4 Generative Neural Networks - Diffusion Models

Diffusion probabilistic models have recently become quite popular in Computer Vision due to their ability to generate high-resolution realistic images. In this project, you will be diving deep into the actual working of such models. You will work your way through the math involved and the tricks used to train neural network models to accomplish the generative task using this method. Due to limited data and computation available, you will be working with simple toy data.

Notes

1. It is recommended to use PyTorch for mandatory part-2 and the advanced part, as there is much larger set of functionalities, documentation and support in PyTorch for the task of training neural networks than MATLAB. Also, the help from TAs on implementing this in MATLAB would be quite limited.
2. blog 1 and blog 2 are excellent sources that can serve as tutorials for the intrinsics of diffusion models.

4.1 Introduction

As discussed in the lectures, a (denoising) diffusion model consists of 2 parts:

1. A pre-defined forward diffusion process q that gradually adds small amounts of Gaussian noise to the original data, until you end up with pure noise.
2. A learned reverse (denoising) model p_θ that removes the noise to eventually produce something that hopefully looks like your original data.

Assuming a data point to have been sampled from an original distribution $\mathbf{x}_0 \sim q(\mathbf{x})$. We sequentially add Gaussian noise to it at timesteps $1 \dots T$ as per the conditional distribution:

$$q(\mathbf{x_t}|\mathbf{x_{t-1}}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta\mathbf{I}) \tag{1}$$

where the step sizes are controlled by the variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$.

4.1.1 Mandatory task 1

If somehow the reverse process $q(\mathbf{x_{t-1}}|\mathbf{x_t})$ was available, one would be able to produce meaningful data by repeatedly applying this denoising process starting from pure noise. Here we will consider a simple 1D case: $\mathbf{x_0} \sim \mathcal{N}(\mathbf{x_0}; m, p)$ and the forward diffusion process as mentioned above in (1). Assuming that the distribution of $\mathbf{x_0}$ is known, the reverse conditional distribution $q(\mathbf{x_{t-1}}|\mathbf{x_t}, \mathbf{x_0})$ can be computed in closed form, using the Bayes rule and the product of Gaussian distributions.

- Choose your own parameters (m, p) for the distribution of $\mathbf{x_0}$ and sample from this.
- Run the forward diffusion process for a number of timesteps, $T = 50$. You can experiment with different variance schedules (eg. linear, quadratic, etc.), but for simplicity, you can start with a constant schedule such that $\beta_t = 0.25\ \forall t \in \{1 \dots T\}$. By repeating this process for various inputs, verify that the values at time T approximately follow a standard normal distribution. Visualize a few forward trajectories and also the distribution of x_T computed over at least 100 trajectories starting from different realizations of $\mathbf{x}_0$ by plotting x_t against t for $t \in (1 \dots T)$.

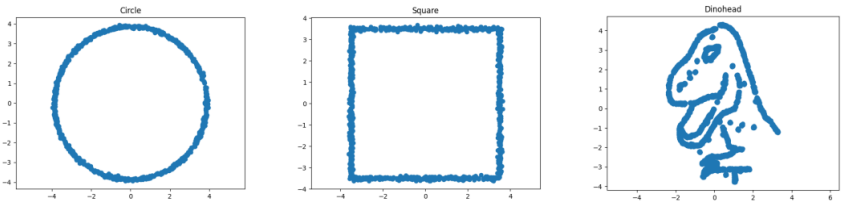

Figure 6: 2D point cloud datasets for mandatory task 2 - *Circle*, *Square* and *Dinohead*

- Compute the reverse process as a function of $\mathbf{x_t}$, $\mathbf{x_0}$ and t. You can take help from the derivations mentioned here.
- Sample $\mathbf{x_T}$ from a standard normal distribution and repeatedly apply the reverse process computed above to obtain $\mathbf{x_0}$. Visualize a few of the reverse trajectories.

4.1.2 Mandatory task 2

In this task you will actually predict the $q(\mathbf{x_{t-1}}|\mathbf{x_t})$ using a neural network with parameters θ. Hence this prediction/appoximation will be termed as $p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})$. To learn this, you will closely follow the approach presented in [3]. You are provided sets of 2D points in the following shapes - a *Circle*, a *Square*, and *Dinohead* - an outline of a (rather simplistic) dinosaur head (Fig. 6). The task is to learn to produce something close to these starting from standard Gaussian noise. You are supposed to train a different model for each shape. This is not a great example to showcase the generative ability of diffusion methods, however, it is interesting in the sense that it allows understanding and implementing the intrinsic details of the method.

Use the following steps as guidelines to implement a training regime that produces the required output. While training the model for a particular shape example, do the following within each training loop:

- Sample a batch (randomly selected small subset) of points and for each point - a timestep, and noise - this is the true noise.
- Predict the 2D noise $\epsilon_\theta(\mathbf{x_t}, t)$ using $\mathbf{x_t}$ and t as inputs.
- Using squared error between the predicted and true noise as the loss, back-propagate and update the model parameters using gradient descent.

Other general tips:

- Instead of feeding the 2D points and the timesteps as inputs to the neural network directly, it is advisable to feed in *positional embeddings* of these inputs. High-frequency sinusoidal embeddings perform well. If you are using MATLAB, you will either have to write your custom intermediate layer as mentioned here, or apply a function to the inputs before feeding to the network every time. A python class extending torch.nn.Module and a relevant MATLAB function for sinusoidal embeddings are provided. You may use them for reference as they might or might not match your required format exactly.
- An MLP (multi-layer perceptron) of 3-4 layers should be able to accomplish this task, trained for enough steps. Training such an MLP with Adam optimizer and learning rate in the range of 10^{-3} to 10^{-4} has been found to be effective.

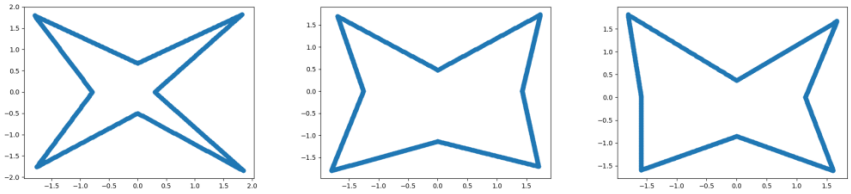

Figure 7: Examples of data samples from the *stars* dataset.

What to include in the report

- An introduction section describing the basics of diffusion probabilistic models, the underlying distributions, and the applications of such methods. This section should also explain the details of how the denoising task can be accomplished by a noise-predicting neural network. (About 1-2 pages)
- A method section with 2 subsections for the 2 mandatory tasks.
    1. In the first subsection, explain in your own words how the reverse process can be computed in closed form in the given setting.
    2. In the second subsection, explain in detail the steps involved in training your neural network and list the hyper-parameters used. Also, explain the steps involved in generating data at inference time.
- An experimental results section, again containing two subsections.
    1. For the first task, include the plots for forward, and backward trajectories and also the resulting distributions (of $\mathbf{x_T}$ in the forward process and $\mathbf{x_0}$ in the reverse process).
    2. For the second task, include scatter plots of the resulting 2D points from your inference process (reverse sampling from 2D Gaussian noise). Show results obtained at different stages of the training process (eg. after every 1k training steps). Also, include analysis over any other hyper-parameter whose impact you found interesting.

4.2 Advanced part

In the advanced part, you attempt to generate (somewhat) new data. Like 4.1.2, the data is still in the form of 2D points, however, instead of learning to generate points in a single pattern, you will try to generate different realizations of a shape template (examples shown in 7). You are provided 1024 data samples with 1024 points each. You can devise your own strategy for this or you can have roughly the same approach as before except the following major change.

1. Instead of your network producing noise for each point independently, the network would take in information about all points in the batch to estimate the noise for the batch. Tip : Since the output noise should permute similarly upon the input points being permuted in a certain way, the additional information from other points should be something that is permutation invariant - like the average position of the points.

Upon successful implementation, you should be able to generate samples which resemble the input data (perhaps noisy versions of them).

4.3 Theoretical Part

Answer the following questions in your report. Make sure to use your own words and justify your answers.

(a) Evaluating a generative model. Generative models should produce something meaningful - which resembles the training data, however the generated samples should also be diverse in nature. Consider a specific application of 3D shape generation, where you'd like to generate 3D points in shapes of chairs. Suppose you train a Diffusion model with a training data of 3D points sampled on 500 different chairs. You have 100 chairs kept as test data. Suggest ways of evaluating for your model (evaluation metrics along with how you would use them) that capture both the aspects - the reconstruction quality and the diversity.

(b) Generative models. (i) In your own words, explain why we optimize the Empirical Lower BOund (ELBO) rather than $\log(p(\mathbf{x}))$ when training a VAE. (ii) What are the major differences between VAEs and Diffusion probabilitic models? Answer in terms of the encoding/decoding processes in the two methods, and the latent variables.

(c) Triangulation. (i) Triangulation is used in order to create 3D points in Structure-from-Motion. Given a set of 2D pixel positions $\mathbf{x}_i$, $i = 1, ..., k$, in k images and the projection matrices $\mathtt{P}_i$ of these images, explain how to obtain the 3D point that best represents these 2D measurements.
(ii) Consider the setup with $k = 2$ images with known relative pose. Please identify in which case there are several 3D points that project exactly to $\mathbf{x}_1$ in the first image plane and to $\mathbf{x}_2$ in the second. Is it possible to derive the position of $\mathbf{x}_1$ and $\mathbf{x}_2$? Motivate your answer!

References

[1] Moises Arias. Cacao2 dataset. https://universe.roboflow.com/moises-arias-mg04u/cacao2-5zuma, nov 2022. visited on 2024-04-23. 4

[2] Universitas Gunadarma. Coffebeans dataset. https://universe.roboflow.com/universitas-gunadarma-1zr7d/coffebeans-5p6py, jul 2023. visited on 2024-04-23. 4

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 12

[4] Matic Švab. *Computer-vision-based tree trunk recognition.* Bsc. thesis, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2014. Available at https://www.vicos.si/resources/trunk12/. 2