Generative models for shape parametrization

Guglielmo Padula

University of Trieste

General Objective

The general objective of our research is, given a set of 3D meshes generated by complex methods with a large number of parameters, to find a generative model such that:

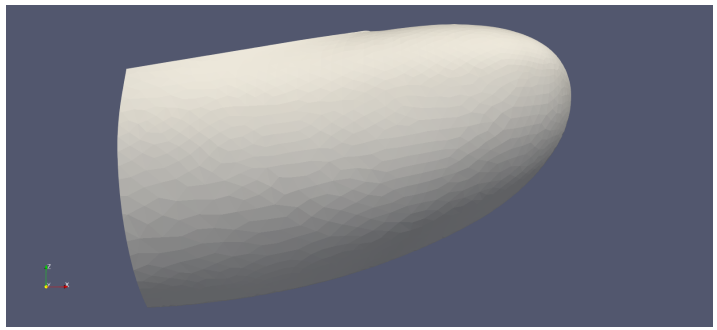
- Reduces the number of effective parameters via dimensionality reduction
- Sampling (and if possible training) using this method should be fast as possible
- Generated samples should satisfy some **convex** equality and/or inequality constraint
- (**Optional**) the distribution of useful quantities associated to the object should be similar between the generated samples and the real samples.

Motivation

- Complex methods are very computationally complex (the data generated in the code uses VFFD which is fast, howewer there are methods based on differential equations that take days to complete)
- A possible application are inverse problems with non convex constraints, which require exaustive search, they would scale exponentially in the number of parameters.
- Ideally we would like to run simulation on this samples, and indeed to get not this simulations to explode, we need some convex constraints on the output.

Data

We focus on the problem of deforming a naval hull bulb. The data is obtained by deforming using Volume Preserving FFD with 27 parameters, which deforms object by moving control points and preserves the volume. We want volume preservation because it is a nice property in simulations. It can be rewritten as a sequence of linear constraints. As we want to focus only on the bulb, we first take only one half of the ship, and then we do two cuts to obtain the bulb. The ship is a triangular mesh so it has an internal topology.



The generative models

Our generative model should not touch the points on the boundary and move the points on the symmetry plane only horizontally. It should also preserve volume exactly. For this reason we need to heavely customize classic generative models and we split the data in two parts, a part (x) that will consist in the points that don't belong to the simmetry plane and a part y that belong to the simmetry plane. We use:

- Vanilla Autoencoder
- Variational Autoencoder
- Boundary Equilibrium GAN
- Adversarial Autoencoder

All this models require an Encoder and a Decoder, whose architectures will be the same in every model. All models are trained using AdamW.

Long life to the LBR

Our basic layer (which we call LBR) is formed with:

- A linear Layer
- A BatchNorm1D Layer
- A ReLU Layer
- A Dropout Layer

Custom Layers

We also use a series of custom layer to improve the training and the sampling:

- A PCA Layer
- A smoothing Layer
- A volume normalization Layer

PCA Layer

We use PCA to reduce the initial dimensionality of the data, which will consist in two layers: a PCATranformLayer and a PCAInverseTransformLayer. Note that PCA does not compress data as the relative reconstruction error is very low (around 10^{-8}).

Smoothing Layer

This is a layer that applies laplacian smoothing

$$K_i = \frac{1}{|N_i|} \sum_{j \in N_i} K_j$$

where N_i are the neighbours of index i. Note that while K_i. Note that the laplacian smoother is applied only to points that are not on the boundary and on the simmetry plane.

Volume Normalization

The Volume Normalization layer solves the optimization problem of finding the lowest variation of the points such that the volume is constant and such that the points don't clash with the boundary. This layer is well behaved because both the constrains are linear and so the solution is unique. Notice that because the problem is quadratic programming problem, a solution is find very quickly and so it does not slow the training.

Encoder

The Encoder takes as input x and y and applies for x:

- A PCATransform from the dimension of x (around 700) to 24.
- A LBR from 24 to 500
- A series of LBR with hidden dim 500
- A final LBR from 500 to the latent space of x which has dimension 11 (estimated with TwoNN).
- The output is standardized.

The same is done with y (original dimension around 200, latent space 1, hidden dim 500).

Decoder

The Decoder takes as input the z of size 11 and w of size and applies for z:

- An LBR from 11 to 500
- A series of LBR with hidden dim 500
- A final LBR from 500 to the latent space of 24
- Then PCAInverseTransform is applied

The same is done with w. Then the two ouputs are concatenated and

- Smoother is applied
- VolumeNormalizer is applied

Vanilla Autoencoder

This is a classic autoencoder, customized for having two inputs and two outputs.

$$z, w = Enc(x, y)$$

$$\hat{x}, \hat{y} = Dec(z, w)$$

$$Loss = \frac{1}{2}(||x - \hat{x}||_2^2 + ||y - \hat{y}||_2^2)$$

Sampling is done by sampling from $Normal(0, 1)$ and then by applying the Decoder. This brings useful results because of the standardization in the Encoder.

Variational Autoencoder

$$\mu_1, \mu_2 = Enc(x, y)$$

$$\sigma_1 = LBR(\mu_1)$$

$$\sigma_2 = LBR(\mu_2)$$

$$z \sim Normal(\mu_1, \sigma_1)$$

$$w \sim Normal(\mu_2, \sigma_2)$$

$$\hat{x}, \hat{y} = Dec(z, w)$$

$$A = Normal(\hat{x}, \sigma)$$

$$B = Normal(\hat{y}, \sigma)$$

where σ is a network parameter.
$Loss = \frac{1}{2}(p_A(x)) + p_B(y)) + \frac{1}{2}(KL(Normal(0,1)|Normal(\mu_1, \sigma_1)) + KL(Normal(0,1)|Normal(\mu_2, \sigma_2)))$