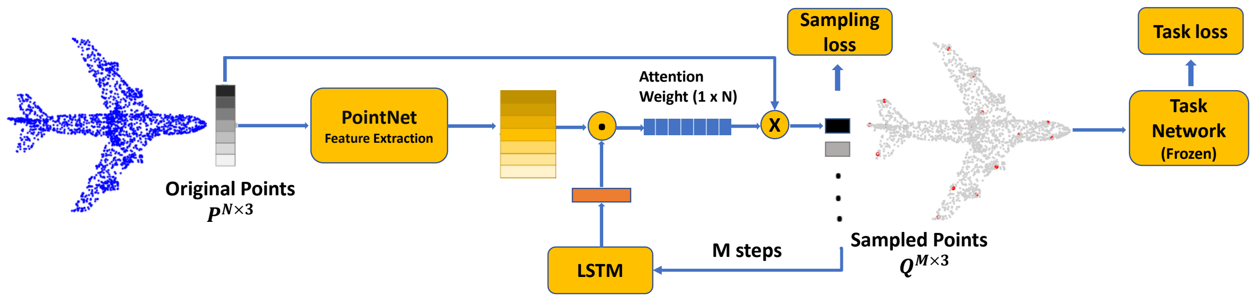

Figure 3.1 Overview of APSNet. APSNet first extracts features with a simplified PointNet that preserves the geometric information of a point cloud. Then, an LSTM with attention mechanism is used to capture the relationship among points and select the most informative point sequentially. Finally, the sampled point cloud is fed to a task network for prediction. The whole pipeline is optimized by minimizing a task loss and a sampling loss jointly.

Depending on the availability of labeled training data, APSNet can be trained in supervised learning or self-supervised learning via knowledge-distillation Hinton et al. (2014). In the latter case, no ground truth label is needed for the training of APSNet. Instead, the soft predictions of task network are leveraged to train APSNet. Interestingly, the self-supervised training of APSNet achieves impressive results that are close to the performance of supervised training. This makes APSNet widely applicable in situations where only a deployed task net is available but the original labeled training dataset of the task net is no longer accessible.

In addition, given the autoregressive model of APSNet, our method can generate arbitrary length of samples from a single model. This entails an effective joint training of APSNet with multiple sample sizes, resulting in a single compact model for point cloud sampling, while S-NET and SampleNet require a growing model size to generate larger sized point samples and the parameter reusing or sharing is not as effective as APSNet Dovrat et al. (2019a); Lang et al. (2020). Our main contributions are summarized as follows:

1. We propose APSNet, a novel attention-based point cloud sampling network, which enables a FPS-like sequential sampling with a task-oriented objective.
2. APSNet can be trained in supervised learning or self-supervised learning via knowledge-distillation, while the latter requires no ground truth labels for training and is thus widely applicable in situations where only a deployed task network is available.
3. APSNet can be jointly trained with multiple sample sizes, yielding a single compact model that can generate arbitrary length of samples with prominent performance.
4. Compared with state-of-the-art sampling methods, APSNet demonstrate superior performance on various 3D point cloud applications.

3.2 Related Work

Deep Learning on Point Clouds

Following the breakthrough results of CNNs in 2D image processing tasks Krizhevsky et al. (2012); He et al. (2016a), there has been a strong interest of adapting such methods to 3D geometric data. Compared to 2D images, point clouds are sparse, unordered and locality-sensitive, making it non-trivial to adapt CNNs to point cloud processing. Early attempts focus on regular representations of the data in the form of 3D voxels Wu et al. (2015b); Qi et al. (2016). These methods quantize point clouds into regular voxels in 3D space with a predefined resolution, and then apply volumetric convolution. More recently, some works explore new designs of local aggregation operators on point clouds to process point sets efficiently and reduce the loss of details Qi et al. (2017a,b); Wang et al. (2019b). PointNet Qi et al. (2017a) is a pioneering deep network architecture that directly processes point clouds for classification and semantic segmentation; it proposes a shared multi-layer perception (MLP), followed by a max-pooling layer, to approximate continuous set functions to deal with unordered point sets. PointNet++ Qi et al. (2017b) further proposes a hierarchical aggregation of point features to extract global features. In later works, DGCNN Wang et al. (2019b) proposes an effective EdgeConv operator that encodes the point relationships as edge features to better capture local geometric features of point clouds while still maintaining permutation invariance. In this chapter, we leverage a simplified PointNet architecture to extract local features from a point cloud before feeding it to an attention-based LSTM for point cloud sampling.

Point Cloud Sampling

Random sampling (RS) selects a set of points randomly from a point cloud and has the smallest computation overhead. But this method is sensitive to density imbalance issue Lang et al. (2020). Farthest point sampling (FPS) Eldar et al. (1997); Moenning & Dodgson (2003) has been widely used as a pooling operator in point cloud processing. It starts from a randomly selected point in the set and iteratively selects the next point from the point cloud that is the farthest from the selected points, such that the sampled points can achieve a maximal coverage of the input point cloud. Recently, S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020) have demonstrated that better sampling strategies can be learnt by a sampling network. These methods aim to generate a small set of samples that optimize the performance of a downstream task. Moreover, the generated 3D coordinates can be pushed towards a subset of original points to minimize the training loss if a matched point set is desired. Both methods treat the sampling process as a generation task and produce all the points in one shot, which does not pay sufficient attention to sample dependency, and leads to suboptimal performances. Our APSNet is a combination of FPS and task-oriented sampling in the sense that points are sampled sequentially with a task-oriented objective.

Knowledge Distillation

As one of the popular model compression techniques, knowledge distillation Hinton et al. (2014) is inspired by knowledge transfer from teachers to students. Its key strategy is to orientate compact student models to approximate over-parameterized teacher models such that student models can achieve the performances that are close to (sometimes even higher than) those of teachers'. Different from traditional knowledge distillation, which forces student networks to approximate the soft prediction of pre-trained teacher networks, self distillation Zhang et al. (2019) distills knowledge within a network itself from its own soft predictions. Our APSNet can be trained both in supervised learning and self-supervised learning via knowledge distillation. In the former case, labeled training data are required to train APSNet, while in the latter case the soft predictions of task network can be used to train APSNet such that the sampled point clouds from APSNet can achieve similar predictions as the original point clouds.

3.3 The Proposed Method

The overview of our proposed APSNet is depicted in Fig. 3.1, which contains two main components: (a) A simplified PointNet for feature extraction, (b) An LSTM with attention mechanism for sequential point sampling. In this section, we first describe the details of these components and then discuss different approaches to train APSNet.

Notation and Problem Statement

Let $\boldsymbol{P} = \{\boldsymbol{p}_i \in \mathbb{R}^3\}_{i=1}^n$ denote a point cloud that contains n points, with $\boldsymbol{p}_i = [x_i, y_i, z_i]$ representing the 3D coordinates of point i. We consider two types of point cloud samples: (1) $\boldsymbol{Q}^* = \{\boldsymbol{q}_i^* \in \mathbb{R}^3\}_{i=1}^m$ denotes a ***sampled*** point cloud of m points that is a subset of $\boldsymbol{P}$ with $m < n$, i.e., $\boldsymbol{Q}^* \subset \boldsymbol{P}$. (2) $\boldsymbol{Q} = \{\boldsymbol{q}_i \in \mathbb{R}^3\}_{i=1}^m$ denotes a ***generated*** point cloud of m points that *may not* be a subset of $\boldsymbol{P}$. Typically, we can convert $\boldsymbol{Q}$ to $\boldsymbol{Q}^*$ by a *matching* process, i.e., match each point in $\boldsymbol{Q}$ to its nearest point in $\boldsymbol{P}$, and then replace the duplicated points[1] in resulting $\boldsymbol{Q}^*$ by FPS. Without loss of generality, in the following we present our algorithm in terms of $\boldsymbol{Q}$ since $\boldsymbol{Q}$ is more general than $\boldsymbol{Q}^*$ and can be converted to $\boldsymbol{Q}^*$ by matching. Moreover, let $f_{\boldsymbol{\theta}}(\cdot) : \boldsymbol{P} \rightarrow \boldsymbol{Q}$ denote APSNet with the parameters $\boldsymbol{\theta}$.

As discussed in the introduction, we are interested in task-oriented sampling that yields a small set of points $\boldsymbol{Q}$ to optimize a downstream task represented by a well-trained deployed task network T, where T can be a model for 3D point cloud classification, reconstruction or registration, etc. Given $\boldsymbol{P}$, the goal of APSNet is to generate a point cloud $\boldsymbol{Q} = f_{\boldsymbol{\theta}}(\boldsymbol{P})$ that maximizes the predictive performance of task network T. Specifically, the parameters of APSNet, $\boldsymbol{\theta}$, is optimized by minimizing a task loss and a sampling loss jointly as

$$\min_{\boldsymbol{\theta}} \ell_{task}(T(\boldsymbol{Q}), y) + \lambda L_{sample}(\boldsymbol{Q}, \boldsymbol{P}), \tag{3.1}$$

whose details are to be discussed in Sec. 4.3.4.

3.3.1 Attention-based Point Cloud Sampling

Existing task-oriented sampling methods, such as S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020), formulate the sampling process as a point cloud generation problem from a global feature vector, and generate all m points in one shot. We argue that the sampling process is naturally a sequential generation process, in which points to be sampled next should depend on the points that have already been sampled. We therefore propose APSNet, an attention-based LSTM for sequential point sampling in order to capture the

[1]Multiple points in $\boldsymbol{Q}$ can be mapped to the same point in $\boldsymbol{P}$.

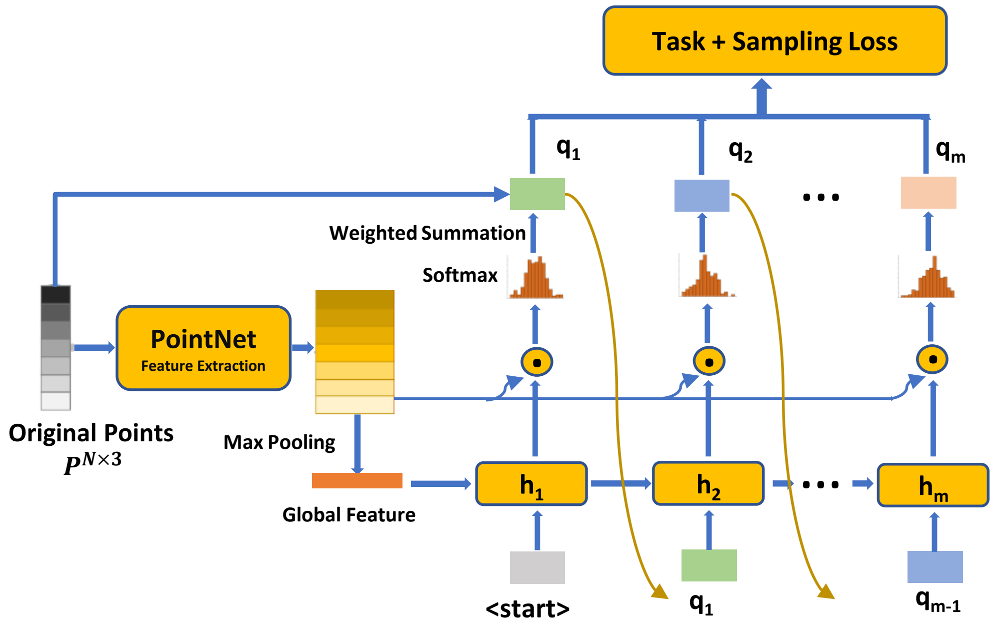

Figure 3.2 APSNet considers point cloud sampling as a sequential generation process with a task-oriented objective, and uses an LSTM with attention mechanism for sampling.

relationship among points.

The overall architecture of APSNet is depicted in Fig. 3.2. First, APSNet takes the original point cloud $\boldsymbol{P}$ as input and samples from $\boldsymbol{P}$ via an LSTM with attention mechanism to produce a small point cloud $\boldsymbol{Q}$ of m points, each point of which is a soft point generated by projecting $\boldsymbol{P}$ on a set of attention coefficients from the LSTM. Finally, the output of APSNet, $\boldsymbol{Q}$, is fed to a well-trained deployed task network T for prediction and task loss evaluation[2].

The first step is to extract a feature representation for each point in $\boldsymbol{P}$. APSNet follows the architecture of PointNet Qi et al. (2017a), a basic feature extraction backbone on 3D point clouds, to extract point-wise local features. Specifically, a set of 1×1 convolution layers are applied to the original point cloud $\boldsymbol{P}$ and produce a set of d-dimensional point-wise feature

[2]The parameters of T is frozen during the training of APSNet.

vectors, denoted by $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]^\mathsf{T} \in \mathbb{R}^{n \times d}$. Then, a symmetric feature-wise max pooling operator is applied to $\boldsymbol{X}$ and produce a global feature vector $\boldsymbol{g} \in \mathbb{R}^d$, which is then fed to an LSTM as the initial state for sequential point generation.

The sequential point generation process is similar to the attention-based sequential model for image captioning Xu et al. (2015a). Given initial state $\boldsymbol{g}$ and `<start>` as inputs, the LSTM updates its hidden state to $\boldsymbol{h}_t \in \mathbb{R}^d$ at each time step $t = 1, 2, \cdots, m$. The hidden state $\boldsymbol{h}_t$ encodes the history of all the sampled points and is indicative for APSNet to identify the next most informative point of $\boldsymbol{P}$ to sample. To achieve this, a set of attention scores is calculated as the dot product of the hidden state $\boldsymbol{h}_t$ and point-wise feature vector $\boldsymbol{x}_i$ for $i = 1, 2, \cdots, n$, followed by a softmax for normalization:

$$s_{it} = \boldsymbol{x}_i \cdot \boldsymbol{h}_t, \qquad a_{it} = \frac{\exp(s_{it})}{\sum_i \exp(s_{it})}. \tag{3.2}$$

The attention coefficients a_{it} indicates the importance of point i at the sampling step t, from which sampled point $\boldsymbol{q}_t \in \mathbb{R}^3$ can be generated as a weighted sum of all the points in $\boldsymbol{P}$:

$$\boldsymbol{q}_t = \sum_i a_{it} \cdot \boldsymbol{p}_i. \tag{3.3}$$

The generated point $\boldsymbol{q}_t$ is then fed to the LSTM as input for the next time step to generate the next point until all m points are generated. During sequential generation process, the attention mechanism enables the model to attend to all the points in $\boldsymbol{P}$ and identify the most informative "point" to sample.

3.3.2 Training of APSNet

APSNet is a task-oriented sampling network, which can be trained to optimize its performance on the downstream tasks of interest. In this section, we discuss the objective functions that can be used to train APSNet. Depending on the availability of labeled training data and deployment requirements, we consider three different approaches to train APSNet: (1) supervised training, (2) self-supervised training, and (3) joint training.

3.3.2.1 Training with or without Ground Truth Labels

We consider two training scenarios: (a) both task network T and a labeled training set $\{\boldsymbol{P}_j, y_j\}_{j=1}^N$ are available; (b) only task network T and some input point clouds $\{\boldsymbol{P}_j\}_{j=1}^N$ are available, but no labels is provided. The latter case corresponds to the situation where original labeled training data of T is no longer available for the development of APSNet.

Supervised Training

When a labeled training set $\{\boldsymbol{P}_j, y_j\}_{j=1}^N$ is available, we can train APSNet in a supervised learning paradigm. Similar to S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020), two types of losses are exploited to train APSNet, i.e., the task loss ℓ_{task} and the sampling loss L_{sample}. Specifically, the task loss measures the quality of predictions based on the sampled point cloud $\boldsymbol{Q}$:

$$\ell_{task}(T(\boldsymbol{Q}), y), \tag{3.4}$$

where y is the ground-truth label of $\boldsymbol{P}$. For different downstream tasks, y can be the class label or the original point cloud $\boldsymbol{P}$ when the task is for classification or reconstruction, respectively. Accordingly, the corresponding task loss $\ell_{task}()$ is defined differently, e.g., the cross-entropy loss for classification or the Chamfer distance for reconstruction Achlioptas et al. (2018).

The sampling loss L_{sample} encourages the sampled points in $\boldsymbol{Q}$ to be close to those of $\boldsymbol{P}$ and also have a maximal coverage w.r.t. the original point cloud $\boldsymbol{P}$. We found that this sampling loss provides an important prior knowledge for sampling, and is critical for APSNet to achieve a good performance. Specifically, given two point sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$, denoting average nearest neighbor loss as:

$$L_a(\boldsymbol{S}_1, \boldsymbol{S}_2) = \frac{1}{|\boldsymbol{S}_1|} \sum_{s_1 \in \boldsymbol{S}_1} \min_{s_2 \in \boldsymbol{S}_2} ||s_1 - s_2||_2^2, \tag{3.5}$$

and maximal nearest neighbor loss as:

$$L_m(\boldsymbol{S}_1, \boldsymbol{S}_2) = \max_{s_1 \in \boldsymbol{S}_1} \min_{s_2 \in \boldsymbol{S}_2} ||s_1 - s_2||_2^2, \tag{3.6}$$

the sampling loss is then given by:

$$L_{sample}(\boldsymbol{Q}, \boldsymbol{P}) = L_a(\boldsymbol{Q}, \boldsymbol{P}) + \beta L_m(\boldsymbol{Q}, \boldsymbol{P}) + (\gamma + \delta|\boldsymbol{Q}|) L_a(\boldsymbol{P}, \boldsymbol{Q}), \tag{3.7}$$

where β, γ and δ are the hyperparameters that balance the contributions from different loss components.

Putting all the components together, the total loss of APSNet is given by:

$$L_{total} = \ell_{task}(T(\boldsymbol{Q}), y) + \lambda L_{sample}(\boldsymbol{Q}, \boldsymbol{P}), \tag{3.8}$$

where λ is a hyperparameter that balances the contribution between the task loss and the sampling loss.

Self-supervised Training with Knowledge Distillation

In some practical scenarios, we may only have task network T and some input point clouds $\{\boldsymbol{P}_j\}_{j=1}^{N}$ at our disposal. This is the situation where a deployed task network T is available, but the original labeled training data of T is no longer available for the development of APSNet. In this case, we propose to train APSNet via self-supervised learning based on the idea of knowledge distillation Hinton et al. (2014); Zhang et al. (2019). In knowledge distillation, we can transfer the knowledge from a teacher network to a student network such that the student network can yield a similar prediction as the teacher network while being much more efficient. Inspired by knowledge distillation, we treat the task network T as the teacher model, and APSNet as the student model and use the soft predictions of T as the targets to train APSNet. Specifically, the task loss for self-supervised training of APSNet is updated to

$$\ell_{task}(T(\boldsymbol{Q}), \tilde{y}), \text{ with } \tilde{y} = T(\boldsymbol{P}), \tag{3.9}$$

where $\tilde{y}$ is the soft prediction of T given the original point cloud $\boldsymbol{P}$, and the loss function ℓ_{task} is defined differently for different downstream tasks. The goal of the new loss function is to generate a sampled point cloud $\boldsymbol{Q}$ that can yield a similar prediction as the original $\boldsymbol{P}$.

Similar idea is also explored in Chen et al. (2018), where mutual information between the predictions of backbone network from sparsified input and original input are maximized for model interpretation, while here we sparsify point clouds.

3.3.2.2 Joint Training

APSNet described above is trained for a specified sample size m. Given the autoregressive model of our method, APSNet can generate arbitrary length of samples from a single model. This entails an effective joint training of APSNet with multiple different sample sizes, resulting in a single compact model to generate arbitrary sized point clouds with prominent performance. Specifically, we can train one APSNet with different sample sizes by:

$$L_{joint} = \sum_{c \in C_s} \Big(\ell_{task}(T(\boldsymbol{Q}_c), y) + \lambda L_{sample}(\boldsymbol{Q}_c, \boldsymbol{P}) \Big), \tag{3.10}$$

where C_s is a set of sample sizes of interest. In our experiments, we set $C_s = \{2^l\}_{l=3}^{7}$.

S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020) propose a progressive training to train a sampling network to generate different sized point clouds. However, their model sizes grow linearly as the sample size m increases. In contrast, due to the autoregressive model of APSNet, we can train one single compact model (with a fixed number of parameters) to generate arbitrary sized point clouds without incurring a linear increase of model parameters. This entails a better parameter reusing or sharing for APSNet, and leads to improved sample efficiency as compared to S-NET and SampleNet.

3.4 Experiments

We demonstrate the performance of APSNet on three different applications of 3D point clouds for classification, reconstruction, and registration. For the purpose of comparison, random sampling (RS), farthest point sampling (FPS) and SampleNet Lang et al. (2020) are used as baselines, where SampleNet is the state-of-the-art task-oriented sampling method. We consider two variants of APSNet: (1) *APSNet*, and (2) *APSNet-KD*, while the former refers to the supervised training of APSNet and the latter refers to the self-supervised training of APSNet with knowledge distillation. A trained APSNet generates point cloud $\boldsymbol{Q}$ that isn't a subset of original input point cloud $\boldsymbol{P}$, but the generated $\boldsymbol{Q}$ can be converted to $\boldsymbol{Q}^*$ by a matching process as discussed in Sec. 3.1. Therefore, we further distinguish them as *APSNet-G* and *APSNet-M*, respectively. SampleNet Lang et al. (2020) also generates point cloud $\boldsymbol{Q}$, which is converted to $\boldsymbol{Q}^*$ by the matching process. Similarly, we denote them as *SampleNet-G* and *SampleNet-M*, respectively. In our experiments, we compare the performances of all these variants. However, we would like to emphasize that the default SampleNet is *SampleNet-M*, while the default APSNet is *APSNet-G* since APSNet-G yields the best predictive performance without an expensive matching process as we will demonstrate in the experiments.

Since our APSNet is implemented in PyTorch, we convert the official TensorFlow implementation of SampleNet[3] to PyTorch for a fair comparison. We found that our PyTorch

[3]`https://github.com/itailang/SampleNet`

implementation achieves better performances than the official TensorFlow version in most of our experiments. Details of experimental settings and implementation are relegated to supplementary material. All our experiments are performed on Nvidia RTX GPUs. Our source code can be found at `https://github.com/Yangyeeee/APSNet`.

3.4.1 Experimental Settings

Task Networks

Similar to SampleNet Lang et al. (2020), we adopt PointNet for classification Qi et al. (2017a), Point Cloud Autoencoder (PCAE) for reconstruction Achlioptas et al. (2018), and PCRNet for registration Sarode et al. (2019). For the classification and reconstruction tasks, PointNet and PCAE are trained with the same settings as reported by their original papers. For the registration task, Sarode et al. (2019) trained the PCRNet with the Chamfer distance between template point cloud and registered point cloud; we follow SampleNet and add a regression loss (besides the Chamfer distance) to train the PCRNet. These pre-trained networks are treated as the task networks for their specific applications, whose parameters are fixed during the training of APSNet.

APSNet

The feature extract component of APSNet follows the design of PointNet Qi et al. (2017a). It contains a sequence of 1×1 convolution layers, followed by a symmetric global pooling layer to generate a global feature vector, which is then used as the initial state of LSTM for sampling. Each convolution layer includes a batch normalization layer Ioffe & Szegedy (2015a) and a ReLU activation function. A 2-layer LSTM Hochreiter & Schmidhuber (1997) with 128 recurrent units in each layer is used to generate samples autoregressively.

We consider two variants of APSNet: (1) *APSNet*, and (2) *APSNet-KD*, while the former refers to the supervised training of APSNet and the latter refers to the self-supervised training of APSNet with knowledge distillation. A trained APSNet generates point cloud $\boldsymbol{Q}$ that isn't a subset of original input point cloud $\boldsymbol{P}$, but the generated $\boldsymbol{Q}$ can be converted to $\boldsymbol{Q}^*$ by a matching process as discussed in Sec. 3.1. Therefore, we further distinguish them as *APSNet-G* and *APSNet-M*, respectively.

SampleNet Lang et al. (2020) also generates point cloud $\boldsymbol{Q}$, which is converted to $\boldsymbol{Q}^*$ by the matching process. Similarly, we denote them as *SampleNet-G* and *SampleNet-M*, respectively. In our experiments, we compare the performances of all these variants. However, we would like to emphasize that the default SampleNet is *SampleNet-M*, while the default APSNet is *APSNet-G* since APSNet-G yields the best predictive performance without an expensive matching process as we will demonstrate in the experiments.

Implementation

We tune the performance of APSNet based on the hyperparameters provided by SampleNet Lang et al. (2020), and set $\beta = 1$, $\gamma = 1$ and $\delta = 0$. We use the Adam optimizer Kingma & Ba (2014) with the batch size of 128 for all the experiments. Learning rate is set to (0.01, 0.001, 0.0005), and λ of the total loss (9) is set (30, 0.01, 0.01) for classification, registration, and reconstruction tasks, respectively. Each experiment is trained for 400 epochs with a learning rate decay of 0.7 at every 20 epochs.

Since our code is PyTorch-based, we convert the official TensorFlow code of SampleNet[4] to PyTorch for a fair comparison. We found that our PyTorch implementation achieves better performances than the official TensorFlow version in most of our experiments. For reproducibility, our source code is also provided as a part of the supplementary material. All our experiments are performed on Nvidia RTX GPUs.

3.4.2 3D Point Cloud Classification

We use the point clouds of 1024 points that were uniformly sampled from the ModelNet40 dataset Wu et al. (2015b) to train PointNet Qi et al. (2017a) (the task network T). The official train-test split is used for the training and evaluation, and the instance-wise accuracy is used as the evaluation metric for performance comparison. The vanilla task network achieves an accuracy of 90.1% with all the 1024 points. We execute different sampling methods with a variety of sample sizes and report their performances for comparison.

Table 3.1 Classification accuracies of five sampling methods with different sample sizes m on ModelNet40. M* denotes the official results from SampleNet Lang et al. (2020).

	RS	FPS	DaNet	MOPS-Net		SampleNet			APSNet		APSNet-KD	
m				G	M	G	M	M*	G	M	G	M
8	8.26	23.29	-	-	-	78.36	73.31	28.7	**81.42**	74.12	80.22	73.81
16	25.11	54.19	-	**84.7**	51.2	80.60	79.68	55.5	83.89	82.25	83.82	82.02
32	55.19	77.32	85.1	86.1	77.6	80.32	82.97	74.4	88.15	86.97	**88.76**	84.95
64	78.26	87.22	86.8	87.1	81.0	79.36	84.01	79.0	88.38	87.58	**88.66**	87.54
128	85.95	88.76	86.8	87.2	85.0	85.52	87.17	79.7	89.22	**89.38**	87.83	88.01
256	88.80	89.30	87.2	87.4	86.7	87.43	89.58	83.4	89.54	**89.86**	88.02	88.21
512	89.66	89.87	-	88.3	88.3	88.01	90.18	88.2	89.78	**90.18**	88.69	88.56

Table 4.1 reports the classification accuracies of all the five sampling methods. To validate our PyTorch implementation of SampleNet, we also include the official SampleNet results

[4]`https://github.com/itailang/SampleNet`

as reported in Lang et al. (2020). It can be observed that our PyTorch implementation outperforms the official TensorFlow code consistently; in particular, when sample size $m = 8$, our implementation has a gain of nearly 45% over the official code. Therefore, for a fair comparison, we compare APSNet mainly with our improved SampleNet.

A few notable observations can be made from Table 4.1. (1) As sample size m increases, all the sampling methods have improved accuracies. The performances of task-oriented samplers, e.g., SampleNet and APSNet, outperform those of task-agnostic samplers, e.g., random sampling and FPS, consistently. However, the gains are getting smaller as sample size m increases; when $m = 512$, all sampling methods achieves a comparable accuracy that is close to the best accuracy (90.1%) achieved with all the 1024 points. (2) In general, SampleNet-M achieves a better performance than SampleNet-G. When sample size m increases, the gain is more pronounced. (3) In contrast, APSNet-G achieves a better performance than APSNet-M. When sample size m is small, the gain is large, while as sample size m increases, both variants of APSNet have very similar performances. This is likely because the downstream task networks are trained with original points P, and the matched Q^* from APSNet-M can fit better to the downstream task networks. The gains are getting smaller because when sampling ratio becomes larger the performance is approaching to the upper bound which uses all the points. (4) Comparing APSNet-G with SampleNet-M (the best defaults for both algorithms), APSNet outperforms SampleNet consistently; especially when $m \leq 128$, we observe a 2% to 8% accuracy gain, demonstrating the effectiveness of APSNet. (5) APSNet-KD achieves a very impressive result without utilizing labeled point clouds for training; its performance is almost on-par with APSNet that is trained with labeled point clouds.

Discussion

The above experiments show that SampleNet-M outperforms SampleNet-G consistently, while the opposite is observed for APSNet. This can be explained by the limitations of SampleNet as we indicated in the introduction. As sample size m increases, SampleNet has a higher probability of generating similar (redundant) points due to the one-shot generation of m samples. Since these redundant points cannot improve the classification accuracy effectively, the matching process (which replaces the redundant points with the FPS samples) becomes critical for SampleNet-M to improve its performance, leading to improved performances over SampleNet-G. On the other hand, APSNet-G generates the next sample depending heavily on previously sampled points, and therefore is able to capture the relationship among points and generate more informative samples, yielding a better performance without an expensive matching process.

Joint Training

Next, we investigate the joint training of APSNet, and compare it with separated training of ASPNet and SampleNet for each sample size m. For the joint training of APSNet, we train a single compact model of APSNet with $C_s = \{8, 16, 32, 64, 128\}$ by optimizing the joint loss (3.10). In contrast, in separated training of SampleNet or APSNet, a separated model is trained for each sample size $m \in C_s$ and its performance is evaluated for the specific m it was trained with.

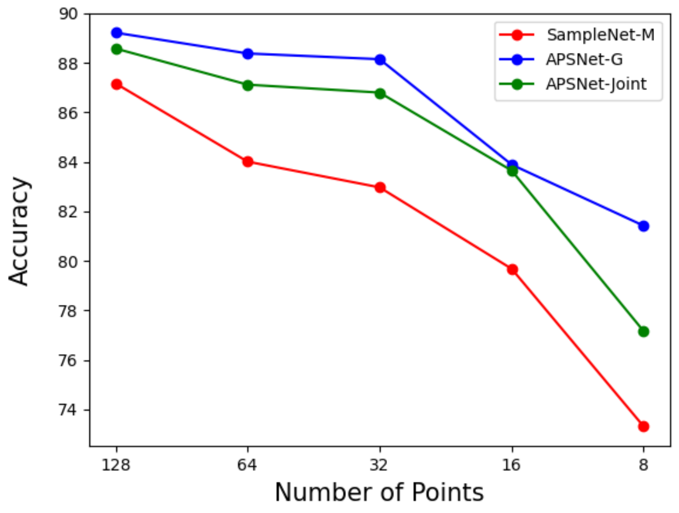

Figure 3.3 Evolution of classification accuracy as a function of sample size m for different sampling methods. APSNet-Joint uses a single model to generate different number of samples, while SampleNet and APSNet use separately trained models to generate each specific number of samples.

Fig. 3.3 shows the performance comparison between joint training of APSNet and separated training. It can be observed a single model trained by APSNet-Joint can generate arbitrary length of samples with competitive performances. Indeed, the performance of APSNet-Joint is lower than the separately trained APSNet, but it still consistently outperforms separately trained SampleNet.

3.4.3 Reconstruction

The reconstruction task is evaluated with point clouds of 2048 points, sampled from the ShapeNet Core55 dataset Chang et al. (2015). We choose the four shape classes that have the largest number of examples: Table, Car, Chair, and Airplane. Each class is split to a 85%, 5%, 10% partition for training, validation and test. The task network, in this case,

Table 3.2 The normalized reconstruction errors of five sampling methods with different sample sizes m on the ShapeNet Core55 dataset. M* denotes the original results from the SampleNet paper Lang et al. (2020). The lower, the better.

	RS	FPS	SampleNet			APSNet		APSNet-KD	
m			G	M	M*	G	M	G	M
8	21.85	12.79	5.29	5.48	-	**4.27**	4.59	4.69	4.98
16	13.47	7.25	2.78	2.89	-	**2.51**	2.62	2.57	2.67
32	8.16	3.84	1.68	1.71	2.32	1.54	1.59	**1.47**	1.52
64	4.54	2.23	1.32	1.27	1.33	**1.07**	1.11	1.12	1.14

is the Point Cloud Autoencoder (PCAE) for reconstruction Achlioptas et al. (2018). We evaluate the reconstruction performance with the normalized reconstruction error (NRE):

$$\mathsf{NRE}_{\mathsf{CD}}(\boldsymbol{Q}, \boldsymbol{P}) = \frac{\mathsf{CD}(\boldsymbol{P}, T(\boldsymbol{Q}))}{\mathsf{CD}(\boldsymbol{P}, T(\boldsymbol{P}))}, \tag{3.11}$$

where CD is the Chamfer distance Achlioptas et al. (2018) between two point clouds. Apparently, the values of $\mathsf{NRE}_{\mathsf{CD}}$ are lower bounded by 1, and the smaller, the better.

Table 4.4 reports the reconstruction results of all the five sampling methods considered. Similar to the results of classification, (1) SampleNet and APSNet outperform RS and FPS by a large margin. (2) SampleNet-M relies on the matching process to replace the redundant samples by FPS to improve its performance over SampleNet-G. (3) In contrast, APSNet-G outperforms APSNet-M consistently without the extra matching process. (4) APSNet-KD again achieves a very competitive result to APSNet. (5) Comparing APSNet-G and SampleNet-M (the best defaults for both algorithms), APSNet outperforms SampleNet consistently by a notable margin.

To investigate why APSNet outperforms SampleNet in the task of reconstruction, we visualize the sampled points and the reconstructed point clouds of both algorithms in Fig. 4.4. As can be seen, SampleNet focuses more on the main body of airplane and samples some

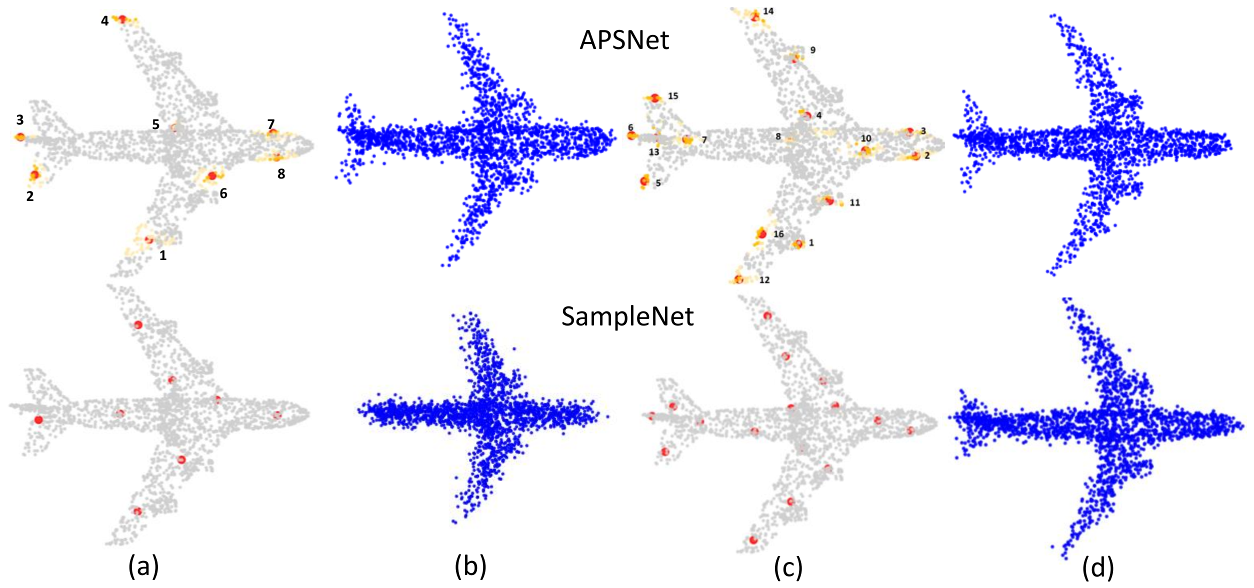

Figure 3.4 Visualization of sampled points and reconstructed point clouds by APSNet (1st row) and SampleNet (2nd row). The red dots are the sampled points; the highlighted yellow regions in APSNet results are points with high attention scores and the number specify the order of sampled points. (a) Sampled points when $m = 8$; (b) Reconstruction when $m = 8$, NRE(APSNet)=2.55, NRE(SampleNet)=5.20; (c) Sampled points when $m = 16$; (d) Reconstruction when $m = 16$, NRE(APSNet)=1.57, NRE(SampleNet)=2.34.

uninformative and symmetric points for reconstruction. In contrast, APSNet focuses more on the outline of the airplane without losing details, which are critical for the reconstruction. This observation is more pronounced when sample size is small, such as $m = 8$. As shown in Fig. 4.4(a) and (b), SampleNet fails to sample a point at the tail of the airplane such that the reconstructed point cloud cannot recover the tail. In comparison, APSNet samples two important points at the tail and ignores the symmetric one on the other side of the tail, and therefore is able to reconstruct the tail precisely. One the other hand, SampleNet samples two symmetric points on the wing, which are likely redundant information for the reconstruction. Overall, the sampled points from APSNet are more reasonable than those of SampleNet from human's perspective, The effectiveness of different loss components

The sampling loss (7) encourages the sampled points in $\boldsymbol{Q}$ to be close to those of $\boldsymbol{P}$ and also have a maximal coverage w.r.t. the original point cloud $\boldsymbol{P}$. We found that this sampling loss provides an important prior knowledge for sampling, and is critical for APSNet to achieve a good performance. In addition, the sampling loss (7) is more generic than the Chamfer distance since when $\beta = 0$, $\gamma = 1$ and $\delta = 0$ it degenerates to the Chamfer distance. The limitation is that we now have more hyperparameters to tune. Table 3.3 reports the ablation study of the sampling loss (7) for APSNet-G on the reconstruction task. It shows that when $L_m(\boldsymbol{Q}, \boldsymbol{P})$ and $L_a(\boldsymbol{P}, \boldsymbol{Q})$ enabled (i.e., $\beta = 1$ and $\gamma = 1$), APSNet-G reaches the best results in almost all settings.

Table 3.3 Ablation study of the sampling loss (7) for APSNet-G on the reconstruction task. * denotes the best results when $\beta = 1$, $\gamma = 1$ and $\delta = 0$.

	$\beta = 0$	$\gamma = 0$	*
8	4.63	4.54	4.27
16	3.07	3.44	2.51
32	1.67	1.49	1.54
64	1.13	1.28	1.07

Inference time comparison

We further evaluate the inference times of different sampling methods in the task of reconstruction. The results are reported in Table 4.3, where SampleNet-M and APSNet-G are the main algorithms to be compared since they are the best defaults. It can be observed that when sample size m increases, the inference times of both SampleNet-M and APSNet-G increase, while SampleNet-G requires roughly a constant time for sampling. This is because SampleNet-G leverages an MLP generator to generate all m samples in one shot; for the problem size considered, one GPU is able to utilize its on-board parallel resources to process different sample sizes in roughly the same time. However, as observed in the experiments above and also proposed by SampleNet Lang et al. (2020), SampletNet relies on the matching process to improve its performance, while matching is the most expensive operation in SampleNet, leading to a dramatic increase of inference-time for SampleNet-M. By contrast, due to the autoregressive model of our method, APSNet generates samples sequentially by an LSTM which results in a linear increase of inference time as m increases. However, APSNet does not need an expensive matching process for its best performance. Therefore, besides the improved sample quality, APSNet also outperforms SampleNet in terms of inference time.

Table 3.4 Inference time comparison of three sampling methods with different sample size m. The time is reported in millisecond. * denotes the best default recommended by each paper.

m	32	128	256	512
SampleNet-G	7.63	7.54	7.79	7.94
SampleNet-M*	44.33	135.23	261.47	515.30
APSNet-G*	9.21	12.84	17.68	27.48
APSNet-M	45.91	139.83	269.40	525.38

3.4.4 Registration

The task of registration aims to align two point clouds by predicting rigid transformations (e.g., rotation and translation) between them. To save memory and computation power, the registration is conducted on the key points that are sampled from the original point clouds. We follow the work of PCRNet Sarode et al. (2019) to construct a point cloud registration network (the task network), and train PCRNet on the point clouds of 1,024 points of the Car category from ModelNet40. Following the settings in SampleNet, 4,925 pairs of source and template point clouds are generated for training, where a template is rotated by three random Euler angles in the range of $[-45^\circ, 45^\circ]$ to obtain the source. An additional 100 source-template pairs are generated from the test split for evaluation. The mean rotation error (MRE) between the predicted rotations and ground-truth rotations is used as the evaluation metric.

Table 3.5 reports the performances of five sampling methods for registration. Similar to the results on classification and reconstruction, APSNet outperforms SampleNet consistently by a notable margin, and achieves the state-of-the-art results in this task. Without leveraging labeled training data, APSNet-KD again demonstrates an impressive performance that is close to supervised APSNet.

Table 3.5 The mean rotation errors of five sampling methods with different sample sizes m on the ModelNet40 dataset for registration. M* denotes the original results from the SampleNet paper Lang et al. (2020). The lower, the better.

	RS	FPS	SampleNet			APSNet		APSNet-KD	
m			G	M	M*	G	M	G	M
8	63.37	31.44	9.72	8.27	10.51	**5.47**	9.40	5.93	10.51
16	43.89	20.34	12.14	7.45	8.21	**4.50**	7.18	5.01	7.07
32	27.06	12.97	10.81	6.13	5.94	**4.37**	5.82	4.56	6.07
64	16.88	7.89	10.93	5.38	5.31	**4.42**	6.34	4.49	4.97

Visualization of Attention Coefficients

For the task of registration, we further visualize the evolution of attention coefficients during the training process. Specifically, we monitor the attention coefficients Eq. (3) when generating a point at a specific time step t (the t-th sample) from a given point cloud of 1024 points. Figure 3.5 visualizes the evolution of attention coefficients over 400 training epochs. At beginning of the training, the sampler cannot decide which point from the point cloud is the most important one to sample, manifested by the dense cluttered coefficients. As the

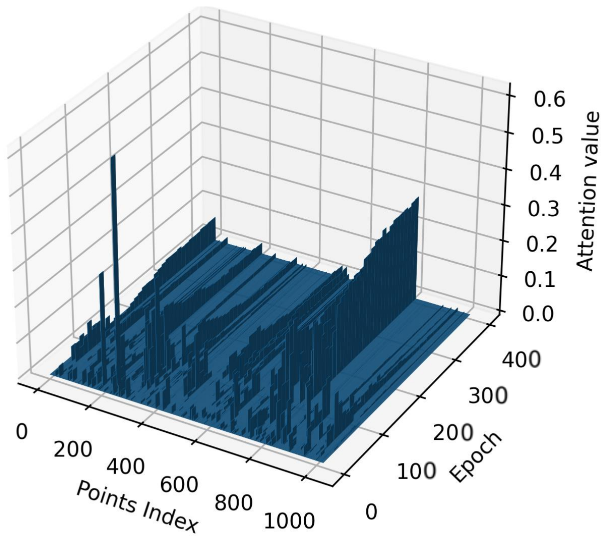

Figure 3.5 Evolution of the attention coefficients of APSNet when generating the t-th sample. As the training proceeds, the coefficients become sparser with peak values on a few points.

training proceeds, the attention coefficients become sparser with peak values on 2-3 points. Further, these attention coefficients are stablized in the late training epochs and consistently concentrate on a few the same points, demonstrating the training stability of APSNet.

3.5 Conclusion

this chapter introduces APSNet, an attention-based sampling network for point cloud sampling. Compared to S-Net and SampleNet, which formulate the sampling process as an one-shot generation task with MLPs, APSNet employs a sequential autoregressive generation with a novel LSTM-based sequential model for sampling. Depending on the availability of labeled training data, APSNet can be trained in supervised learning or self-supervised learning via knowledge distillation. We also present a joint training of APSNet, yielding a single compact model that can generate arbitrary length of samples with prominent performances. Extensive experiments demonstrate the superior performance of APSNet over the state-of-the-arts both in terms of sample quality and inference speed, which make APSNet widely applicable in many practical application scenarios.

CHAPTER 4

PTSNet: A Point Transformer for Task-oriented Point Cloud Sampling

4.1 Introduction

The development of 3D sensing devices, such as LiDAR and RGB-D cameras, has led to the generation of large amounts of point cloud data in fields such as robotics, autonomous driving, and virtual reality Nüchter & Hertzberg (2008); Geiger et al. (2012); Park et al. (2008). Point clouds, which are composed of raw coordinates in 3D space, provide accurate representations of objects and shapes, and are crucial to perceive the surrounding environment. However, traditional feature extraction methods like convolutional neural networks (CNNs) designed for 2D grid-structured data are not well-suited for point clouds, which are irregularly structured with variable densities. Some methods attempt to first stiffly transform point clouds into grid-structured data and then take advantage of CNNs for feature extraction, such as projection-based methods Simony et al. (2018); Beltrán et al. (2018) and volumetric convolution-based methods Engelcke et al. (2017); Li (2017). However, placing a point cloud on a regular grid generates an uneven number of points in grid cells. Thus, applying the same convolution operation on such grid cells leads to information loss in crowded cells and wasting computation in empty cells. Recently, many methods of directly processing point clouds Qi et al. (2017b); Yu et al. (2018); Li et al. (2018d); Qi et al. (2019) have been proposed, which enable efficient computation and improved performances in myriad applications, including 3D point cloud classification Qi et al. (2017b); Li et al. (2018d); Thomas et al. (2019); Wu et al. (2019b), semantic segmentation Li et al. (2018b); Su et al. (2018); Liu

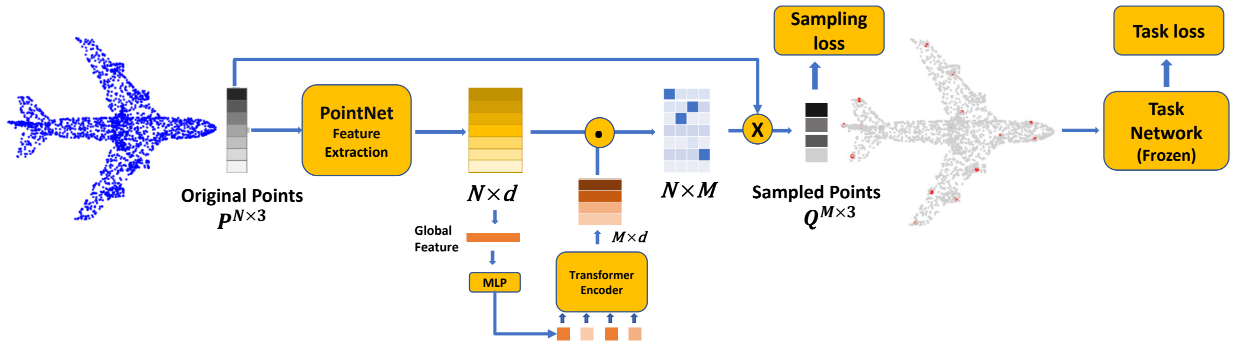

Figure 4.1 Overview of PTSNet. PTSNet first extracts features with a simplified PointNet architecture that preserves the geometric information of a point cloud. Then, a transformer encoder with self-attention mechanism is used to capture the relationship among the dynamic queries and select the most informative points. Finally, the sampled point cloud is fed to a task network for prediction. The whole pipeline is optimized by minimizing a task loss and a sampling loss jointly.

et al. (2019); Wang et al. (2019b, 2018) and reconstruction Achlioptas et al. (2018); Yang et al. (2018); Han et al. (2019); Zhao et al. (2019). However, dealing with large-scale 3D point clouds remains a challenge due to the high cost involved in storing, transmitting, and processing such data.

Point cloud sampling, a task of selecting a subset of points to represent the original point clouds at a sparse scale, can reduce data redundancy and improve the efficiency of 3D data processing. So far, there are a few heuristics-based sampling methods, such as random sampling (RS), farthest point sampling (FPS) Eldar et al. (1997); Moenning & Dodgson (2003), and grid (voxel) sampling Wu et al. (2015b); Qi et al. (2016). However, all of these methods are task-agnostic as they do not take into account the subsequent processing of the sampled points and may select non-informative points to the downstream tasks, leading to suboptimal performance. Recent works have concentrated on designing a downsampling block that subjects to a subsequent pretrained task network. This idea of task-oriented point cloud sampling was first developed by S-NET Dovrat et al. (2019b), which generates a small number of samples that optimize the performance of a downstream task and outperforms traditional task-agnostic samplers significantly in various applications. Following this work, SampleNet Lang et al. (2020) and MOPS-Net Qian et al. (2020) utilize a soft projection strategy to promote the learned points to be the proper subset of the original point cloud for performance improvement. Both methods formulate the sampling process as a one-shot generation task with MLPs, and do not pay enough attention to the dependency among samples. Ye et al. (2022) argue that point cloud sampling can be considered as a sequential generation process, in which points to be sampled next should depend on the points that have already been sampled. Based on this idea, they propose APSNet, an attention-based sampling network, to capture the dependence among the selected points. Although APSNet has achieved state-of-the-art performance compared with other methods, the two-layer LSTM employed by APSNet suffers from the vanishing gradient problem, manifested by many duplicate samples, especially when dealing with long sequences.

Inspired by the success of transformer in natural language processing Vaswani et al. (2017); Wu et al. (2019a); Dai et al. (2019); Devlin et al. (2019); Yang et al. (2019b) and image analysis Dosovitskiy et al. (2021); Hu et al. (2019); Zhao et al. (2020), recent works **?**Zhao et al. (2021a) have introduced transformer-based models to handle irregular and unordered point cloud data, demonstrating their superior capabilities. The permutation-invariant property of self-attention mechanism of transformer makes it well-suited to process point cloud data. Therefore, this chapter introduces PTSNet, a novel point transformer for task-oriented point cloud sampling. PTSNet leverages a simplified PointNet to extract point-wise feature vectors, followed by an MLP to generate initial selection queries, which are then optimized by a transformer encoder to capture the most valuable information from the point cloud. Extensive experiments demonstrate the superior performance of PTSNet compared to prior methods across various downstream tasks, including 3D point cloud classification, reconstruction, and registration.

4.2 Related Work

Deep Learning on Point Clouds

Following the breakthrough results of CNNs in 2D image processing tasks Krizhevsky et al. (2012); He et al. (2016a), there has been a strong interest of adapting such methods to 3D geometric data. Compared to 2D images, point clouds are sparse, unordered and locality-sensitive, making it non-trivial to adapt CNNs for point cloud processing. Early attempts focus on regular representations of the data in the form of 3D voxels Wu et al. (2015b); Qi et al. (2016). These methods quantize point clouds into regular voxels in 3D space with a predefined resolution and then apply volumetric convolution. More recently, some works explore new designs of local aggregation operators on point clouds to process point sets efficiently and reduce the loss of details Qi et al. (2017a,b); Wang et al. (2019b). PointNet Qi et al. (2017a) is a pioneering deep network architecture that directly processes point clouds for classification and semantic segmentation; it proposes a shared multi-layer perception (MLP), followed by a max-pooling layer, to approximate continuous set functions to deal with unordered point sets. PointNet++ Qi et al. (2017b) further proposes a hierarchical aggregation of point features to extract global features. In later works, DGCNN Wang et al. (2019b) proposes an effective EdgeConv operator that encodes the point relationships as edge features to better capture local geometric features of point clouds while still maintaining permutation invariance. In this chapter, we leverage a simplified PointNet architecture to extract local features from a point cloud before feeding them to a transformer encoder for point cloud sampling.

Point Cloud Sampling

Random sampling (RS) selects a set of points randomly from a point cloud and has the smallest computation overhead. However, this method is sensitive to density imbalance issue Lang et al. (2020). Farthest point sampling (FPS) Eldar et al. (1997); Moenning & Dodgson (2003) has been widely used as a pooling operator in point cloud processing. It starts from a randomly selected point in the set and iteratively selects the next point from the point cloud that is the farthest from the selected points, such that the sampled points can achieve a maximal coverage of the original point cloud. Recently, S-NET Dovrat et al. (2019a) has demonstrated that better sampling strategies can be learnt by a sampling network that can generate a small set of samples to optimize the performance of a downstream task. Moreover, the generated 3D coordinates can be pushed towards a subset of original points if a matched point set is desired. Following this work, SampleNet Lang et al. (2020) and MOPS-Net Qian et al. (2020) utilize a soft projection to promote the learnt points to be a proper subset of the original point cloud. Both methods formulate the sampling process as an one-shot generation task with MLPs and do not pay enough attention to the sample dependency, leading to suboptimal performances. DA-Net Lin et al. (2021) combines the K-nearest neighbors (KNN) with local adjustment such that the sampled points have noise immunity characteristics. Our PTSNet retains the advantage of one-shot generation, while capturing long-range correlations of sampled points through the global attention of transformer.

Transformer and self-attention

The introduction of self-attention and transformer Vaswani et al. (2017) ignites a revolution in the field of natural language processing Vaswani et al. (2017); Wu et al. (2019a); Dai et al. (2019); Devlin et al. (2019); Yang et al. (2019b) and computer vision Dosovitskiy et al. (2021); Hu et al. (2019); Zhao et al. (2020). Transformer utilizes a self-attention mechanism to scan through each element of a sequence and extract features by aggregating information from the entire sequence. The global computation and dynamic memory retrieval of transformer make it a better architecture choice than recurrent neural networks (RNNs) to process long sequences. The breakthrough has also inspired the development of attention networks for 2D image analysis. Hu et al. (2019) and Ramachandran et al. (2019) apply scalar dot-product self-attention within local image patches. Zhao et al. (2020) develops a family of vector self-attention operators. Dosovitskiy et al. (2021) treats images as sequences of patches. The permutation-invariant property of self-attention also makes it well-suited to process the irregular and unordered point cloud data. **?** proposes a local-global transformer, named PCT, with 2-layer local neighbor embedding and 4 stacked offset-attention blocks.

Point Transformer Zhao et al. (2021a), based on vector self-attention, uses the subtraction relation and adds a positional encoding to both the attention vector and the transformed features. Recently, Wang et al. (2021) introduce a transformer-based downsampling network PST-NET that utilizes local-global context information for improved performance. Wang et al. (2022) design a lightweight transformer network named LighTN with favorable FLOPs and parameters budgets. Compared to these prior methods, our PTSNet employs a novel dynamic query generator to produce selection queries based on a global representation of point cloud. The selection queries can be well optimized to capture the long-range correlations among the selected points, leading to state-of-the-art performance.

4.3 The Proposed Method

Figure 5.2 provides an overview of our proposed PTSNet, comprising two primary components: (a) a simplified PointNet for feature extraction, and (b) a transformer encoder with a self-attention mechanism for dynamic query generation. In this section, we delve into the details of these components.

4.3.1 Notation and Problem Statement

Let $\boldsymbol{P} = \{\boldsymbol{p}_i \in \mathbb{R}^3\}_{i=1}^{n}$ denote a point cloud that contains n points, with $\boldsymbol{p}_i = [x_i, y_i, z_i]$ representing the 3D coordinates of point i. We consider two types of point cloud samples: (1) $\boldsymbol{Q}^* = \{\boldsymbol{q}_i^* \in \mathbb{R}^3\}_{i=1}^{m}$ denotes a ***sampled*** point cloud of m points that is a subset of $\boldsymbol{P}$ with $m < n$, i.e., $\boldsymbol{Q}^* \subset \boldsymbol{P}$. (2) $\boldsymbol{Q} = \{\boldsymbol{q}_i \in \mathbb{R}^3\}_{i=1}^{m}$ denotes a ***generated*** point cloud of m points that *may not* be a subset of $\boldsymbol{P}$. Typically, we can convert $\boldsymbol{Q}$ to $\boldsymbol{Q}^*$ by a *matching* process, i.e., match each point in $\boldsymbol{Q}$ to its nearest point in $\boldsymbol{P}$, and then replace the duplicated points[1] in resulting $\boldsymbol{Q}^*$ by FPS. Since the matching process is non-differentiable, $\boldsymbol{Q}$ is often trained end to end without evoking a matching process in the training stage, and the matching process is involved only when inference. Moreover, let $f_{\boldsymbol{\theta}}(\cdot) : \boldsymbol{P} \to \boldsymbol{Q}$ denote PTSNet with parameters $\boldsymbol{\theta}$.

Our goal is to optimize PTSNet to generate a point cloud $\boldsymbol{Q} = f_{\boldsymbol{\theta}}(\boldsymbol{P})$ that maximizes the predictive performance of a task network T, where T can be a model for 3D point cloud classification, reconstruction or registration, etc. The optimization involves minimizing a joint objective function comprising a task loss and a sampling loss:

$$\min_{\boldsymbol{\theta}} \ell_{task}(T(\boldsymbol{Q}), y) + \lambda L_{sample}(\boldsymbol{Q}, \boldsymbol{P}), \tag{4.1}$$

where ℓ_{task} measures the quality of predictions based on $\boldsymbol{Q}$, y is the ground-truth label, and L_{sample} encourages proper sampling of points in $\boldsymbol{Q}$.

4.3.2 Transformer-based Dynamic Query Generator

In PTSNet, a novel transformer-based dynamic query generator is utilized to generate selection queries to ensure that the selection matrix can retrieve the most informative points as output. The process begins by extracting feature representations for each point in point cloud $\boldsymbol{P}$. APSNet Ye et al. (2022) adopts a simplified PointNet Qi et al. (2017a) as the feature extraction backbone for 3D point clouds. Specifically, a set of 1×1 convolution layers are applied to the original point cloud $\boldsymbol{P}$ and produce a set of d-dimensional point-wise

[1]Multiple points in $\boldsymbol{Q}$ can be mapped to the same point in $\boldsymbol{P}$.

feature vectors, denoted by $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_N]^\top \in \mathbb{R}^{N \times d}$. Then, a symmetric feature-wise max pooling operator is applied to $\boldsymbol{F}$ and produces a global feature vector $\boldsymbol{g} \in \mathbb{R}^d$, which is then fed to an MLP and generates M initial dynamic selection queries. The selection queries are designed to capture the relationship among sampled points, and select the most valuable information from the point cloud. Formally, the M initial dynamic selection queries $\boldsymbol{I} \in \mathbb{R}^{M \times d}$ are learned through an MLP module, which projects the global feature $\boldsymbol{g}$ into a higher dimension of $M * d$ and reshapes it into a matrix with the shape of $M \times d$.

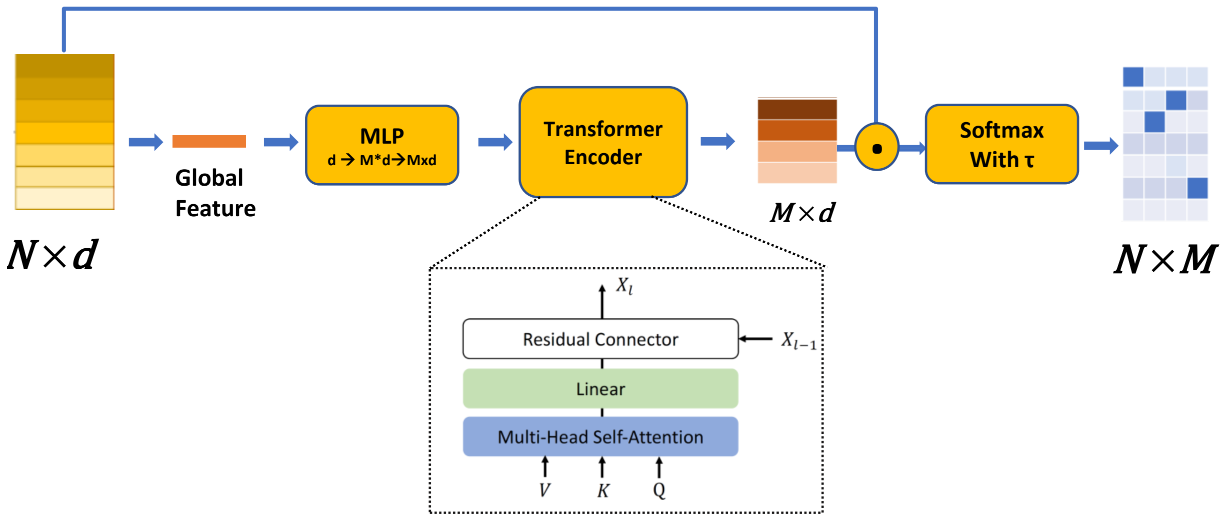

Figure 4.2 The dynamic query generator processes the point-wise feature matrix $\boldsymbol{F}$ to produce an $N \times M$ selection matrix. Initially, M dynamic selection queries are generated using an MLP module based on the global feature. These queries are then enriched through four stacked transformer layers, which aim to learn a semantically robust and distinctive representation for each query. The selection matrix is obtained by computing the dot product between the enriched queries and the point-wise features. To ensure the resulting selection vector approximates a one-hot vector, a softmax operation with a temperature parameter τ is applied.

Building upon the insight from APSNet Ye et al. (2022), transformer encoder layers are employed to optimize the dependency among the queries. Specifically, four stacked transformer layers are used to learn a semantically rich and discriminative representation for each query. Overall, the dynamic query generator shares almost the same philosophy of design as the original transformer, except that the positional embedding is discarded since each point's positional information has already been reserved in its point-wise feature. We refer the reader to Vaswani et al. (2017) for architectural details of the original transformer.

The d-dimensional query vectors $\mathbf{F}_i \in \mathbb{R}^{M\times d}$ yielded by transformer encoder are:

$$\mathbf{F}_1 = \mathrm{AT}^1(\mathbf{F}), \quad \mathbf{F}_i = \mathrm{AT}^i(\mathbf{F}_{i-1}), \quad i = 2, 3, 4, \tag{4.2}$$

where AT^i represents the i-th attention layer, each having the same output dimension as its input. We adopt multi-head self-attention (MHSA) as introduced in the original transformer Vaswani et al. (2017). The architecture of the SA layer is depicted in Figure 4.2. Following the terminology in Vaswani et al. (2017), let $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ be the *query*, *key* and *value* matrices, respectively, generated by linear transformations of the input features $\mathbf{F}_{in} \in \mathbb{R}^{M\times d}$ as follows:

$$(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{F}_{in} \cdot (\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v),$$
$$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}, \mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$$

where $\mathbf{W}_q$, $\mathbf{W}_k$ and $\mathbf{W}_v$ are the shared learnable linear transformation, and d is the dimension of query and key vectors. First, we can use the query and key matrices to calculate the attention weights via the matrix dot-product: $\tilde{\mathbf{A}} = (\tilde{\alpha})_{i,j} = \mathbf{Q} \cdot \mathbf{K}^{\mathrm{T}}$. These weights are then normalized to yield $\mathbf{A} = (\alpha)_{i,j}$:

$$\bar{\alpha}_{i,j} = \frac{\tilde{\alpha}_{i,j}}{\sqrt{d}}, \qquad \alpha_{i,j} = \mathrm{softmax}(\bar{\alpha}_{i,j}) = \frac{\exp\left(\bar{\alpha}_{i,j}\right)}{\sum\limits_{k} \exp\left(\bar{\alpha}_{i,k}\right)}.$$

The self-attention output features $\mathbf{F}_{sa}$ are the weighted sums of the value vector using the corresponding attention weights: $\mathbf{F}_{sa} = \mathbf{A} \cdot \mathbf{V}$. As the query, key and value matrices are determined by the shared corresponding linear transformation matrices and the input feature $\mathbf{F}_{in}$, they are all order independent. Moreover, softmax and weighted sum are both permutation-independent operators. Therefore, the whole self-attention process is permutation-invariant, making it well-suited to the unordered, irregular domain presented by point clouds. Finally, the self-attention feature $\mathbf{X}_{sa}$ and the input feature $\mathbf{F}_{in}$, are further used to provide the output feature $\mathbf{F}_{out}$ for the whole SA layer through an LBR (combining Linear, BatchNorm (BN) and ReLU layers) network:

$$\mathbf{F}_{out} = \mathrm{SA}(\mathbf{F}_{in}) = \mathrm{LBR}(\mathbf{F}_{sa}) + \mathbf{F}_{in}. \tag{4.3}$$

4.3.3 Differentiable sampling

Achieving accurate point cloud sampling involves training a separate neural network to generate a selection matrix, where each column/row represents a one-hot selection vector to guarantee that the sampled points are a proper subset of the original input. In practice, since the selection matrix involves discrete and combinatorial variables, the problem is often addressed by approximating it as a differentiable matrix optimization problem with relaxed binary constraints. An additional matching operation, such as nearest neighbor search, is often required to associate each generated point with its nearest neighbor in the original point cloud during the inference stage. Formally, given the optimized selection queries $\mathbf{F}$, a set of selection scores is calculated as the dot product with point-wise feature $\boldsymbol{x}$. Inspired by soft projection on weight coordinates Lang et al. (2020), differentiable relaxation to the matching phase can solve the above problem. Based on differentiable relaxation, we apply temperature annealing technique to encourage the PTSNet to produce a proper subset from the original point cloud. In this chapter, we utilize a similar soft projection operation Lang et al. (2020), denoting the average weight of original points of $\boldsymbol{P}$ as soft projected point q_i with $q_i = \sum\limits_{p_j \in \boldsymbol{P}} s_{ij} \cdot p_j$. Temperature annealing is used to optimize the constraints for selection matrix $\mathbf{S}$:

$$s_{ij} = \frac{\exp(dis_{ij}/\tau)}{\sum_{j=1}^{N} \exp(dis_{ij}/\tau)}, \tag{4.4}$$

where $dis_{ij} = \boldsymbol{f}_i \cdot \boldsymbol{x}_j$, τ is a temperature coefficient that controls the distribution shape of weight w_{ij}. It is clear that when $\tau \rightarrow 0^+$, point q_i is approximately considered to be the proper subset of input point cloud. During training, τ is annealed to a small value and fixed for inference.

4.3.4 Training Objectives

Similar to S-NET Dovrat et al. (2019a) and SampleNet Lang et al. (2020), PTSNet utilizes two types of losses during training: the task loss ℓ_{task} and the sampling loss L_{sample}. The task loss evaluates the quality of predictions based on the sampled point cloud $\boldsymbol{Q}$ denoted as $\ell_{task}(T(\boldsymbol{Q}), y)$, where y is the ground-truth label of $\boldsymbol{Q}$. Depending on the downstream task, y

can represent the class label or the original point cloud $\boldsymbol{P}$ for classification or reconstruction, respectively. Accordingly, the corresponding task loss $\ell_{task}()$ varies, such as using cross-entropy loss for classification or Chamfer distance for reconstruction. The sampling loss L_{sample} encourages the sampled points in $\boldsymbol{Q}$ to be close to those of $\boldsymbol{P}$ and also have a maximal coverage w.r.t. the original point cloud $\boldsymbol{P}$. This sampling loss provides crucial prior knowledge for effective sampling, playing a vital role in PTSNet's overall performance. Specifically, considering two point sets $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, we define the average nearest neighbor loss as:

$$L_a(\boldsymbol{P}_1, \boldsymbol{P}_2) = \frac{1}{|\boldsymbol{P}_1|} \sum_{p_1 \in \boldsymbol{p}_1} \min_{p_2 \in \boldsymbol{p}_2} ||p_1 - p_2||_2^2, \tag{4.5}$$

and maximal nearest neighbor loss as:

$$L_m(\boldsymbol{P}_1, \boldsymbol{P}_2) = \max_{p_1 \in \boldsymbol{P}_1} \min_{p_2 \in \boldsymbol{P}_2} ||p_1 - p_2||_2^2, \tag{4.6}$$

the sampling loss is then given by:

$$L_{sample}(\boldsymbol{Q}, \boldsymbol{P}) = L_a(\boldsymbol{Q}, \boldsymbol{P}) + \beta L_m(\boldsymbol{Q}, \boldsymbol{P}) \tag{4.7}$$

$$+ (\gamma + \delta|\boldsymbol{Q}|) L_a(\boldsymbol{P}, \boldsymbol{Q}), \tag{4.8}$$

where β, γ and δ are the hyperparameters that balance the contributions from different loss components. Putting all the components together, the total loss of PTSNet is given by:

$$L_{total} = \ell_{task}(T(\boldsymbol{Q}), y) + \lambda L_{sample}(\boldsymbol{Q}, \boldsymbol{P}), \tag{4.9}$$

where λ is a hyperparameter that balances the contribution between the task loss and the sampling loss.

4.4 Experiments

We assess the performance of PTSNet across three distinct applications of 3D point clouds: classification, reconstruction, and registration. Our comparative analysis involves several baseline methods, including random sampling (RS), farthest point sampling (FPS), SampleNet Lang et al. (2020), MOPS-Net Qian et al. (2020), DA-Net Lin et al. (2021), and APSNet Ye et al. (2022). Specifically, we focus on comparing PTSNet with APSNet to analyze potential performance improvements. For each application, we employ established networks: PointNet for classification Qi et al. (2017a), Point Cloud Autoencoder (PCAE) for reconstruction Achlioptas et al. (2018), and PCRNet for registration Sarode et al. (2019). PointNet and PCAE are trained with configurations consistent with their original papers. In the registration task, we augment the training of PCRNet with an additional regression loss, in addition to the Chamfer distance. To demonstrate the scalability of PTSNet to large datasets, we also utilize DGCNN Wang et al. (2019b) as the backbone network, and train the model on the ScanObjectNN dataset Uy et al. (2019), which is a challenging benchmark that facilitates a robust evaluation of PTSNet. For all the experiments, the pre-trained networks are treated as task networks for their specific applications, whose parameters are fixed during the training of PTSNet.

Implementation

The feature extraction component of PTSNet follows the architecture of PointNet Qi et al. (2017a), incorporating 1×1 convolution layers and a symmetric global pooling layer for global feature vector generation. The MLPs used for initial query generation are composed of three linear layers with batch normalization. The transformer encoder consists of four stacked layers, each with four heads. Hyperparameters are tuned, and specific values are set for $\beta = 1$, $\gamma = 1$, and $\delta = 0$. Adam optimizer Kingma & Ba (2014) is employed with a batch size of 32, and the learning rate is set to 0.0001. The coefficient λ in the total loss (Eq. 4.9) is adjusted to (30, 0.01, 0.01) for classification, registration, and reconstruction tasks, respectively. Each experiment undergoes 400 epochs, with a learning rate decay of 0.7 every 20 epochs. The temperature parameter τ is annealed from 1 to 0.01 starting at 200 epochs, and it remains fixed at 0.01 for inference. Exponential decay with an annealing rate of 0.00015 is applied every 50 steps. The hidden dimension d is set to 128, and a dropout rate of 0.5 is applied to the self-attention layers. All experiments are conducted on Nvidia RTX GPUs, ensuring a consistent hardware environment. Results are reported as averages over 10 independent runs to ensure the robustness and reliability of the findings. Our source code is provided as supplementary materials for reproducible research.

4.4.1 3D Point Cloud Classification

Table 4.1 Classification accuracies of different sampling methods with different sample sizes m on ModelNet40.

m	RS	FPS	SampleNet	MOPS-Net	DA-Net	APSNet	PTSNet (Ours)
8	8.26	23.29	73.31	32.62	-	74.12	**75.72**
16	25.11	54.19	79.68	64.83	51.2	82.25	**83.54**
32	55.19	77.32	82.97	79.74	77.6	86.97	**87.10**
64	78.26	87.22	84.01	84.00	81.0	87.58	**88.55**
128	85.95	88.76	87.17	85.29	85.0	89.38	**89.78**
256	88.80	89.30	89.58	86.10	86.7	89.86	**90.07**
512	89.66	89.87	90.18	86.75	88.3	**90.18**	90.13

Initially, we use point clouds from the ModelNet40 dataset Wu et al. (2015b) to train the PointNet network Qi et al. (2017a) for the classification task. The point clouds consist of 1024 uniformly sampled points, with the official train-test split used for training and evaluation. To facilitate a comparison of sampling methods, we evaluate different sample sizes, reporting their respective performances. The vanilla PointNet achieves a baseline accuracy of 90.1% using all 1024 points.

Analyzing the classification results on the ModelNet40 dataset (Table 4.1), several noteworthy trends emerge. Firstly, as the sample size (m) increases, there is a consistent improvement in accuracies across various sampling methods. This is expected, as a larger sample size allows for better representation of the underlying point cloud, contributing to enhanced classification performance. Task-oriented sampling methods, exemplified by APSNet and PTSNet, consistently outperform task-agnostic approaches such as Random Sampling (RS) and Farthest Point Sampling (FPS). This underscores the significance of integrating task-specific information into the sampling process, enabling more informed and context-aware point cloud sampling. Comparing APSNet and PTSNet, it is evident that PTSNet consistently achieves higher accuracies across almost all sample sizes. This superior performance can be attributed to the effective utilization of the Transformer architecture in PTSNet. Transformers excel in capturing long-range dependencies in sequential data, making them well-suited for modeling point clouds. In contrast, APSNet uses a two-layer LSTM and may suffer from the potential vanishing gradient problem, as we will analyze in a later section. Moreover, the analysis reveals an interesting trend for larger sample sizes ($m \geq 512$), where accuracies tend to plateau. This suggests that beyond a certain point, increasing the sample size may yield diminishing returns in terms of classification improvement. This observation underscores the importance of finding a balance between sample size and computational efficiency for practical applications.

Table 4.2 Classification accuracies of different sampling methods with different sample sizes m on on the ScanObjectNN dataset.

m	RS	FPS	APSNet	PTSNet (Ours)
16	8.43	8.43	9.04	**10.11**
32	12.61	11.38	16.11	**17.43**
64	14.66	15.12	**20.23**	20.17
128	32.31	36.57	**40.29**	36.73
256	52.82	53.04	55.28	**60.14**
512	68.33	73.55	74.53	**76.72**
1024	78.15	81.56	79.53	**81.74**

ScanObjectNN Dataset

Table 4.2 presents the classification accuracies of two prominent sampling methods, APSNet and PTSNet (Ours), across various sample sizes (m) on the ScanObjectNN dataset. This dataset poses a greater challenge as it is sampled from real-world scans containing background and occlusions. Following the experiment settings of previous works, we conducted experiments on objects with background. The task network DGCNN was trained with 2,048 points, achieving a baseline accuracy of 82.2%. A discernible trend emerges, demonstrating that PTSNet consistently outperforms APSNet in terms of classification accuracy. For example, at $m = 16$, PTSNet achieves an accuracy of 10.11%, surpassing APSNet's 9.04%. This performance gap persists and even widens as the sample size increases. At $m = 1,024$, PTSNet achieves an impressive accuracy of 81.74%, outstripping APSNet's 79.53%. This significant and consistent improvement of PTSNet underscores its remarkable effectiveness over APSNet for this specific dataset and classification task. The utilization of the Transformer architecture and task-specific information in PTSNet evidently contributes to its ability to achieve more robust and accurate classification results on challenging real-world datasets compared to APSNet on the ScanObjectNN dataset.

Table 4.3 Statistics of unique points and inference time of two methods. APSNet tends to sample more duplicate points (less unique points) as sample size m increases.

	Unique Points		Inference Time (s)	
m	PTSNet	APSNet	PTSNet	APSNet
64	60	58	3.74	3.30
128	96	85	4.25	5.91
256	215	165	5.83	11.22
512	261	187	6.10	22.45
1024	342	218	6.15	40.12

Table 4.3 provides a comparative analysis between PTSNet and APSNet in terms of unique points and inference time across different sample sizes (m). It is important to note that APSNet employs an autoregressive approach based on LSTM, while PTSNet utilizes a non-autoregressive method. As the sample size (m) increases, we can observe that APSNet tends to sample more duplicate points, resulting in a reduced count of unique points in comparison to PTSNet. For instance, at a sample size of 1024, PTSNet captures a noteworthy 342 unique points, while APSNet captures only 218. In terms of inference time, APSNet

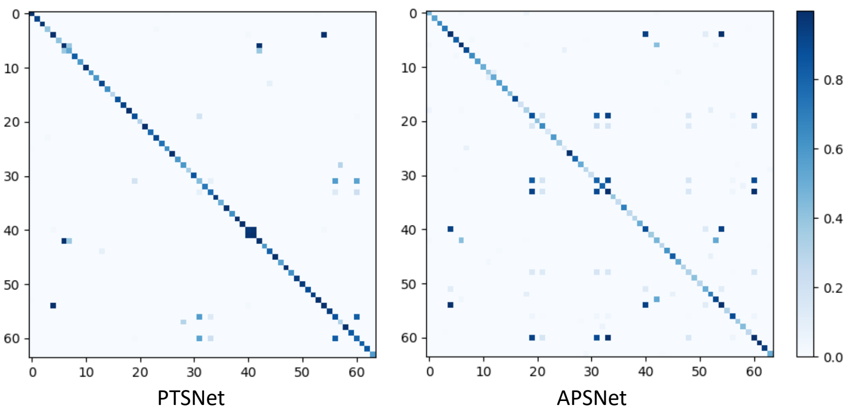

Figure 4.3 Correlation matrix of the selection matrix when $m = 64$ for the two methods. Selection vectors generated by PTSNet concentrated on one point to select in almost all the cases. APSNet tends to generate more duplicate points.

generally exhibits higher values compared to PTSNet as the sample size increases. This is expected, as the non-autoregressive nature of PTSNet allows for parallelized processing, contributing to faster inference. APSNet, which employs an autoregressive approach based on LSTM, experiences sequential processing overhead, leading to longer inference time.

The correlation matrix, calculated as $\mathbf{S} \cdot \mathbf{S}^{\mathrm{T}}$, illustrates the similarities between the selection vectors, and provides insights into the selection patterns of PTSNet and APSNet. Ideally, the correlation matrix should be close to an identity matrix, indicating no duplicate points is selected. Figure 4.3 reports the correlation matrices of PTSNet and APSNet. It can be observed that the selection vectors generated by PTSNet tend to concentrate on selecting unique point in most cases. On the other hand, APSNet shows a tendency of generating more duplicate points, which is also reflected by a lower number of unique points as shown in Table 4.3. The presence of duplicate points in the selection matrix of APSNet is likely attributed to the vanishing gradient caused by the two-layer LSTM employed by APSNet.

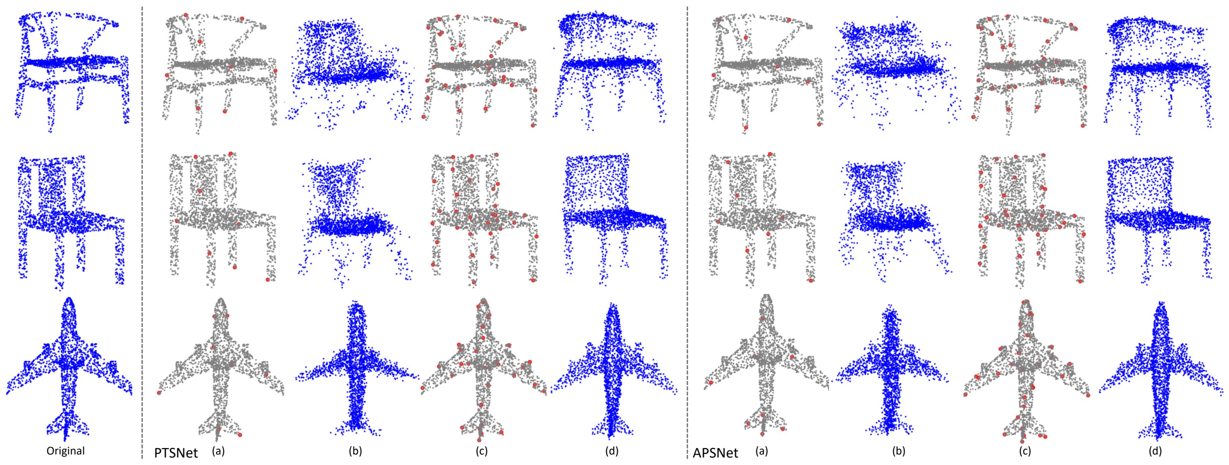

Figure 4.4 Visualization of sampled points and reconstructed point clouds by PTSNet (2nd block) and APSNet (3rd block). (a) Sampled points when $m = 8$; (b) Reconstruction when $m = 8$; (c) Sampled points when $m = 32$; (d) Reconstruction when $m = 32$. The red dots are the sampled points.

The vanishing gradient of LSTM limits the capability of APSNet to capture long-range correlations among the selection queries, leading to duplicate points and thus suboptimal performances.

4.4.2 Reconstruction

The reconstruction task is evaluated with point clouds of 2048 points, sampled from the ShapeNet Core55 dataset Chang et al. (2015). We chose the four shape classes that have the largest number of examples: Table, Car, Chair, and Airplane. Each class is split to a 85%, 5%, 10% partition for training, validation and test. The task network, in this case, is the Point Cloud Autoencoder (PCAE) for reconstruction Achlioptas et al. (2018). We evaluate the reconstruction performance with the normalized reconstruction error (NRE):

$$\mathsf{NRE}_{\mathsf{CD}}(\boldsymbol{Q}, \boldsymbol{P}) = \frac{\mathsf{CD}(\boldsymbol{P}, T(\boldsymbol{Q}))}{\mathsf{CD}(\boldsymbol{P}, T(\boldsymbol{P}))}, \tag{4.10}$$

where CD is the Chamfer distance Achlioptas et al. (2018) between two point clouds. Apparently, the values of $\mathsf{NRE}_{\mathsf{CD}}$ are lower bounded by 1, and the smaller, the better.

Table 4.4 The normalized reconstruction errors with different sample sizes m on ModelNet40.

m	RS	FPS	SampleNet	APSNet	PTSNet
8	21.85	12.79	5.48	**4.59**	4.78
16	13.47	7.25	2.89	2.62	**2.60**
32	8.16	3.84	1.71	1.59	**1.40**
64	4.54	2.23	1.27	**1.11**	**1.11**

Table 4.4 presents the reconstruction results for the different sampling methods, including PTSNet and APSNet. Consistent with the classification results, task-oriented samplers exhibit better performance compared to task-agnostic samplers. Specifically, both PTSNet and APSNet demonstrate superior reconstruction performance compared to random sampling and FPS. This highlights the effectiveness of incorporating task-specific information in the sampling process. Moreover, as the sample size m increases, PTSNet shows an improvement in performance. This suggests that PTSNet can capture more detailed information and preserve finer structures as the sample size increases.

For an in-depth analysis of the reconstruction quality, Figure 4.4 visually represents the sampled points and the reconstructed point clouds for PTSNet and APSNet. Notably, when $m = 8$, PTSNet and APSNet exhibit similar sampling behavior, focusing on capturing the outline of the airplane while preserving crucial details. As the sample size increases to $m = 32$, PTSNet outperforms APSNet in terms of reconstruction quality. The Normalized Reconstruction Errors (NRE) for PTSNet (1.36, 1.56, 1.25) are lower than those of APSNet (1.38, 1.57, 1.39), as indicated in column (d) of Figure 4.4. This signifies that PTSNet excels in preserving both the overall shape and finer details of the reconstructed point clouds. In summary, these results highlight the advantageous reconstruction performance of PTSNet over APSNet.

4.4.3 Registration

The registration task involves aligning two point clouds by estimating the rigid transformations between them. In this study, we utilize PCRNet as the task network for point cloud registration. The network is trained on the point clouds of the Car category from the ModelNet40 dataset, following the approach outlined in SampleNet. During training, 4,925 pairs of source and template point clouds are generated. Each template is randomly rotated by three Euler angles within the range of $[-45°, 45°]$ to obtain the corresponding source. For evaluation, an additional 100 source-template pairs are generated from the test split. The mean rotation error (MRE) is employed as the evaluation metric for the registration task. The MRE measures the average difference between the predicted rotations and the ground-truth rotations. Lower MRE values indicate better registration accuracy.

Table 4.5 The mean rotation errors with different sample sizes m on ModelNet40.

m	RS	FPS	SampleNet	APSNet	PTSNet
8	63.37	31.44	10.51	**9.40**	9.45
16	43.89	20.34	8.21	7.18	**7.08**
32	27.06	12.97	5.94	5.82	**5.53**
64	16.88	7.89	5.31	6.34	**5.08**

Table 4.5 presents the MRE of different sampling methods for the registration task on the ModelNet40 dataset. It can be observed that PTSNet consistently outperforms APSNet and SampleNet across almost all sample sizes. The lower the MRE, the better the registration accuracy. When the sample size is small, such as $m = 16$, PTSNet achieves a mean

rotation error of 7.08, outperforming APSNet (7.18) and SampleNet (8.21). As the sample size increases, PTSNet continues to demonstrate superior performance, achieving the lowest MRE at each sample size. For example, when $m = 64$, PTSNet achieves an MRE of 5.08, while APSNet and SampleNet achieve 6.34 and 5.31, respectively. These results indicate that PTSNet is effective in improving the accuracy of point cloud registration compared to APSNet and SampleNet. The Transformer-based architecture enables PTSNet to capture long-range dependencies and correlations among the sampled points, leading to more accurate registration results.

4.5 Conclusion

this chapter introduces a Point Transformer Sampling Network (PTSNet) to effectively sample points from 3D point clouds for downstream applications. By leveraging the power of transformer-based architectures, PTSNet outperforms existing sampling methods across various tasks, including classification, reconstruction, and registration. The utilization of the Transformer architecture and task-oriented sampling in PTSNet evidently contributes to its ability to achieve more robust and accurate results on challenging real-world ScanObjectNN dataset compared to APSNet.

CHAPTER 5
A Hybrid Generative and Discriminative PointNet on Unordered Point Sets

5.1 Introduction

In recent years, a myriad of processing methods Qi et al. (2017b); Yu et al. (2018); Li et al. (2018d); Qi et al. (2019) have been proposed for efficient point cloud analysis, and their performances in applications, such as 3D point cloud classification Qi et al. (2017b); Li et al. (2018d); Thomas et al. (2019); Wu et al. (2019b), semantic segmentation Li et al. (2018b); Su et al. (2018); Liu et al. (2019); Wang et al. (2019b, 2018); Landrieu & Simonovsky (2018) and reconstruction Achlioptas et al. (2018); Yang et al. (2018); Han et al. (2019); Zhao et al. (2019), have been improved significantly. Despite the significant progress of discriminative models for point cloud classification and segmentation, the research of generative models for point clouds is still far behind the discriminative ones. Learning generative models for point clouds is crucial to characterize the data distribution and analyze point clouds, which lays the foundation for various tasks such as shape completion, upsampling, synthesis and data augmentation. Although generative models such as variational auto-encoders (VAEs) Kingma & Welling (2014) and generative adversarial networks (GANs) Goodfellow et al. (2014) have shown great success in 2D image generation, it is challenging to extend these well-established methods to unordered 3D point sets. Images are structured data on 2D grids, while point clouds lie in irregular 3D space with variable densities. Existing methods for point could generation are mainly based on volumetric data, e.g., 3D GAN Wu et al. (2016), Generative VoxelNet Xie et al. (2018, 2020b), 3D-INN Huang et al. (2019), PointGrow Sun

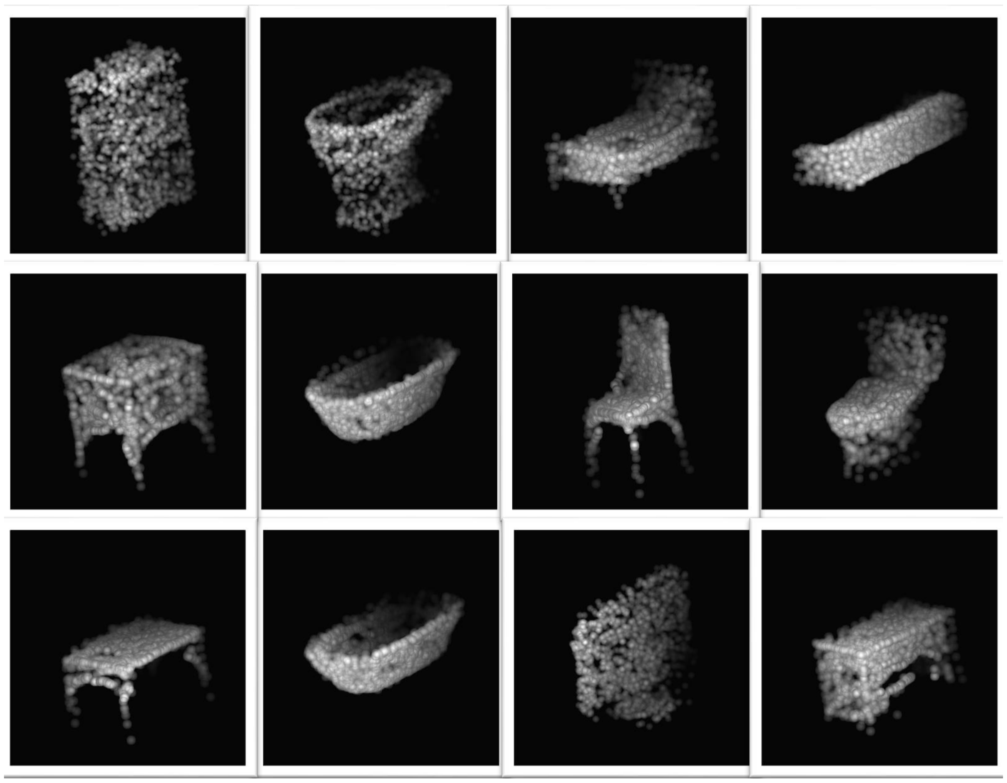

Figure 5.1 GDPNet performs point cloud classification and generation with a single network. It can generate 10 categories of point clouds, while achieving a 92.8% classification accuracy on ModelNet10. Sample point clouds generated by GDPNet are provided above.

et al. (2020), etc. While remarkable progress has been made, these methods have a few inherent limitations for point cloud generation. For instance, the training procedure of GAN-based approaches Wu et al. (2016) is rather unstable due to the adversarial losses, and the auto-regressive models Sun et al. (2020) assume an order of point generation, which is unnatural for orderless point cloud generation and restricts the modeling flexibility.

Energy-based models (EBMs) Zhu et al. (1998); LeCun et al. (2006) is a family of probabilistic generative models that can explicitly characterize the data distribution by learning an energy function that assigns lower values to observed data and higher values to unobserved ones. Besides, the training of EBMs can be much more stable in contrast to GANs by unifying representation and generation in one single model optimized by the maximum likelihood principle. Successful applications of EBMs include generations of images Xie et al. (2016, 2020a), videos Xie et al. (2017, 2019), 3D volumetric shapes Xie et al. (2018, 2020b), texts Deng et al. (2020), molecules Ingraham et al. (2018) as well as image-to-image translation Xie et al. (2021b) and out-of-distribution detection Liu et al. (2020). Recently, Xie et al. (2021a) propose GPointNet, an EBM for unordered point cloud generation. Unlike models that leverage an encoder-decoder architecture for generation, GPointNet Xie et al. (2021a) does not rely on either an auxiliary network or hand-crafted distance metrics to train the model. By incorporating PointNet Qi et al. (2017a), GPointNet extracts the features for each point independently and aggregates the point features from the whole point cloud into an energy scalar. The ”fake” examples are then generated by the Langevin dynamics sampling Welling & Teh (2011), and the model parameters are updated based on the energy difference between the ”fake” examples and the ”real” observed examples in order to match the ”fake” examples to the “real” ones in terms of some permutation-invariant statistical properties enabled by the energy function. Despite the model achieving an impressive performance for point cloud generation, one model needs to be trained separately for each category without sharing statistical regularities among different point cloud categories, e.g., structural smoothness and point density transition. Besides, their method is unable to classify point clouds directly and requires additional fine-tuning with an SVM classifier for classification.

In this chapter, we propose GDPNet, the first hybrid Generative and Discriminative PointNet for point cloud classification and generation with a single network. Our design follows the framework of joint energy-based model (JEM) Grathwohl et al. (2020), which reinterprets CNN softmax classifier as an EBM for image classification and generation. Instead, we extend JEM for point cloud classification and generation based on a modern PointNet classifier Qi et al. (2017a). The direct extension of JEM to PointNet, however, does not perform well as manifested by a classification accuracy gap to the standard classifier and a generation quality gap to the state-of-the-art generative approaches. We therefore investigate training techniques to bridge both gaps of GDPNet. We leverage the Sharpness-Aware Minimization (SAM) Foret et al. (2021) to improve the generalization of GDPNet (Section 5.3.2). We further demonstrate that the smoothness of the activation function can improve the training stability and synthesis quality of GDPNet significantly. As a result, our GDPNet retains strong discriminative power of modern PointNet classifier, while generating point cloud samples rivaling state-of-the-art generative approaches. More importantly, GDPNet yields one single model for classification and generation for all point cloud categories without resorting to a dedicated model for each category or additional fine-tuning step for classification. Example point clouds generated by GDPNet are provided in Figure 5.1.

5.2 Related Work

5.2.1 Deep Learning on Point Clouds

Following the breakthrough results of CNNs in 2D image processing tasks Krizhevsky et al. (2012); He et al. (2016a), there has been a strong interest in adapting such methods to 3D geometric data. Compared to 2D images, point cloud data are sparse, unordered and locality-sensitive, making it non-trivial to adapt CNNs to point cloud processing. Early attempts focus on regular representations of the data in the form of 3D voxels Wu et al. (2015b); Qi et al. (2016). These methods quantize point clouds into regular voxels in 3D space with a predefined resolution and then apply volumetric convolution. Recently, new designs of local aggregators over point clouds are proposed to improve the efficiency of point cloud processing and reduce the loss of details Qi et al. (2017a,b); Wang et al. (2019b). PointNet Qi et al. (2017a) is a pioneer in deep architecture design that directly processes point clouds for classification and semantic segmentation by a shared multi-layer perception (MLP) and a max-pooling layer. However, it treats each point independently and ignores the geometric relationships among them, and thus only local features are extracted. PointNet++ Qi et al. (2017b) further introduces a hierarchical aggregation of point features to extract global features. In later works, DGCNN Wang et al. (2019b) proposes an effective EdgeConv that encodes the point relationships as edge features to better capture local geometric features, while still maintaining permutation-invariance. Our GDPNet reinterprets PointNet classifier as an EBM and empowers it for point cloud generation while retaining its strong discriminative power.

5.2.2 Point Cloud Generation

Since point clouds lie in irregular 3D space with variable densities, early point cloud generation methods Achlioptas et al. (2018); Gadelha et al. (2018) convert the point cloud generation into a matrix generation problem. They take advantage of the power of well-established frameworks of variational auto-encoders (VAEs) Kingma & Welling (2014) and generative adversarial networks (GANs) Goodfellow et al. (2014) to train generative models with hand-crafted distance metrics, such as Chamfer distance or earth mover's distance, to measure the dissimilarity of two point clouds. The main defect of these methods is that they are restricted to generating point clouds with a fixed number of points and lack the property of permutation-invariance. FoldingNet Yang et al. (2018) and AtlasNet Groueix et al. (2018) learn a mapping that deforms the 2D patches into 3D shapes of point clouds to generate an arbitrary number of points, while being permutation-invariant. On the other hand, point clouds can also be regarded as samples from a point distribution and the maximum likelihood principle can be utilized for point cloud generation. PointFlow Yang et al. (2019a) employs continuous normalizing flows Grathwohl et al. (2019) to model the distribution of points. The invertibility of normalizing flows enables the computation of the likelihood during training and the variational inference is adopted for model training. PointGrow Sun et al. (2020) is an auto-regressive model that dynamically aggregates long-range dependencies among points for point cloud generation. ShapeGF Cai et al. (2020) proposes a score-matching energy-based model to represent the distribution of points. Luo and Hu Luo & Hu (2021) view points in a point cloud as particles in a thermodynamic system that diffuse from the original distribution to a noise distribution and leverage the reverse diffusion Markov chain to model the distribution of points. GPointNet Xie et al. (2021a) explicitly models this distribution as an EBM and learns the model by the maximum likelihood estimation. However, all these methods focus on point cloud generation and cannot perform classification at the same time. To the best of our knowledge, our GDPNet is the first hybrid generative and discriminative model for point cloud classification and generation with a single network.

5.2.3 Flat Minima and Generalization

A great number of prior works have investigated the relationship between the flatness of local minima and the generalization of learned models Li et al. (2018a); Keskar et al. (2017); Wei et al. (2020); Chen et al. (2022); Foret et al. (2021); Kwon et al. (2021). Now it is widely accepted and empirically verified that flat minima tend to give better generalization performance. Based on these observations, several recent regularization techniques are proposed to search for the flat minima of loss landscapes Wei et al. (2020); Chen et al. (2022); Foret et al. (2021); Kwon et al. (2021). Among them, the Sharpness-Aware Minimization (SAM) Foret et al. (2021) is a recently introduced optimizer that demonstrates promising performance across all kinds of models and tasks, such as ResNet He et al. (2016b), Vision Transformer Chen et al. (2022) and Language Models Bahri et al. (2022). Furthermore, score matching-based methods Hyvärinen (2005); Swersky et al. (2011); Song & Ermon (2019); Song et al. (2020) also explore the behavior of flat minima in generative models and learn unnormalized statistical models by matching the gradient of the log probability density of model distribution to that of data distribution. Our GDPNet incorporates SAM to promote the energy landscape smoothness and thus improves the generalization of trained EBMs.

5.3 The Proposed Method

We first introduce the Joint Energy-based Models (JEM) Grathwohl et al. (2020) and discuss its extension to GDPNet, the first hybrid generative and discriminative model for point clouds. We then present the Sharpness-Aware-Minimization (SAM) Foret et al. (2021) and its integration to GDPNet to improve the generalization of trained EBMs.

5.3.1 Joint Energy-based Models for Point Clouds

Let $\boldsymbol{X} = \{\boldsymbol{x}_i \in \mathcal{R}^3\}_{i=1}^n$ denote a point cloud that contains n points, with $\boldsymbol{x}_i$ representing the 3D coordinates of point i. Following the framework of Energy-Based Models (EBMs) Zhu et al. (1998); LeCun et al. (2006), we define the probability density function of a point cloud $\boldsymbol{X}$ explicitly as

$$p_{\boldsymbol{\theta}}(\boldsymbol{X}) = \frac{\exp\big(-E_{\boldsymbol{\theta}}(\boldsymbol{X})\big)}{Z(\boldsymbol{\theta})}, \tag{5.1}$$

where $E_{\boldsymbol{\theta}}(\boldsymbol{X})$ is an energy function, parameterized by $\boldsymbol{\theta}$, that maps input point cloud $\boldsymbol{X}$ to a scalar, and $Z(\boldsymbol{\theta}) = \int_{\boldsymbol{X}} \exp\big(-E_{\boldsymbol{\theta}}(\boldsymbol{X})\big)$ is the normalizing constant w.r.t. $\boldsymbol{X}$ (also known as the partition function). Ideally, the energy function should assign low energy values to the samples drawn from data distribution, and high values otherwise. Since point cloud $\boldsymbol{X}$ is a set of unordered points, the energy function, $E_{\boldsymbol{\theta}}(\boldsymbol{X})$, defined on a point set needs to be invariant to the permutation of points in the set $\boldsymbol{X}$. We follow the design of PointNet Qi et al. (2017a) to employ a shared multi-layer perception (MLP) for each point in the set $\boldsymbol{X}$, followed by an average-pooling layer, to approximate a continuous set function to process the unordered point sets.

The maximum likelihood estimate (MLE) can be used to estimate parameters $\boldsymbol{\theta}$ of $E_{\boldsymbol{\theta}}(\boldsymbol{X})$. However, since the partition function $Z(\boldsymbol{\theta})$ is intractable, the MLE of $\boldsymbol{\theta}$ is not straightforward.

Specifically, the derivative of the log-likelihood of $\boldsymbol{X}$ w.r.t. $\boldsymbol{\theta}$ can be expressed as

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{X})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{X})}\left[\frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{X})}{\partial \boldsymbol{\theta}}\right] - \mathbb{E}_{p_d(\boldsymbol{X})}\left[\frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{X})}{\partial \boldsymbol{\theta}}\right], \tag{5.2}$$

where $p_d(\boldsymbol{X})$ is the real data distribution (i.e., training dataset), and $p_{\boldsymbol{\theta}}(\boldsymbol{X})$ is the estimated probability density function (5.1), sampling from which is challenging due to the intractable $Z(\boldsymbol{\theta})$.

Prior works have developed a number of methods to sample from $p_{\boldsymbol{\theta}}(\boldsymbol{X})$ efficiently, such as MCMC and Gibbs sampling Hinton (2002). To speed up the sampling process further, recently Stochastic Gradient Langevin Dynamics (SGLD) Welling & Teh (2011) has been employed to sample from $p_{\boldsymbol{\theta}}(\boldsymbol{X})$ by utilizing the gradient information Nijkamp et al. (2019); Du & Mordatch (2019); Grathwohl et al. (2020). Specifically, to sample from $p_{\boldsymbol{\theta}}(\boldsymbol{X})$, SGLD follows

$$\boldsymbol{X}^0 \sim p_0(\boldsymbol{X}), \qquad \boldsymbol{X}^{t+1} = \boldsymbol{X}^t - \frac{\alpha}{2}\frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{X}^t)}{\partial \boldsymbol{X}^t} + \alpha \boldsymbol{\epsilon}^t, \tag{5.3}$$

where $\boldsymbol{\epsilon}^t$ is random noise that is sampled from a unit Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$, and $p_0(\boldsymbol{X})$ is typically a uniform distribution over $[-1, 1]$, whose samples are refined via a noisy gradient decent with step-size α over a sampling chain.

In order to train a hybrid generative and discriminative model, Joint Energy-based Model (JEM) Grathwohl et al. (2020) reinterprets the standard softmax classifier as an EBM. In particular, the logits $f_{\boldsymbol{\theta}}(\boldsymbol{X})[y]$ from a standard softmax classifier can be considered as an energy function over $(\boldsymbol{X}, y)$, where y is class label, and thus the joint density function of $(\boldsymbol{X}, y)$ can be expressed as $p_{\boldsymbol{\theta}}(\boldsymbol{X}, y) = e^{f_{\boldsymbol{\theta}}(\boldsymbol{X})[y]}/Z(\boldsymbol{\theta})$, where $Z(\boldsymbol{\theta})$ is an unknown normalizing constant (regardless of $\boldsymbol{X}$ or y). Then the density of $\boldsymbol{X}$ can be derived by marginalizing over y: $p_{\boldsymbol{\theta}}(\boldsymbol{X}) = \sum_y p_{\boldsymbol{\theta}}(\boldsymbol{X}, y) = \sum_y e^{f_{\boldsymbol{\theta}}(\boldsymbol{X})[y]}/Z(\boldsymbol{\theta})$. Subsequently, the corresponding energy function of $\boldsymbol{X}$ can be identified as

$$E_{\boldsymbol{\theta}}(\boldsymbol{X}) = -\log\sum_{y}\exp(f_{\boldsymbol{\theta}}(\boldsymbol{X})\,[y]) = -\mathrm{LSE}(f_{\boldsymbol{\theta}}(\boldsymbol{X})), \tag{5.4}$$

where LSE$(\cdot)$ denotes the Log-Sum-Exp function.

To optimize the model parameter $\boldsymbol{\theta}$, JEM maximizes the logarithm of joint density function $p_{\boldsymbol{\theta}}(\boldsymbol{X}, y)$:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{X}, y) = \log p_{\boldsymbol{\theta}}(y|\boldsymbol{X}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{X}), \tag{5.5}$$

where the first term denotes the cross-entropy objective for classification, and the second term can be optimized by the maximum likelihood learning of EBM as shown in Eq. (5.2). Sharing the same objective function (5.5) with JEM, GDPNet optimizes a single PointNet backbone for point cloud classification and generation. The overview of GDPNet is depicted in Figure 5.2.

5.3.2 Sharpness-Aware Minimization

The direct extension of JEM to PointNet above does not perform very well as manifested in our empirical studies (Tables 5.2, 5.3). In general, we notice two performance gaps of GDPNet as compared to the standard PointNet classifier and state-of-the-art generative approaches, i.e., a classification accuracy gap and a generation quality gap. We therefore

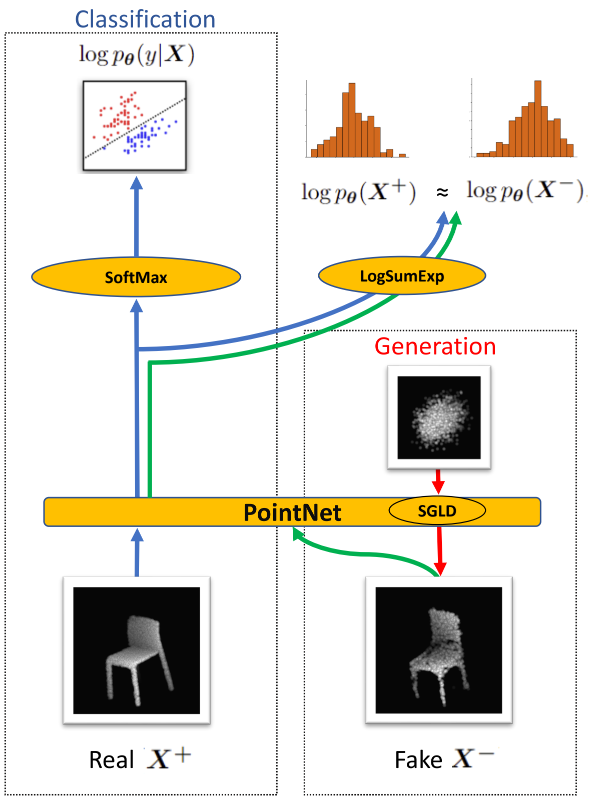
Classification
$\log p_{\boldsymbol{\theta}}(y|\boldsymbol{X})$
$\log p_{\boldsymbol{\theta}}(\boldsymbol{X}^+) \approx \log p_{\boldsymbol{\theta}}(\boldsymbol{X}^-)$
SoftMax
LogSumExp
Generation
PointNet
SGLD
Real $\boldsymbol{X}^+$
Fake $\boldsymbol{X}^-$

Figure 5.2 Overview of GDPNet architecture. Following the design of JEM Grathwohl et al. (2020), PointNet is leveraged for unordered point set feature extraction, and the LogSumExp$(\cdot)$ of the logits from the softmax classifier can be re-used to define an energy function of point cloud $\boldsymbol{X}$, which leads to a hybrid generative and discriminative model with the fake samples generated from the SGLD sampling. The model is optimized to perform the classification and maximize the energy difference between fake and real samples.

investigate training techniques to bridge both gaps of GDPNet. In particular, we leverage the Sharpness-Aware Minimization (SAM) Foret et al. (2021) to improve the generalization of GDPNet.

SAM Foret et al. (2021) is a recently proposed optimization method that searches for model parameters $\boldsymbol{\theta}$ whose entire neighborhoods have uniformly low loss values by optimizing a minimax objective:

$$\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{train}(\boldsymbol{\theta} + \boldsymbol{\epsilon}) + \lambda \|\boldsymbol{\theta}\|_2^2, \tag{5.6}$$

where ρ is the radius of an L_2-ball centered at model parameter $\boldsymbol{\theta}$, and λ is a hyperparameter for L_2 regularization on $\boldsymbol{\theta}$. To solve the inner maximization problem, SAM employs the Taylor expansion to develop an efficient first-order approximation to the optimal $\boldsymbol{\epsilon}^*$ as:

$$\begin{aligned}\hat{\boldsymbol{\epsilon}}(\boldsymbol{\theta}) &= \arg\max_{\|\epsilon\|_2 \leq \rho} L_{train}(\boldsymbol{\theta}) + \epsilon^T \nabla_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}) \\ &= \rho \nabla_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}) / \|\nabla_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta})\|_2, \end{aligned} \tag{5.7}$$

which is a scaled L_2 normalized gradient at the current model parameters $\boldsymbol{\theta}$. After $\hat{\boldsymbol{\epsilon}}$ is determined, SAM updates $\boldsymbol{\theta}$ based on the gradient $\nabla_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\epsilon}(\boldsymbol{\theta})} + 2\lambda\boldsymbol{\theta}$ at an updated parameter location $\boldsymbol{\theta} + \hat{\epsilon}$.

We incorporate SAM into the original training pipeline of GDPNet in order to improve the generalization of trained EBMs. Specifically, instead of the traditional maximum likelihood training of objective (5.5), we optimize the joint density function in a minimax objective:

$$\max_{\boldsymbol{\theta}} \min_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \log p_{(\boldsymbol{\theta}+\boldsymbol{\epsilon})}(\boldsymbol{X}, y) - \lambda \|\boldsymbol{\theta}\|_2^2. \tag{5.8}$$

For the outer maximization that involves $\log p_{\boldsymbol{\theta}}(\boldsymbol{X})$, SGLD is again used to sample from $p_{\boldsymbol{\theta}}(\boldsymbol{X})$ as in the original JEM.

Algorithm 1 GDPNet training: Given network f_{θ}, SGLD step-size α, SGLD noise σ, SGLD steps K, replay buffer B, reinitialization frequency γ, SAM noise bound ρ, and learning rate lr

```
1:  while not converged do
2:      Sample $\boldsymbol{X}^+$ and $y$ from training dataset
3:      Sample $\widehat{\boldsymbol{X}}_0 \sim B$ with probability $1-\gamma$, else $\widehat{\boldsymbol{X}}_0 \sim p_0(\boldsymbol{X})$
4:      for $t = 1, 2, \cdots, K$ do
5:          $\widehat{\boldsymbol{X}}_t = \widehat{\boldsymbol{X}}_{t-1} - \alpha \cdot \frac{\partial E(\widehat{\boldsymbol{X}}_{t-1})}{\partial \widehat{\boldsymbol{X}}_{t-1}} + \sigma \cdot \mathcal{N}(0, I)$
6:      end for
7:      $\boldsymbol{X}^- = \text{StopGrad}(\widehat{\boldsymbol{X}}_K)$
8:      $L_{\text{gen}}(\theta) = E(\boldsymbol{X}^+) - E(\boldsymbol{X}^-)$
9:      $L(\boldsymbol{\theta}) = L_{\text{clf}}(\boldsymbol{\theta}) + L_{\text{gen}}(\boldsymbol{\theta})$ with $L_{\text{clf}}(\boldsymbol{\theta}) = \text{xent}(f_{\boldsymbol{\theta}}(\boldsymbol{X}^+), y)$
10:     # Apply SAM optimizer as follows:
11:     Compute gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ of the training loss
12:     Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{\theta})$ with $\rho$ as in Eq. (5.7)
13:     Compute gradient $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\boldsymbol{\epsilon}(\hat{\boldsymbol{\theta}})}$
14:     Update model parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} - lr \cdot \boldsymbol{g}$
15:     Add $\boldsymbol{X}^-$ to $B$
16: end while
```

5.3.3 Smooth Activation Functions

We further study the effect of the activation function used in the energy function $E_{\boldsymbol{\theta}}(\boldsymbol{X})$. Zhao et al. (2021b) demonstrate that when data $\boldsymbol{X}$ is continuous, the smoothness of the activation function will substantially affect the Langevin sampling process (because the derivative $\frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{X})}{\partial \boldsymbol{X}}$ is inside of Eq. 5.3). Therefore, employing an activation function with continuous gradients everywhere can stabilize the sampling, while the non-smooth activation functions like ReLU Nair & Hinton (2011) and LeakyReLU Maas et al. (2013) may cause the divergence of EBM training. From our empirical studies, we have similar observations with the details provided in the experiments.

5.3.4 Training Algorithm

The pseudo-code of training GDPNet is provided in Algorithm 1, which follows a similar design of JEM Grathwohl et al. (2020) and JEM++ Yang & Ji (2021) with a replay buffer. For brevity, only one real sample and one generated sample are used to optimize the model parameter $\boldsymbol{\theta}$. But it is straightforward to generalize the pseudo-code to a mini-batch setting, which we use in our experiments. It is worth mentioning that we adopt the Informative Initialization in JEM++ to initialize the Markov chain from $p_0(\boldsymbol{X})$, which enables batch normalization Ioffe & Szegedy (2015b) in PointNet and plays a crucial role in the tradeoff between the number of SGLD sampling steps K and overall performance, including the classification accuracy and training stability.

5.4 Experiments

We evaluate the classification and generation performance of GDPNet in this section, and compare it with standard PointNet classifier Qi et al. (2017a) and state-of-the-art generative models, including PointFlow Yang et al. (2019a) and GPointNet Xie et al. (2021a). Ablation studies are performed to illustrate the impacts of SAM and smooth activation functions on the performance of GDPNet. Our source code is provided as a part of supplementary materials.

5.4.1 Experimental Setup

Our experiments largely follow the setup of GPointNet Xie et al. (2021a), and evaluate GDPNet for point cloud classification and generation on ModelNet10, which is a 10-category subset of ModelNet Wu et al. (2015a). We first create a dataset from ModelNet10 by sampling 2,048 points uniformly from the mesh surface of each object and then scale the point cloud features to the range of [-1, 1]. In contrast to GPointNet Xie et al. (2021a), which trains 10 models to generate point clouds for 10 different categories of ModelNet10, we train one single network to classify and generate point clouds for all 10 categories.

For a fair comparison with GPointNet, PointNet Qi et al. (2017a) is used as the backbone network of our GDPNet. As discussed earlier, PointNet is permutation-invariant and thus works well with unordered point sets. It first maps each point (i.e., 3-dimensional coordinates) of a point cloud to a 1,024-dimensional feature vector by an MLP, then leverages an average pooling layer to aggregate information from all the points to a 1,024-dimensional global feature vector to represent the point cloud. A softmax layer is then appended at the end of the network to yield the logits for the 10 categories, which are used for classification and to calculate the energy score via an LSE($\cdot$) operator (Eq. 5.4).

Furthermore, GDPNet employs SAM Foret et al. (2021) to improve the generalization of trained EBMs. In our experiments, we use Adam Kingma & Ba (2015) as the base optimizer for SAM with an initial learning rate of 0.01, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We set the learning rate decay multiplier to 0.2 for every 50 iterations. We adopt the informative initialization of JEM++ Yang & Ji (2021) to initialize the Markov chain from $p_0(\boldsymbol{X})$. The reinitialization frequency γ is set to 0.05, and the replay buffer size is set to 5000. With the informative initialization, the number of SGLD sampling steps K can be reduced significantly as compared to that of GPointNet Xie et al. (2021a). In our experiments, we set $K = 32$

with a step size $\alpha = 0.05$. To mitigate the exploding gradients in the SGLD sampling, we clip the gradient values to the range of [-1, 1] at each sampling step. We run 200 epochs for training with a minibatch size of 128.

	Model	JSD (↓)	MMD (↓)		Coverage (↑)			Model	JSD (↓)	MMD (↓)		Coverage (↑)	
			CD	EMD	CD	EMD				CD	EMD	CD	EMD
night stand	r-GAN	2.679	1.163	2.394	50.00	38.37	sofa	r-GAN	1.866	2.037	2.247	13.00	23.00
	l-GAN	1.000	0.746	1.563	44.19	39.53		l-GAN	0.681	0.631	**1.028**	43.00	44.00
	PointFlow	**0.240**	0.888	1.451	55.81	39.53		PointFlow	**0.244**	0.585	1.313	34.00	33.00
	GPointNet	0.590	**0.692**	**1.148**	**59.30**	**61.63**		GPointNet	0.647	**0.547**	1.089	39.00	45.00
	Ours	0.3771	0.886	1.369	54.83	59.30		Ours	0.275	0.576	1.104	**45.00**	**46.00**
	Training Set	0.263	0.793	1.096	60.40	52.32		Training Set	0.185	0.467	0.904	56.00	56.00
toilet	r-GAN	3.180	2.995	2.891	17.00	16.00	bed	r-GAN	1.973	1.250	2.441	27.00	21.00
	l-GAN	1.253	1.258	1.481	21.00	28.00		l-GAN	0.646	**0.539**	**0.992**	48.00	44.00
	PointFlow	**0.362**	0.965	1.513	39.00	33.00		PointFlow	**0.219**	0.544	1.230	**50.00**	35.00
	GPointNet	0.386	**0.816**	**1.265**	**44.00**	37.00		GPointNet	0.461	0.552	1.004	**50.00**	**50.00**
	Ours	0.578	0.867	1.314	43.00	**42.00**		Ours	0.240	0.540	1.088	45.00	41.00
	Training Set	0.249	0.823	1.116	48.00	51.00		Training Set	0.169	0.516	0.927	57.00	55.00
monitor	r-GAN	2.936	1.524	2.021	21.00	24.00	table	r-GAN	3.801	3.714	2.625	8.00	14.00
	l-GAN	1.653	0.915	1.349	28.00	27.00		l-GAN	4.254	1.232	2.166	14.00	9.00
	PointFlow	**0.326**	0.831	1.288	37.00	32.00		PointFlow	1.044	1.630	1.535	16.00	29.00
	GPointNet	0.780	0.803	1.213	40.00	38.00		GPointNet	0.869	**0.640**	**1.000**	**44.00**	**37.00**
	Ours	0.434	**0.535**	**1.029**	**52.00**	**46.00**		Ours	**0.761**	1.085	1.299	38.00	33.00
	Training Set	0.283	0.554	0.938	48.00	53.00		Training Set	0.703	1.218	1.182	31.00	38.00
chair	r-GAN	2.772	1.709	2.164	23.00	28.00	desk	r-GAN	3.575	2.712	3.678	22.09	22.09
	l-GAN	1.358	1.419	1.480	23.00	26.00		l-GAN	2.233	1.139	2.345	38.37	25.58
	PointFlow	**0.278**	0.965	1.322	42.00	51.00		PointFlow	0.327	1.254	1.548	38.37	46.51
	GPointNet	0.563	**0.889**	**1.280**	**56.00**	**57.00**		GPointNet	0.454	1.223	1.567	**56.98**	**52.33**
	Ours	0.387	0.909	1.361	44.00	50.00		Ours	0.512	**1.077**	**1.486**	55.81	50.51
	Training Set	0.365	0.858	1.190	54.00	59.00		Training Set	0.329	1.055	1.332	53.48	50.00
bathtub	r-GAN	3.014	2.478	2.536	26.00	30.00	dresser	r-GAN	1.726	1.299	1.675	36.05	30.23
	l-GAN	0.928	0.865	1.324	32.00	38.00		l-GAN	0.648	0.642	1.010	45.35	43.02
	PointFlow	**0.350**	**0.593**	1.320	50.00	44.00		PointFlow	**0.270**	0.715	1.349	46.51	37.21
	GPointNet	0.460	0.660	**1.108**	**58.00**	**50.00**		GPointNet	0.457	**0.485**	**0.988**	**53.49**	**52.33**
	Ours	0.490	0.647	1.103	54.00	**50.00**		Ours	0.440	0.708	1.125	48.88	49.53
	Training Set	0.344	0.652	0.980	56.00	52.00		Training Set	0.215	0.551	0.882	56.98	54.65

Table 5.1 Qualities of point cloud synthesis on ModelNet10 from different methods. In contrast to the state-of-the-art generative approaches, GDPNet only trains a single network to generate all the 10 categories of ModelNet10. ↓: the lower the better; ↑: the higher the better. MMD-CD scores are multiplied by 100; MMD-EMD scores and JSDs are multiplied by 10.

5.4.2 Evaluation Metrics

We adopt three evaluation metrics: Jensen-Shannon Divergence (JSD), Coverage (COV) and Minimum Matching Distance (MMD) to evaluate the quality of generated point clouds. These metrics are commonly used in prior works Achlioptas et al. (2018); Yang et al. (2019a); Xie et al. (2021a) for point cloud quality evaluation. When evaluating COV and MMD, we use Chamfer Distance (CD) and Earth Mover's Distance (EMD) to measure the dissimilarity between two point clouds, which are defined formally as follows:

$$\begin{aligned}
\mathrm{CD}(X,Y) &= \sum_{x\in X}\min_{y\in Y}\|x-y\|_2^2 + \sum_{y\in Y}\min_{x\in X}\|x-y\|_2^2, \\
\mathrm{EMD}(X,Y) &= \min_{\phi:X\to Y}\sum_{x\in X}\|x-\phi(x)\|_2,
\end{aligned}$$

where X and Y are two point clouds with the same number of points and ϕ is a bijection between them.

Jensen-Shannon Divergence (JSD) is a symmetrized Kullback-Leibler divergence between two marginal point distributions:

$$\mathrm{JSD}(P_g, P_r) = \frac{1}{2}D_{KL}(P_r||M) + \frac{1}{2}D_{KL}(P_g||M)\,,$$

where $M = \frac{1}{2}(P_r + P_g)$, P_r and P_g are marginal distributions of points in the reference and generated sets, approximated by discretizing the space into 28^3 voxels and assigning each point to one of them.

Coverage (COV) measures the fraction of point clouds in the reference set that are matched to at least one point cloud in the generated set. For each point cloud in the generated set, its nearest neighbor in the reference set is marked as a match:

$$\mathrm{COV}(S_g, S_r) = \frac{|\{\arg\min_{Y\in S_r} D(X,Y)|X\in S_g\}|}{|S_r|},$$

	Method	JSD (↓)	MMD (↓)		Coverage (↑)	
			CD	EMD	CD	EMD
Night Stand	ReLU	0.487	0.911	1.320	43.02	47.67
	CELU	**0.368**	**0.867**	**1.311**	53.48	54.65
	CELU + SAM	0.377	0.886	1.369	**54.83**	**59.30**
Toilet	ReLU	0.581	1.022	1.478	31.00	42.00
	CELU	**0.546**	0.941	1.380	32.00	37.00
	CELU + SAM	0.578	**0.867**	**1.314**	**43.00**	**42.00**
Monitor	ReLU	0.410	0.692	1.261	45.00	46.00
	CELU	**0.389**	0.571	1.085	46.00	44.00
	CELU + SAM	0.434	**0.535**	**1.029**	**52.00**	**46.00**
Chair	ReLU	0.449	0.916	1.495	44.00	46.00
	CeLU	**0.363**	**0.8541**	**1.316**	**45.00**	**51.00**
	CELU + SAM	0.387	0.909	1.361	44.00	50.00

Table 5.2 The impacts of SAM and activation functions on the qualities of generated point clouds by GDPNet. ↓: the lower the better; ↑: the higher the better. MMD-CD scores are multiplied by 100; MMD-EMD scores and JSDs are multiplied by 10.

where $D(\cdot, \cdot)$ can be either CD or EMD.

Minimum Matching Distance (MMD) is proposed to complement coverage to measure the quality of generated point clouds. For each point cloud in the reference set, the distance to its nearest neighbor in the generated set is computed and averaged:

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} D(X, Y).$$

5.4.3 Results of Point Cloud Generation

We compare the generative performance of GDPNet with four baseline generative approaches: l-GAN Achlioptas et al. (2018), r-GAN Achlioptas et al. (2018), PointFlow Yang et al. (2019a) and GPointNet Xie et al. (2021a), and the results are reported in Table 5.1. It can be observed that GDPNet achieves a competitive generative performance as compared to the state-of-the-art results of PointFlow and GPointNet even though our method only employs a single network to generate point clouds from 10 different categories of ModelNet10 [1]. For the

[1]Let alone our GDPNet can also classify point cloud directly with an accuracy of 92.8%.

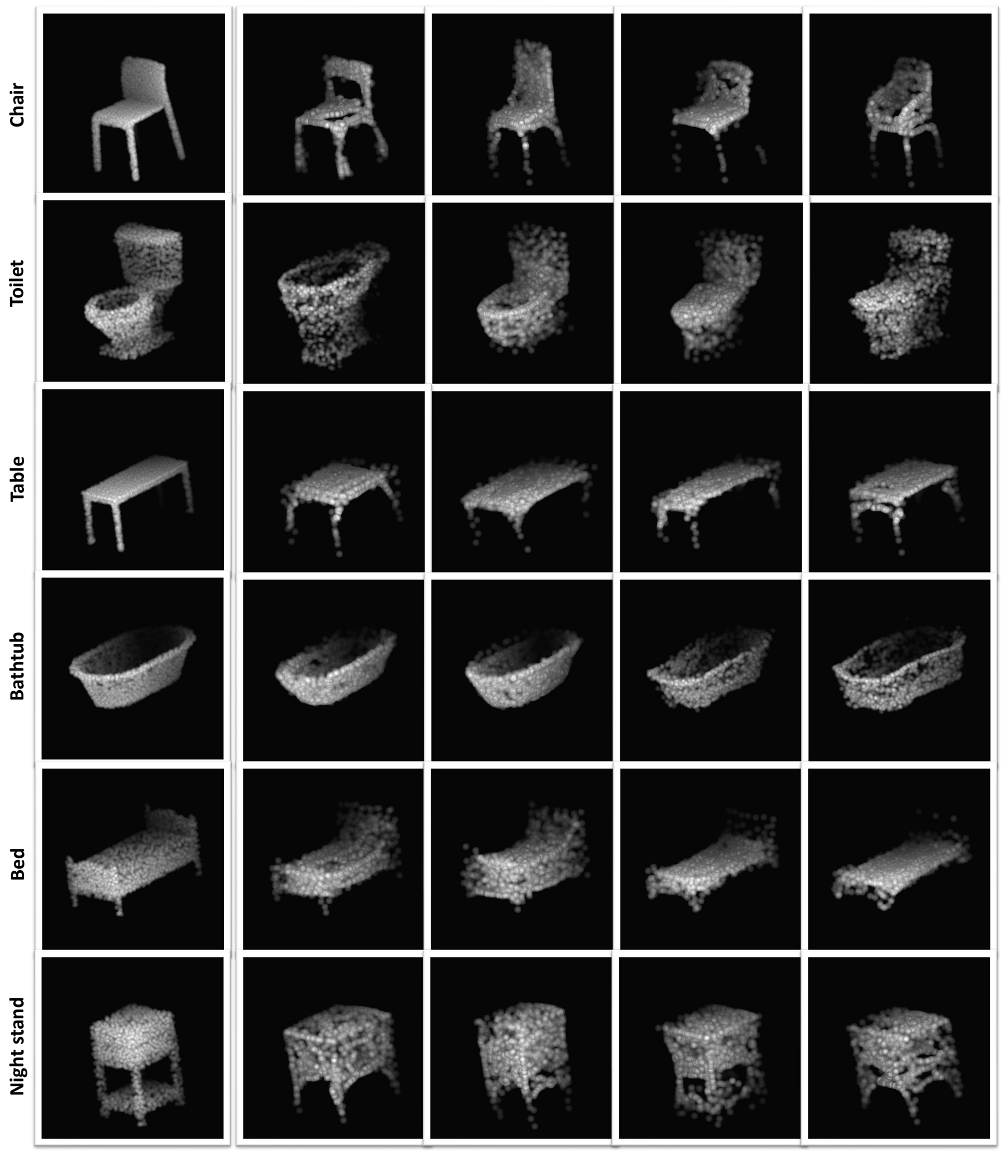

Figure 5.3 Sample point clouds generated by GDPNet. Each row corresponds to one category. The first column is a sample from ModelNet10 training set, and the rest of the columns are synthesized point clouds generated via SGLD.

generation of ”monitor”, GDPNet achieves an even better result than that of GPointNet, while being competitive with GPointNet for the rest of the categories. As shown in Figure 5.1, GPDNet can learn the complex distributions among all the categories and generate point clouds of each category with diverse styles. It also can generate point clouds that have

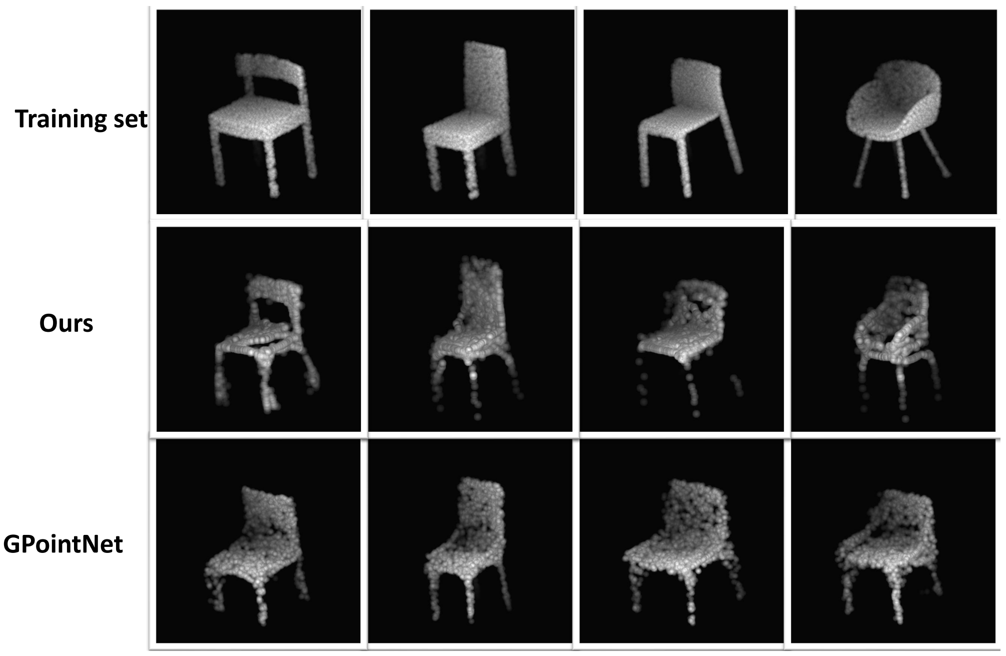

Figure 5.4 Sample point clouds generated by GPointNet Xie et al. (2021a) and GDPNet. Our GDPNet generates chairs with more diverse styles, while GPointNet generates chairs with better details on the four legs.

features from multiple categories, such as a toilet-like chair. This is because toilet and chair share some similar features, and GDPNet can generate samples that interpolate between them.

Figure 5.3 provides more sample point clouds generated by GDPNet for categories of "chair", "toilet", "table", "bathtub", "bed" and "night stand". These samples are selected when the GDPNet classifier has a classification confidence over 90%. Not surprisingly, the generated samples for each category are exactly as GDPNet predicted, which also indicates an accurate classification of GDPNet. The results in Figure 5.3 show that GDPNet can learn the complex point distribution to generate quality point cloud samples. For "chair" and "table", the details of four legs are captured by our model, and the "bathtub" samples are as good as training samples, while the toilet samples are also decent. As for "night stand", whose shape is much more complex, the generated samples can still capture parts of the object features. We also used the official GPointNet checkpoint for the chair category to generate 1,000 samples, from which we selected some high-quality ones and compare them with the samples from our GDPNet. It can be observed from Figure 5.4 that GDPNet can generate chairs with more diverse styles, while GPointNet generates chairs with better details on the four legs. As a result, GDPNet has a slightly lower evaluation value on "chair" as reported in Table 5.1.

As for model complexity, GDPNet has a 1/10 model size of GPointNet since GDPNet only trains one single network to generate point clouds for all the 10 categories, which is a significant advantage of our method as compared to other generative approaches. For models based on VAEs or GANs, their model sizes are even larger than that of GPointNet as they need auxiliary networks for training.

5.4.4 Ablation Studies

We study the impacts of SAM and the activation functions on the performance of GDPNet for point cloud classification and generation, respectively.

Point cloud generation Table 5.2 reports the impacts of SAM and activation functions to the generative performance of GDPNet. By replacing the popular ReLU activation function Nair & Hinton (2011) with CELU Barron (2017), a continuously differentiable exponential linear unit, GDPNet achieves notable quality gains in generating point clouds of different categories. This observation is consistent with that of Zhao et al. (2021b) who demonstrate that the smoothness of the activation functions substantially improves the SGLD sampling process and thus the synthesis quality. Table 5.2 also shows that incorporating SAM to GDPNet does not improve the synthesis quality consistently. However, from our experiments, we find that SAM facilitates the convergence of the model, stabilizes the training of GDPNet, and improves the classification accuracy as shown in the experiments below.

Point cloud classification Table 5.3 reports the impacts of SAM and the activation functions on the classification performance of GDPNet. It can be observed that without SAM and CELU activation function, GDPNet has a non-competitive classification accuracy of 90.7% as compared to the standard PointNet classifier, which achieves a 92.8% accuracy. Incorporating SAM into GDPNet significantly improves the classification accuracy (92.8%), which matches with that of the standard PointNet classifier. Replacing CELU by ReLU in GDPNet does not affect the classification accuracy much (92.9% vs. 92.8%), but GDPNet with CELU achieves the best synthesis quality as shown in Table 5.2. Therefore, GDPNet with SAM and CELU bridges both the classification accuracy gap and the synthesis quality gap as compared to the standard PointNet classifier and state-of-the-art generative approaches.

It is worth mentioning that even though l-GAN Achlioptas et al. (2018), PointFlow Yang et al. (2019a) and GPointNet Xie et al. (2021a) achieve better classification accuracies as reported in Table 5.3. They can not classify point clouds directly since they are generative models. Specifically, to classify point clouds with GPointNet Xie et al. (2021a), an oneversus-all SVM classifier needs to be trained on the extracted features of GPointNet and the class labels. A similar procedure has also been used by l-GAN and PointFlow for classification. In contrast, GDPNet does not need any extra training step for classification, which is another significant advantage of GDPNet as compared to these generative approaches.

Method	Accuracy
l-GAN* Achlioptas et al. (2018)	95.4%
PointFlow* Yang et al. (2019a)	93.7%
GPointNet* Xie et al. (2021a)	93.7%
PointNet Qi et al. (2017a)	92.8%
GDPNet w/ ReLU - SAM	90.7%
GDPNet w/ ReLU	92.9%
GDPNet - SAM	90.3%
GDPNet	92.8%

Table 5.3 Point cloud classification accuracies on ModelNet10. * denotes the method needs to train an SVM classifier on the extracted features for classification.

5.5 Conclusion

This chapter introduces GDPNet, a hybrid generative and discriminative model for point clouds, that is based on joint energy-based models. GDPNet further leverages SAM and CELU activation function to bridge the classification accuracy gap and the generation quality gap to the standard PointNet classifier and state-of-the-art generative models. Compared to prior generative models of point clouds, GDPNet only trains a single compact network to classify and generate point clouds of all categories. Experiments demonstrate our GDPNet retains strong discriminative power of modern PointNet classifiers, while generating point cloud samples rivaling state-of-the-art generative approaches. To the best of our knowledge, GDPNet is the first hybrid generative and discriminative model for point clouds.

CHAPTER 6
Conclusion

This dissertation revolves around the exploration of graph neural networks (GNNs) and the analysis of 3D point clouds, with a specific emphasis on sparsification, classification and generative.

Chapter 2: Sparse Graph Attention Networks (SGAT): we introduce Sparse Graph Attention Networks (SGAT), seamlessly integrating a sparse attention mechanism into graph attention networks (GATs) through innovative L0 -norm regularization. SGAT not only effectively eliminates noisy edges but also exhibits exceptional capabilities in edge removal. It achieves reductions of 50%-80% on large graphs without compromising performance on assortative graphs, simultaneously enhancing performance on disassortative graphs.

Chapters 3 and 4: Point Cloud Sampling Approaches: these two chapters present both autoregressive and non-autoregressive approaches for task-oriented point cloud sampling. APSNet employs a sequential autoregressive generation with a novel LSTM-based sequential model for sampling and achieves optimal performance with only 8 out of 1024 points, tailoring the sampling process for downstream tasks like 3D point cloud classification, reconstruction, and registration. PTSNet, a point transformer, enhances performance by leveraging a global representation of the point cloud and a transformer-based dynamic query generator. This addresses issues such as gradient vanishing and reduces duplicate samples compared to LSTM-based methods.

Chapter 5: Hybrid Generative and Discriminative PointNet (GDPNet): In this chapter, we introduce GDPNet, a hybrid Generative and Discriminative PointNet. An extension of the Joint Energy-based Model (JEM), GDPNet seamlessly integrates the robust discriminative power of modern PointNet classifiers with the generation of point cloud samples rivaling state-of-the-art approaches.

REFERENCES

Achlioptas, P., Diamanti, O., Mitliagkas, I., & Guibas, L. J. 2018, Proceedings of the 35th International Conference on Machine Learning (ICML), 40

Bahdanau, D., Cho, K., & Bengio, Y. 2015, in International Conference on Representation Learning (ICLR)

Bahri, D., Mobahi, H., & Tay, Y. 2022, in Annual Meeting of the Association for Computational Linguistics (ACL)

Barron, J. T. 2017, arXiv preprint arXiv:1704.07483

Beltrán, J., Guindel, C., Moreno, F. M., Cruzado, D., García, F., & De La Escalera, A. 2018, in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 3517–3523

Bird, T., Kunze, J., & Barber, D. 2018, arXiv preprint arXiv:1809.04855

Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. 2014, in International Conference on Representation Learning (ICLR)

Cai, R., Yang, G., Averbuch-Elor, H., Hao, Z., Belongie, S., Snavely, N., & Hariharan, B. 2020, in European Conference on Computer Vision, Springer, 364–381

Calandriello, D., Koutis, I., Lazaric, A., & Valko, M. 2018, in International Conference on Machine Learning (ICML)

Cao, S., Lu, W., & Xu, Q. 2015, in ACM International on Conference on Information and Knowledge Management (CIKM)

Chakeri, A., Farhidzadeh, H., & Hall, L. O. 2016, in International Conference on Learning Representations (ICLR)

Chang, A. X. et al. 2015, arXiv preprint arXiv:1512.03012

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., & Sun, X. 2020a, in AAAI

Chen, H. et al. 2020b, in Proceedings of the 29th ACM International Conference on Information Knowledge Management

Chen, J., Song, L., Wainwright, M. J., & Jordan, M. I. 2018, Proceedings of the International Conference on Machine Learning (ICML)

Chen, X., Hsieh, C.-j., & Gong, B. 2022, in International Conference on Learning Representations (ICLR)

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. 2019, arXiv preprint arXiv:1901.02860

Defferrard, M., Bresson, X., & Vandergheynst, P. 2016, in NIPS

Deng, Y., Bakhtin, A., Ott, M., Szlam, A., & Ranzato, M. 2020, arXiv preprint arXiv:2004.11714

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2019, in Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL)

Dosovitskiy, A. et al. 2021, in International Conference on Learning Representations (ICLR)

Dovrat, O., Lang, I., & Avidan, S. 2019a, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2760–2769

Dovrat, O., Lang, I., & Avidan, S. 2019b, Proceedings of the IEEE Conference on Computer

Vision and Pattern Recognition (CVPR), 2760

Du, Y., & Mordatch, I. 2019, in Neural Information Processing Systems (NeurIPS)

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gomez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. 2015, in Advances in Neural Information Processing Systems (NIPS)

Eldar, Y., Lindenbaum, M., Porat, M., & Zeevi, Y. Y. 1997, IEEE Transactions on Image Processing, 6, 1305

Engelcke, M., Rao, D., Wang, D. Z., Tong, C. H., & Posner, I. 2017, in 2017 IEEE International Conference on Robotics and Automation (ICRA), 1355–1361

Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. 2021, in International Conference on Learning Representations

Gadelha, M., Wang, R., & Maji, S. 2018, in Proceedings of the European Conference on Computer Vision (ECCV), 103–118

Geiger, A., Lenz, P., & Urtasun, R. 2012, in 2012 IEEE Conference on Computer Vision and Pattern Recognition, 3354–3361

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. 2014, in Neural Information Processing Systems (NeurIPS)

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., & Duvenaud, D. 2019, in International Conference on Learning Representations (ICLR)

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., & Duvenaud, D. 2018, in International Conference on Learning Representations (ICLR)

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., Norouzi, M., & Swersky, K. 2020, in International Conference on Learning Representations (ICLR)

Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., & Aubry, M. 2018, in Proceedings of the IEEE conference on computer vision and pattern recognition, 216–224

Grover, A., & Leskovec, J. 2016, in Knowledge Discovery and Data mining (KDD)

Hamilton, W. L., Ying, R., & Leskovec, J. 2017a, in Advances in Neural Information Processing Systems (NIPS)

Hamilton, W. L., Ying, R., & Leskovec, J. 2017b, IEEE Data Engineering Bulletin, 40, 52

Han, Z., Wang, X., Liu, Y.-S., & Zwicker, M. 2019, Proceedings of the International Conference on Computer Vision (ICCV)

Hasanzadeh, A., Hajiramezanali, E., Boluki, S., Zhou, M., Duffield, N., Narayanan, K., & Qian, X. 2020, in arXiv preprint arXiv:2006.04064

He, K., Zhang, X., Ren, S., & Sun, J. 2016a, in Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)

He, K., Zhang, X., Ren, S., & Sun, J. 2016b, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

Hinton, G., Vinyals, O., & Dean, J. 2014, Proceedings of Advances in Neural Information Processing Systems (NIPS)

Hinton, G. E. 2002, Neural computation

Hochreiter, S., & Schmidhuber, J. 1997, Neural Computation, 9, 1735

Hu, H., Zhang, Z., Xie, Z., & Lin, S. 2019, in Proceedings of the IEEE/CVF International

Conference on Computer Vision, 3464–3473

Huang, W., Lai, B., Xu, W., & Tu, Z. 2019, in Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 8481–8488

Hyvärinen, A. 2005, Journal of Machine Learning Research

Ingraham, J., Riesselman, A., Sander, C., & Marks, D. 2018, in International Conference on Learning Representations

Ioffe, S., & Szegedy, C. 2015a, Proceedings of the International Conference on Machine Learning (ICML)

Ioffe, S., & Szegedy, C. 2015b, in International Conference on Machine Learning (ICML)

Jang, E., Gu, S., & Poole, B. 2017, in International Conference on Learning Representations (ICLR)

Ji, S., Satish, N., Li, S., & Dubey, P. 2016, in NIPS workshop on Efficient Methods for Deep Neural Networks

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. 2017, in International Conference on Learning Representations (ICLR)

Kim, D., & Oh, A. 2021, in International Conference on Learning Representations

Kingma, D. P., & Ba, J. 2014, arXiv preprint arXiv:1412.6980

Kingma, D. P., & Ba, J. 2015, in International Conference on Learning Representations (ICLR)

Kingma, D. P., & Welling, M. 2014, in International Conference on Learning Representations (ICLR)

Kipf, T. N., & Welling, M. 2017, in International Conference on Learning Representations (ICLR)

Krizhevsky, A., Sutskever, I., & Hinton, G. E. 2012, in Advances in Neural Information Processing Systems (NeurIPS)

Kwon, J., Kim, J., Park, H., & Choi, I. K. 2021, in International Conference on Machine Learning (ICML)

Landrieu, L., & Simonovsky, M. 2018, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4558

Lang, I., Manor, A., & Avidan, S. 2020, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 7578–7588

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., & Huang, F. 2006, Predicting structured data

Li, B. 2017, in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1513–1518

Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. 2018a, in Neural Information Processing Systems (NeurIPS)

Li, J., Chen, B. M., & Lee, G. H. 2018b, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 9397

Li, Q., Han, Z., & Wu, X.-M. 2018c, in 32th AAAI Conference on Artificial Intelligence

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., & Chen, B. 2018d, Proceedings of Advances in Neural Information Processing Systems (NeuralIPS)

Lin, Y., Huang, Y., Zhou, S., Jiang, M., Wang, T., & Lei, Y. 2021, in 2021 4th International Conference on Pattern Recognition and Artificial Intelligence (PRAI), IEEE, 13–18

Liu, W., Wang, X., Owens, J., & Li, Y. 2020, Neural Information Processing Systems (NeurIPS)

Liu, Y., Fan, B., Xiang, S., & Pan, C. 2019, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 8895

Louizos, C., Welling, M., & Kingma, D. P. 2018, in International Conference on Learning Representations (ICLR)

Luo, D., Cheng, W., Yu, W., Zong, B., Ni, J., Chen, H., & Zhang, X. 2021, in Proceedings of the 14th ACM International Conference on Web Search and Data Mining

Luo, S., & Hu, W. 2021, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2837–2845

Maas, A., Hannun, A., & Ng, A. 2013, in International Conference on Machine Learning (ICML)

Maaten, L. v. d., & Hinton, G. 2008, Journal of Machine Learning Research (JMLR), 9, 2579

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. 2013, in Advances in Neural Information Processing Systems 26, 3111–3119

Moenning, C., & Dodgson, N. A. 2003, Eurographics Poster Presentation

Nair, V., & Hinton, G. E. 2011, in International Conference on Machine Learning (ICML)

Nijkamp, E., Zhu, S.-C., & Wu, Y. N. 2019, in Neural Information Processing Systems

(NeurIPS)

Nüchter, A., & Hertzberg, J. 2008, Robotics Auton. Syst., 56, 915

Ou, M., Cui, P., Pei, J., Zhang, Z., & Zhu, W. 2016, in Knowledge Discovery and Data mining (KDD)

Park, Y., Lepetit, V., & Woo, W. 2008, in 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, 117–120

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., & Yang, B. 2020, in Knowledge Discovery and Data Mining(KDD)

Pennington, J., Socher, R., & Manning, C. D. 2014, in Empirical Methods in Natural Language Processing (EMNLP), 1532–1543

Perozzi, B., Al-Rfou, R., & Skiena, S. 2014, in Knowledge Discovery and Data mining (KDD)

Qi, C. R., Litany, O., He, K., & Guibas, L. J. 2019, Proceedings of the International Conference on Computer Vision (ICCV)

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. 2017a, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 652

Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., & Guibas, L. J. 2016, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5648

Qi, C. R., Yi, L., Su, H., & Guibas, L. J. 2017b, Proceedings of Advances in Neural Information Processing Systems (NeuralIPS)

Qian, Y., Hou, J., Zhang, Q., Zeng, Y., Kwong, S., & He, Y. 2020, arXiv:2005.00383

Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., & Shlens, J. 2019, Advances in neural information processing systems, 32

Ribeiro, L. F., Saverese, P. H., & Figueiredo, D. R. 2017, in Knowledge Discovery and Data Mining(KDD)

Rong, Y., Huang, W., Xu, T., & Huang, J. 2020, in International Conference on Learning Representations (ICLR)

Sarode, V., Li, X., Goforth, H., Aoki, Yasuhiro Srivatsan, R. A., Lucey, S., & Choset, H. 2019, arXiv preprint arXiv:1908.07906

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. 2009, IEEE Transactions on Neural Networks, 20, 61

Sen, P., Namata, G. M., Bilgic, M., Getoor, L., Gallagher, B., & Eliassi-Rad, T. 2008, AI Magazine, 29, 93

Shchur, O., Mumme, M., Bojchevski, A., & Gunnemann, S. 2018, in NeurIPS Workshop on Relational Representation Learning

Simony, M., Milzy, S., Amendey, K., & Gross, H.-M. 2018, in Proceedings of the European Conference on Computer Vision (ECCV) Workshops

Song, Y., & Ermon, S. 2019, in Neural Information Processing Systems (NeurIPS)

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. 2020, in International Conference on Learning Representations

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, Journal of Machine Learning Research, 15, 1929

Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.-H., & Kautz, J. 2018, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2530

Sun, Y., Wang, Y., Liu, Z., Siegel, J., & Sarma, S. 2020, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 61–70

Swersky, K., Buchman, D., Freitas, N. D., Marlin, B. M., et al. 2011, in International Conference on Machine Learning (ICML)

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. 2015, in International World Wide Web Conference (WWW)

Tang, J., Sun, J., Wang, C., & Yang, Z. 2009, in Knowledge Discovery and Data Mining(KDD)

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., & Guibas, L. J. 2019, Proceedings of the IEEE International Conference on Computer Vision (ICCV)

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., & Sohl-Dickstein, J. 2017, in Advances in Neural Information Processing Systems (NIPS)

Uy, M. A., Pham, Q.-H., Hua, B.-S., Nguyen, D. T., & Yeung, S.-K. 2019, in International Conference on Computer Vision (ICCV)

Van den Berg, R., Kipf, T. N., & Welling, M. 2017, arXiv preprint arXiv:1706.02263

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. 2017, in Advances in Neural Information Processing Systems (NIPS)

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. 2018, in International Conference on Learning Representations (ICLR)

Wang, M. et al. 2019a, in ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds

Wang, W., Yu, R., Huang, Q., & Neumann, U. 2018, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2569

Wang, X., Jin, Y., Cen, Y., Lang, C., & Li, Y. 2021, in Image and Graphics: 11th International Conference, ICIG 2021, Haikou, China, August 6–8, 2021, Proceedings, Part III 11, Springer, 57–69

Wang, X., Jin, Y., Cen, Y., Wang, T., Tang, B., & Li, Y. 2022, arXiv preprint arXiv:2202.06263

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. 2019b, ACM Transactions on Graphics (TOG)

Wei, C., Kakade, S., & Ma, T. 2020, in International Conference on Machine Learning (ICML)

Welling, M., & Teh, Y. W. 2011, in International Conference on Machine Learning (ICML)

Williams, R. J. 1992, Machine Learning, 8, 229

Wu, F., Fan, A., Baevski, A., Dauphin, Y. N., & Auli, M. 2019a, arXiv preprint arXiv:1901.10430

Wu, J., Zhang, C., Xue, T., Freeman, B., & Tenenbaum, J. 2016, Advances in neural information processing systems, 29

Wu, M., Pan, S., Zhou, C., Chang, X., & Zhu, X. 2020, in WWW '20: The Web Conference

Wu, W., Qi, Z., & Fuxin, L. 2019b, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 9622

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. 2015a, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Los Alamitos, CA, USA: IEEE Computer Society), 1912–1920

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. 2015b, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1912

Xie, J., Lu, Y., Gao, R., Zhu, S.-C., & Wu, Y. N. 2020a, IEEE Transactions on Pattern Analysis and Machine Intelligence

Xie, J., Lu, Y., Zhu, S.-C., & Wu, Y. 2016, in International Conference on Machine Learning (ICML)

Xie, J., Xu, Y., Zheng, Z., Zhu, S.-C., & Wu, Y. N. 2021a, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 14976–14985

Xie, J., Zheng, Z., Fang, X., Zhu, S.-C., & Wu, Y. N. 2021b, IEEE Transactions on Pattern Analysis and Machine Intelligence

Xie, J., Zheng, Z., Gao, R., Wang, W., Zhu, S.-C., & Wu, Y. N. 2018, in Proceedings of the IEEE conference on computer vision and pattern recognition, 8629–8638

Xie, J., Zheng, Z., Gao, R., Wang, W., Zhu, S.-C., & Wu, Y. N. 2020b, IEEE Transactions on Pattern Analysis and Machine Intelligence

Xie, J., Zhu, S.-C., & Nian Wu, Y. 2017, in Proceedings of the ieee conference on computer vision and pattern recognition, 7093–7101

Xie, J., Zhu, S.-C., & Wu, Y. N. 2019, IEEE transactions on pattern analysis and machine intelligence, 43, 516

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., & Bengio, Y. 2015a, in Proceedings of the 32nd International Conference on Machine Learning

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. 2015b, in International Conference on Machine Learning (ICML)

Yang, G., Huang, X., Hao, Z., Liu, M.-Y., Belongie, S., & Hariharan, B. 2019a, Proceedings of the IEEE International Conference on Computer Vision (ICCV)

Yang, X., & Ji, S. 2021, in International Conference on Computer Vision (ICCV)

Yang, Y., Feng, C., Shen, Y., & Tian, D. 2018, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 206

Yang, Z., Cohen, W. W., & Salakhutdinov, R. 2016, in International Conference on Machine Learning (ICML)

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. 2019b, Advances in neural information processing systems, 32

Ye, Y., Yang, X., & Ji, S. 2022, arXiv preprint arXiv:2210.05638

Yu, L., Li, X., Fu, C.-W., Cohen-Or, D., & Heng, P. A. 2018, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2790

Zachary, W. W. 1977, Journal of Anthropological Research, 33, 452

——. 2002, Physical review letters, 89, 208701

Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., & Ma, K. 2019, Proceedings of the International Conference on Computer Vision (ICCV)

Zhao, H., Jia, J., & Koltun, V. 2020, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 10076–10085

Zhao, H., Jiang, L., Jia, J., Torr, P. H., & Koltun, V. 2021a, in Proceedings of the IEEE/CVF international conference on computer vision, 16259–16268

Zhao, Y., Birdal, T., Deng, H., & Tombari, F. 2019, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1009

Zhao, Y., Xie, J., & Li, P. 2021b, in International Conference on Learning Representations ICLR

Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., & Wang, W. 2020, in Proceedings of the 37th International Conference on Machine Learning

Zhu, S. C., Wu, Y., & Mumford, D. 1998, International Journal of Computer Vision, 27, 107

Zitnik, M., & Leskovec, J. 2017, Bioinformatics, 33, 190