Homework #3: Structured 3D Shape Generative Models [100 points]
Due Date: Wednesday, 17 May 2023

Goals of This Assignment

This assignment is a two-week assignment on shape generation from structured representations. The learning objectives of this assignment are to:

- get familiar with representing 3D shapes with parts and structure;
- implement and train a structured generative model.

Instructions

This assignment contains three programming problems. It requires decent CPUs and one GPU to train neural networks over big scale 3D data. Please use Google CoLab (check https://www.youtube.com/watch?v=IZUz4pRYlus for instructions) or the Cloud Credits to access a GPU. All the code you need to start this assignment will be found in the code .zip. Please read the `README.md` for setup instructions.

We use Python 3.6.9 and PyTorch 1.5.1 for the codebase, but it should be nearly issue-free, though not guaranteed, if you use more latest versions. We recommend you start as early as possible and definitely should not to wait until the last day or two to begin as the network training may take time. Please contact TA or ask on Edstem if you have any question.

Problem 1. Structured Representations for 3D Shapes [20 points]

The space of 3D data is exceptionally rich and diverse. It is a very difficult task to learn generative models that can generate everything in our 3D world. Fortunately, 3D shapes and scenes can be naturally decomposed into small components, which are usually much simpler to model and largely reusable across different data. Each 3D shape and scene is then structurally organized as a composition of such sub-units in some semantically meaningful ways. For example, in 3D objects, this could be a decomposition of an object into spatially localized parts (e.g., chair is composed of a back, a seat, and four legs) and a sparse set of relationships between them (e.g., chair legs are of equal length and symmetric), or in scenes, it could be a scene graph with the 3D objects as the nodes and their rich inter-object relationships as the edges (e.g., adjacency, co-occurrence, supporting). For the 3D data, compositionality and structure can also enable compact representations, which has become key not only for vision tasks, but also for developing advanced 3D generative models.

In this problem, we will get you familiar with structured 3D shape representations. Specifically, consider a family of shapes from the same object category (e.g., chairs), we can break

all the different shapes into semantically consistent parts where each part has more easy geometry to generate. There are also relationships among parts, such as two parts are adjacent or symmetric (e.g., the chair back and seat are adjacent, legs are symmetric to each other).

We use the PartNet dataset [2] for this assignment, where each shape is decomposed into a set of parts and is organized using a part hierarchy representing parts at different segmentation granularity. We provide a web page to visualize the part hierarchy for each shape. See https://partnet.cs.stanford.edu/visu_htmls/2230/tree_hier_after_merging for the visualisation for shape ID 2230. You may modify the ID in the link to other shape IDs to check other shapes. We have provided 8176 PartNet chairs as the data for you to train structured generative models for 3D shapes. Each data composes of a JSON file defining the part bounding boxes, semantics, hierarchical structures, and the rich part relationships.

Before training over these data, let us get more understanding over the data.

(a) (10 points). We use a part hierarchy to represent the structure of a shape. Among these parts, there are vertical relationships indicating that one part can be further decomposed into children subparts, and horizontal ones among siblings describing adjacency and symmetry relationships. The provided JSON file encodes these information. You may use some JSON visualization tools, such as https://jsonformatter.org/json-viewer, to visualize and help you understand the JSON files.

Let us implement some utility functions to print the tree structure. In the code, we use the `Tree` and `Node` classes in the `data.py` file to load the input JSON data. Read the code and finish the implementation of `Node._to_str` function. Then, you should be able to print out a simple hierarchy structure by calling `print(object)` in `prob1.ipynb`. For simplicity, you do not need to worry about the horizontal edges.

Deliveriables: submit the code between `STUDENT CODE START` and `STUDENT CODE END` in your solution PDF. And, for shapes with the ID `2230`/`2231`/`2233`, take screenshots and submit the printed outputs for the tree structure.

(b) (10 points). We can assemble the geometry of parts to obtain the final shape geometry. The JSON file defines a bounding box for each part. Understand the provided data and implement the `Node.get_part_hierarchy` function to assemble the bounding boxes of leaf-level nodes to form the final shape. Then, you can call the `draw_partnet_objects` function in `prob1.ipynb` to visualize the assembled shapes.

Deliverables: submit the code between `STUDENT CODE START` and `STUDENT CODE END` in your solution PDF; for the shape ID `2230`/`2231`/`2233`, take screenshots and submit the visualization for the assembled shapes.

Problem 2. Structured Generative Networks for 3D shapes [80 points]

StructureNet [1] introduces a way to learn structured generative models for 3D shapes using the PartNet [2] part tree representation. It employs a Recursive Neural Network (RvNN) to encode and decode the input part hierarchy and trains a VAE for obtaining a generative model.

In this problem, you are given the codebase implementing the StructureNet system. Since different shapes have different part hierarchical structures, it is a bit difficult to form batches for GPU training, which is a common issue for such RvNN architecture. Therefore, we use batch size 1 and require modern CPUs, such as Intel i5/i7/i9 CPUs, for the training. GPU is also used for speeding up the matrix computation, but you will see that most jobs are done with batch size 1. We do not consider the horizontal edges in this problem. Also, we only train the StructureNet models with bounding boxes as the part geometry.

(a) (50 points). We have provided the codebase for data loading, training, and most parts of the model. Read the code and finish the implementation for the encoder and decoder parts, namely, the `SymmetricChildEncoder.forward`, `ConcatChildDecoder.forward` and `RecursiveDecoder.node_recon_loss` functions. Then, train a StructureNet VAE network for **10 epochs (roughly 2-3 hours)**. During training, you will see a few results on the validation set being dumped to the `val_visu` directory. If you have implemented the code correctly for problem 1, the code will output the ground-truth and predicted part hierarchy and assembled boxes at the leaf level, after each training epoch.

Deliverables: submit the three pieces of code between `STUDENT CODE START` and `STUDENT CODE END` for the encoder and decoder of StructureNet in your solution PDF; submit the training and validation curves; and, show the ground-truth and your VAE reconstructed results for five structurally different shapes.

(b) (10 points). Unfortunately, the StructureNet VAE model takes 1-3 days of training until converging to good results. So, we provide our pre-trained models `pretrained_encoder.ckpt` and `pretrained_decoder.ckpt` (trained for 143 epochs and 30 hours). Implement `recon.py` and report the reconstruction performances of the pretrained models.

Deliverables: submit the code between `STUDENT CODE START` and `STUDENT CODE END`; pick five shapes in different structures and show the ground-truth and your VAE reconstruction result (show three good ones and two bad ones); and, write a few sentences what you find about the results (e.g., in what cases the network performs well and in what cases it does not? what are the possible reasons? do you have any idea to further improve the results?).

(c) (10 points). The trained StructureNet VAE is a generative model, so we can sample random Gaussian noises and use the decoder to generate novel 3D structured shapes. Implement `randgen.py` and use the pretrained mdoels to generate several shapes using your model.

Deliverables: submit the code between `STUDENT CODE START` and `STUDENT CODE END`; visualize five shapes you randomly generate (show three good ones and two bad ones); and, write a few sentences what you find.

(d) (10 points). The StructureNet VAE also allows us to interpolate between two shapes over the learned latent space. The interpolation involves both discrete structural changes and concrete geometry deformations. Using the pretrained StructureNet VAE models, please finish the implementation in `interp.py` and perform part-aware shape interpolation a given pair of shapes. Basically, you simply need to encode the two shapes, obtain their latent codes, perform linear interpolations between the codes, and decode the interpolated shapes as the outputs.

Deliverables: submit the code between `STUDENT CODE START` and `STUDENT CODE END`; submit the interpolation results for pairs `(1282, 721)`, `(2333, 2436)`, and `(2368, 2821)`; and, write a few sentences about your findings (e.g. which cases work well and which work badly? why? any ideas to improve?).

References

[1] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy J Mitra, and Leonidas J Guibas. Structurenet: hierarchical graph networks for 3d shape generation. *ACM Transactions on Graphics*, 38(6):1–19, 2019.

[2] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 909–918, 2019.