Homework #2: Neural Implicit 3D Representations [100 points]
Due Date: Wednesday, 3 May 2023

Goal of this Assignment

This assignment is a two-week project on neural implicit representations for 3D shape modelling. The goal of this assignment is to:

- Familiarize with neural implicit 3D representations.
- Use neural implicit representations to represent 3D shapes for shape reconstruction and completion.

Instructions

This assignment contains three programming problems that require access to a machine with decent CPUs and one GPU to train the neural networks on big scale 3D data. Please use Google CoLab (check `https://www.youtube.com/watch?v=IZUz4pRYlus` for instructions) or the Cloud Credits (check `https://github.com/cs231n/gcloud` for instructions; ask the TAs for coupons) to get access to a GPU. All necessary files for this assignment can be found in the folder `code`. Please follow the instructions in the `README.md`, provided in the `code` folder, to install all necessary dependencies for running the provided codes. Please make sure to start as early as possible and definitely not wait until the last days to begin, as the network training will take some time.

For this exercise you need to download a subset of 100 Shapenet Chairs from this link and add them within the `data` folder. In case the data folder was renamed, please rename it to 03001627.

Problem 1. Representing a Single Scene with a NIR [50 points]

Existing 3D representations can be categorized to explicit representations such as voxels, pointclouds and meshes and implicit representations that capture the 3D object geometry implicitly in the weights of a neural network. Voxel-based representations are a straightforward generalization of pixels in 3D, as they directly discretize the 3D space into a regular grid. However, their high memory footprint that grows cubically with the resolution size makes them impractical for several applications. Pointclouds are more memory efficient but lack surface connectivity, thus post-processing is necessary for extracting the final 3D mesh from the model. On the other hand, mesh-based representations naturally yield smooth reconstructions but as they often require a deformable template mesh or represent the geometry as an atlas of multiple mapping, they cannot capture arbitrary topologies. To address these limitations, implicit representations have recently gained more popularity. These representations capture the 3D shape geometry as the level-set of a distance or occupancy field implemented as a neural network, that takes a context vector and a query point and predicts either a signed distance value [3] or a binary occupancy value [2, 1] for the query point.

For the case of occupancy fields, we define the 3D shape geometry as the decision boundary of a binary classifier. The network takes a query 3D position $\mathbf{x} \in \mathbb{R}^3$ as input and classifies whether this point is inside or outside of a watertight mesh, namely a mesh without holes. Similarly, one can consider modeling a signed distance field (SDF) whose sign indicates occupancy, and whose magnitude describes the distance to its closest surface. For SDFs the surface is implicitly represented as the zero-level-set, where the sign flips. In the first problem, we will learn how to represent shapes using SDF-based implicit representations.

(a) (15 points) We approximate the shape's SDF at all locations in a target volume $\Omega \subset \mathbb{R}^3$, using a neural network $f_\theta(\cdot)$ with parameters θ, as follows:

$$f_\theta(\mathbf{x}) \approx SDF(\mathbf{x}), \forall \mathbf{x} \in \Omega. \tag{1}$$

For the first assignment, we have prepared a dataset X composed of 3D locations $\mathbf{x}$ and their precomputed SDFs s:

$$X := \{(\mathbf{x}, s) : SDF(\mathbf{x}) = s\}. \tag{2}$$

Using this dataset, we want to optimize the network parameters of f_θ to approximate the SDF using the sampled points of X, by minimizing the loss:

$$\mathscr{L}(\mathbf{f}_\theta, x, s) = |\mathbf{f}(x) - clamp(s, \delta)|, \tag{3}$$

where $clamp(s, \delta) := \min(\delta, \max(-\delta, x))$ clamps the SDF absolute value to be within $\delta = 0.1$, because we are more interested in SDF values near zero-crossing.

For the first question, your task is to complete the missing code in function `train_epoch()` in the `single_scene.py` script that computes the loss of Eq. 3. In your report, please provide your loss implementation and also attach the loss curve, along with the visualization of the cross section. Note that both images are automatically generated by the script.

(b) (25 points) In this question, instead of directly training from precomputed SDFs, we can solve for a differential equation to indirectly learn SDF only from the surface points and their surface normals. As we explained before, in the case of SDFs, the surface is defined as the zero-level-set, thus SDFs at the surface points are equal to 0. Remember the definition of SDF: distance to the closest surface boundary (sign indicates inside-outside state). The gradient of SDF at a point $\mathbf{x}$, from gradient definition of fastest changing direction, is directed away from the closest surface point in surface normal direction. Therefore, when $\mathbf{x}$ is a surface point $\nabla_x f_\theta(\mathbf{x}) = \mathbf{n}(\mathbf{x})$, where $\mathbf{n}$ is surface normal. For non-surface points, the derivative $\nabla_x f_\theta(\mathbf{x})$ can be expressed as the direction of the fastest changing SDF, which has magnitude of 1 for all locations:

$$|| \nabla_x f_\theta(\mathbf{x}) ||_2 = 1, \tag{4}$$

Your task is to add the missing code in function `train()` in the `Eikonal/train.py` script, that implements the loss function to train the network to fit to the bunny model from the provided oriented point cloud. The overall loss function that you need to implement is:

$$\sum_{\mathbf{x}\in S} |f_\theta(\mathbf{x})| + ||\nabla_x f_\theta(\mathbf{x}) - \mathbf{n}_x|| + \sum_{\mathbf{x}\in R} |||\nabla_x f_\theta(\mathbf{x})|| - 1|, \tag{5}$$

where S is the provided surface points and $R \subset \Omega$ is a set of randomly selected points within the target volume Ω. For simplicity, you do not need to implement the normal loss and only need to implement the first and the third term of Eq. 5. In your report, please provide your loss implementation and also attach the loss curves generated by the script.

Hint: Use the following automatic gradient computation from PyTorch to compute gradient of SDF with respect to a 3D point $\mathbf{x}$.

```python
import torch.autograd as autograd
g = autograd.grad(
    outputs=SDF,
    inputs=xyz,
    grad_outputs=torch.ones(SDF.size()).to(device),
    create_graph=True,
    retain_graph=True,
    only_inputs=True
)[0]
```

(c) (10 points) To extract a mesh from an SDF, we use an algorithm called Marching Cubes (https://en.wikipedia.org/wiki/Marching_cubes). Your task is to complete the function `compute_SDFs()` from the `Eikonal/marching_cubes.py` script to evaluate your network $\mathbf{f}_\theta(\cdot)$ at points within a grid of resolution 128 in our target volume (ranges from -1.5 to 1.5). In your report, please provide yoru marching cubes implementation and also show the bunny rendering using the model that you trained for the previous question.

Problem 2. Training a SDF Generative Model [50 points]

In this problem, instead of focusing on a single shape, we will implement a generative model using SDFs, using and auto-decoder architecture. An auto-decoder is an encoder-less auto-encoder. Namely, an auto-decoder takes as input a per-instance feature vector, as opposed to raw data such as pointclouds or voxels. To generate the per-instance feature vectors, we associate each sample in the training set with a random latent code, namely, assuming our training set consists of 1000 shapes, we will associate each shape with a single latent vector. At training time, the auto-decoder jointly optimizes the latent vectors and the decoder network weights to minimize the reconstruction error of corresponding data points. The advantage of an auto-decoder algorithm is that we do not need to devise an encoder network, as we are directly optimizing the latent vectors.

(a) (25 points) In this problem you need to add the missing code in function `train_epoch()` of the `decoder_deepsdf.py` script that is used for training an autode-coder, by minimizing the following loss:

$$\texttt{argmin}_{\theta,\{z_i\}} \sum_{i=1}^{N} \left(\sum_{j=1}^{K} \mathscr{L}(f_\theta, z_i, \mathbf{x}_j, s_j) + \frac{1}{\sigma^2}||z_i||^2 \right), \tag{6}$$

where the second term regularizes the latent vectors z_i to follow a zero-mean Gaussian. The loss function $\mathscr{L} := |f_\theta(\mathbf{x}_i, z_i) - clamp(s_i, \delta)|$. Run the above optimization step for 1,000 iterations. In your report, please provide your loss implementation and also visualize of the first code z_1 that is generated, as you run the code.

(b) (25 points) Using our generative model, now we want to try to generate novel shapes conditioned on partial or complete point clouds of shapes, not seen during training. To do this, we try to find a latent code that produces SDF most similar to the given shape via gradient-based optimization, typically referred to as test time optimization. Since we trained our generative model using only 100 shapes, the learned latent space might not be rich enough, therefore we provide a pre-trained model (`weights.pth`) that is trained with 600 chairs. With the oriented point clouds from the `data/point_test` folder, use the differential equation loss from Problem 1 (c), to conduct the following optimization:

$$\hat{z} = \texttt{argmin}_z \sum_{\mathbf{x}\in S} |f_\theta(\mathbf{x}, z)| + ||\nabla_x f_\theta(\mathbf{x}, z) - \mathbf{n}_x|| + \sum_{\mathbf{x}\in R} |||\nabla_x f_\theta(\mathbf{x}, z)|| - 1| + \frac{1}{\sigma^2}||z||^2, \tag{7}$$

where S is the provided depth points, and R is randomly sampled points within the target volume. Note that the decoder network weights are fixed and only the latent code is optimized. For this task, you need to complete the missing code in function `train_epoch()` in the `shape_completion_deepsdf.py` script. In your report, please provide your implementation and also show the shape completion results on the two point clouds as well as the extracted meshes.

References

[1] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019.

[2] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[3] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.