Paper Summary Deep SDF

Abstract

Deep SDF enables high quality shape reconstruction, interpolation, and completion from **partial** and **noisy** 3D input data. Implicitly encodes a shape's boundary as the zero-level-set of the learned function.

1. Introduction

Fully continuous. Generative. Learned shape-conditioned classifier for which the decision boundary is the surface of the shape itself. Contribution:

1. The formulation of generative shape-conditioned 3D modeling with a continuous implicit surface.
2. $*$ A learning method for 3D shapes based on a probabilistic auto-decoder.
3. Shape modeling and completion.

2.2 Representation Learning Techniques

Optimizing Latent Vectors. An alternative is to learn compact data representations by training *decoder-only* networks. Simultaneously optimizes the latent vectors assigned to each point and the decoder weights through back-propagation. Auto-decoders are trained with *self-reconstruction* loss on decoder-only architecture.

3. Modeling SDFs with Neural Networks

Describe modeling shapes as the zero iso-surface decision boundaries of feed-forward networks trained to represent SDFs. Continuous function that, for a given spatial point, outputs the point's distance to the closet surface, whose sign encodes whether the point is inside or outside of the watertight surface:

$$SDF(x) = s : \mathbf{x} \in R^3, s \in R$$

Key idea is to directly regress the continuous SDF from point samples using DNN. Such surface representation can be intuitively understood as a learned **binary classifier** for which the decision boundary is the surface of the shape itself. Given a target shape, we prepare a set of pairs X composed of 3D point samples and their SDF values:

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

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

where Ω is target domain. Training is done by minimizing the sum over losses b/t the predicted and real SDF values of points in X under the following L_1 loss function:

$$L(f_\theta(\mathbf{x}), s) = |clamp(f_\theta(\mathbf{x}), \delta) - clamp(s, \delta)|$$

where δ control the distance from the surface over which we expect to maintain a metric SDF.
larger values of δ allow for fast ray-tracing since each sample gives information to safe step sizes.
smaller values of δ can be used to concentrate network capacity on details near the surface.

Accurate normals can be analytically computed by calculating the spatial derivative $\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}}$ via back-propogation through the network.

4.2 Auto-decoder-based DeepSDF Formulation

Given a dataset of N shapes represented with signed distance function $SDF^i_{i=1\ldots N}$, we prepare a set of K point samples and their signed distance values:

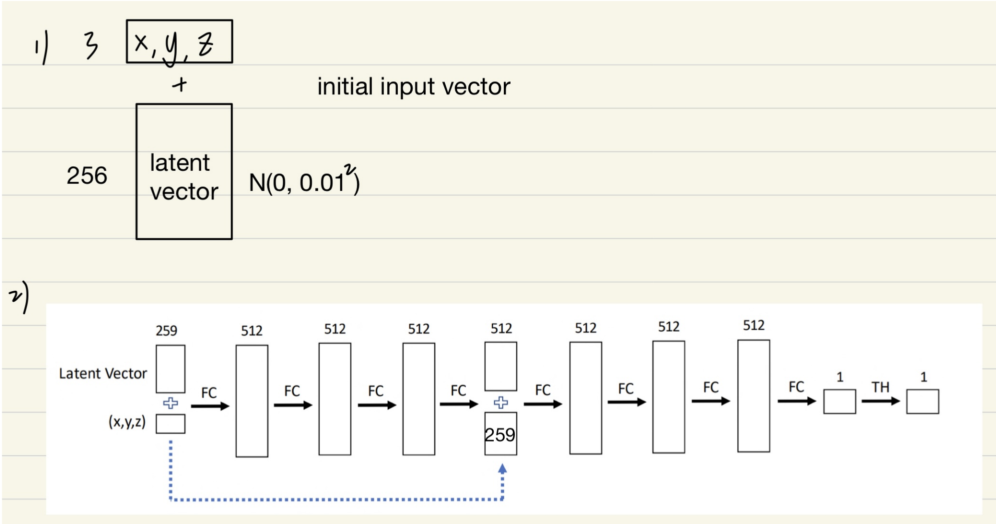
$$X_i = \{(\mathbf{x}_j, s_j) : s_j = SDF^i(\mathbf{x}_j)\}$$

For an auto-decoder, as there is no encoder, each latent code z_i is paired with training shape X_i
Introducing a latent vector z, which can be though of as encoding the desired shape, as a second input to the neural network. Conceptually, mapping this latent vector to a 3D shape represented by a continuous SDF.

$$f_\theta(z_i, \mathbf{x}) \approx SDF^i(\mathbf{x})$$

By conditioning the network output on a **latent vector**, this formulation allows modeling multiple SDFs with a single NN.

Training Pipeline



We train both latent vector z_i and whole decoder θ (the coefficient values in neurons). Recall the training loss function:

$$\arg\min_{\theta,\{z_i\}_{i=1}^{N}} \sum_{i=1}^{N} \left(\sum_{j=1}^{K} \mathbf{L}(f_\theta(z_i, \mathbf{x}_j), s_j) + \frac{1}{\sigma^2}||z_i||_2^2 \right)$$

We need to find two args: θ and z_i, where θ is the parameters for decoder, and z_i for $i = 1...N$ is the latent vector for each shape like car and airplane. Each shape in training dataset X_i has its corresponding z_i (need to be found by training). Operations above are all done during the training

After training this loss function, we will get parameter θ for decoder f_θ and latent vector z_i for each shape in the training dataset. Our decoder f_θ will be generalized to fit for most shapes, and latent vector z_i will work for certain shape in training set.

Question: what if we have new shape (or how is our function f_θ work for **test set**). Let's say a new car in the test set but not including in the training set. Inference:

Fixing parameter θ for decoder, and retrain the latent vector z_{N+1} by the given sample points in $X_{new-car}$.

Overall, in training set, we use single neural network to train two parameters θ and z_i.
How to achieve that: By inserting vector $z_i + \mathbf{x}_j$ into the intermediate layer of training network, which means our input latent vector z_i also become a part of network.

Commonly, during the training, we only update the network θ, but here we also update input z_i. It's like:

$$z_i^{j+1} = f_\theta^j\ (z_i^j,\ \mathbf{x}_j)$$

$$f_\theta^{j+1} = f_\theta^j\ (z_i^j,\ \mathbf{x}_j)\ \text{ doing back-propogation}$$

where $z_i^0$is randomly defined by $N(0, 0.01^2)$, i is the index of shapes in training dataset, and j is the index of sample points for each shape X_i.