Introduction

Diffusion Probabilistic models use latent variables to learn a distribution of the original data and then can produce high quality samples from that distribution. It includes an unparameterized forward process in which noise is added to the original example step by step according to a noise schedule until the final result is an isotropic standard gaussian noise sample. It also includes a reverse process, which uses a neural network to predict noise in the image at a particular step and returns the original image step by step. This assignment includes implementing a denoising diffusion probabilistic model process on toy datasets and studying the effect of various hyperparameters like No. of Training Steps, Model Complexity, No. of diffusion steps and various types of noise schedulers.

Model Setup

For this assignment, we built a DDPM model that uses a Feed Forward Neural Network as the function predictor $\epsilon_\theta(x_t,t)$. We use the sinusoidal position encoding generator in Transformers to embed positional information into the input data. We use an embedding constructed of 3 sinusoids to embed time information into the network. For the purpose of analysis of the model we fixed a few hyperparameters which, when changed, are mentioned explicitly.

- NN architecture - 3x16x32x16x3, where every layer except the output layer gets the time embedding concatenated to it (6x19x35x19x3). RELU was used in every layer except the output layer.
- Epochs - 500
- Diffusion Steps - 1000
- Beta Scheduler - Clamped Cosine
- Learning Rate - 1e-3
- Optimizer - Adam
- Batch Size - 1024
- Seed - 1618
- lbeta - 1e-5
- ubeta - 1.28e-2

We plotted graphs for three evaluation metrics, namely Earth Mover's Distance (EMD), Chamfers Distance and Negative Log Likelihood. The best model we used for submission has been trained on an architecture of 5 hidden layers along with 1000 diffusion steps and 2000 epochs.

Empirical Observation

We studied the model based on the four criterias presented in the paper and tried to reason about the process based on the results we obtained for both datasets provided. We also present a few hypotheses that we devised based on the empirical observations made. However, we couldn't conduct conclusive experiments to prove or disprove them, and we wish to do that in the near future whenever feasible.

1. Number of Epochs

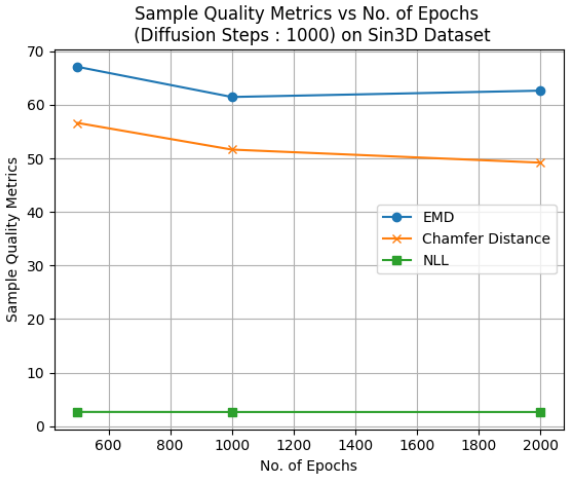

Fig. 1 Sample quality based on epochs for 3D Sin dataset

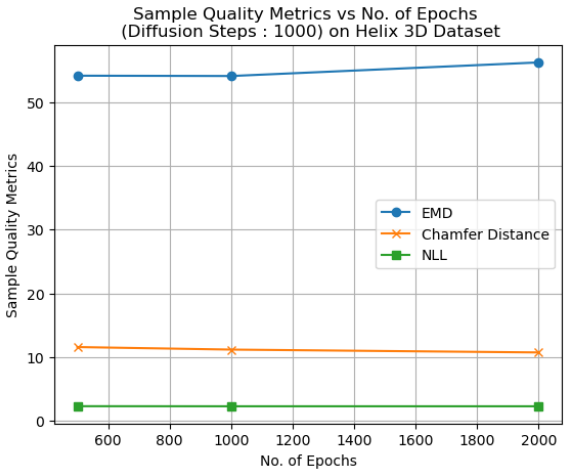

Fig. 2 Sample quality based on epochs for Helix dataset

As evident from Fig. 1 & Fig. 2, we can see a decrease in the metrics when the number of epochs for the training increases, keeping every other parameter the same. This shows that allowing for more epochs improves the performance of the model. The improvement might seem marginal metric-wise, but when we observed the visualisation for samples generated from the models, The higher epoch model gave results representing the ground truth much better than their counterparts.

2. Model Complexity

For this assignment, we chose the number of hidden layers used as the metric for measuring the complexity of our NN. We plotted the scores for three different numbers of hidden layers, 1,3 & 5, respectively.

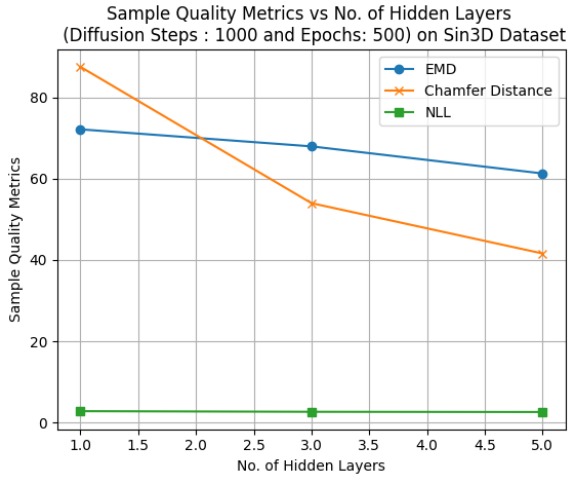

Fig. 3 Sample quality based on the number of hidden layers for the 3D Sin dataset.

We can see from Fig. 3 & Fig. 4 that the score drops when the complexity increases and the most changes we observed were when we increased the model complexity. This shows the importance of a powerful model to be able to learn the underlying distribution of noise as a crucial component in being able to create a successful model.

3. Diffusion Steps

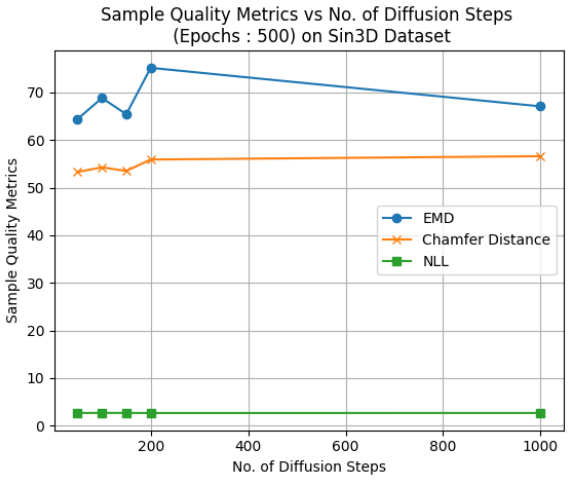

Fig. 5 Sample quality based on diffusion steps for 3D Sin dataset

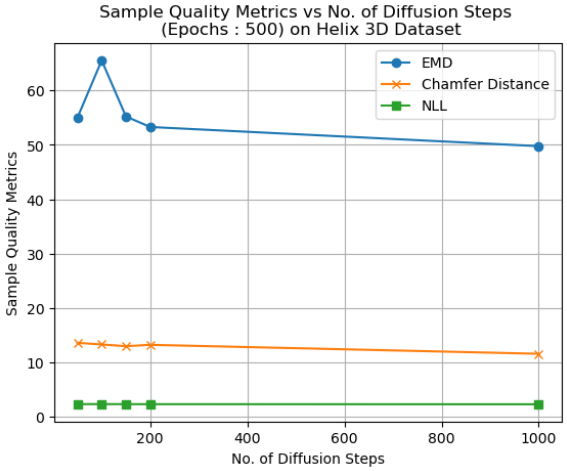

Fig. 6 Sample quality based on number of hidden layers for Helix dataset

We observed the scores for different numbers of diffusion steps mentioned in the assignment and tried to compare the results. It seems the scores fluctuate when we increase the number of steps keeping everything else fixed, as seen in Fig. 5 & Fig. 6. To ensure it wasn't a one-off error; we also tried changing the default epochs and rerunning the models.

The assignment required us to test on four different steps 50,100,150 & 200 resp. But due to such fluctuations we also tried an additional 1000 steps to see how the model behaved. We found that after the increase in the number of epochs, the results stabilised better compared to smaller epochs. However, we couldn't find an exact relation between the number of steps and epochs. We strongly believe there exists a correlation among these.

We present a hypothesis that the performance of the model will improve if the number of diffusion steps increase for sufficiently large epoch size, as a lower epoch size means the model doesn't get enough opportunity to learn the parameters for all $t \in T$. We tried to test it for a few different epoch sizes, but we couldn't verify the hypothesis due to several resource constraints.

4. Noise Schedule

For the purpose of this assignment, we decided to try five different schedulers and compare their results. In the paper [Nichol and Dhariwal 2021], they proposed the usage of cosine schedulers to get the values of β_t for the model. They hypothesized that a function whose gradient changes slowly at inception and towards the end would provide for a good noise scheduler.

We don't want to add too much noise in the initial stages of the diffusion process, as that would cause an accelerated loss of

information. Similarly, we don't want to add noise towards the end as the image is already very close to random noise and adding a high amount won't change anything.

We tried the linear, quadratic, root, cosine schedulers, respectively. Also, we used a cosine scheduler to calculate $\bar{\alpha}_t$ [Nichol and Dhariwal 2021] and a different cosine scheduler to calculate β_t. We wanted to see if simply using the cosine function to generate β_t and then squashing it in the range of lbeta and ubeta would give similar results as the method proposed in [Nichol and Dhariwal 2021].

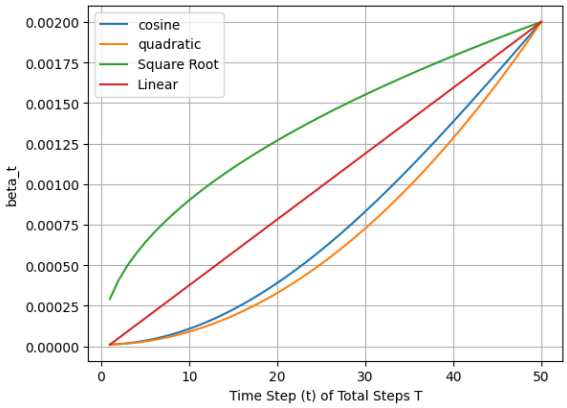

Fig. 7 Comparision of various Noise Scheduler curves

In Fig. 7, we can see how the curves compare to the cosine curve; based on that, we can predict the performance of our model. The curves that imitate the cosine function perform better when used for the noise scheduler. As evident from the graph, the quadratic function best imitates the cosine curve and also gives better performance than the rest . However, the cosine scheduler for $\bar{\alpha}_t$ still creates the best visualization for the ground truth even when we compare it to the cosine beta noise scheduler.

Observing the values generated by the noise functions, it became clear that the idea presented in [Nichol and Dhariwal 2021] did hold for the task. The functions with smaller gradient towards the end and start did give better noise schedules, and hence the square root noise scheduler gave the worst performance of them all. In case of the cosine schedulers, we observed that the $\bar{\alpha}_t$ creation function clamped the final values to ubeta instead of squashing the values in the range of [lbeta,ubeta], thus creating a better effect of asymptotic curve than our scheduler for βt and thus gave the best results.

We hypothesize that a noise scheduler that can map the sigmoid function to be clamped for the range of [lbeta,ubeta] will perform better than the cosine function we used. We weren't able to create the required sigmoid function, but we would definitely like to test on this hypothesis when the opportunity presents.

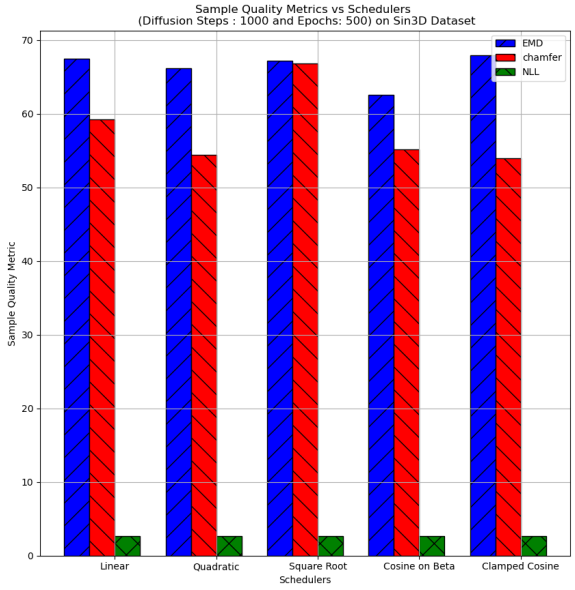

Fig. 8 Sample quality based on noise scheduler for 3D Sin dataset

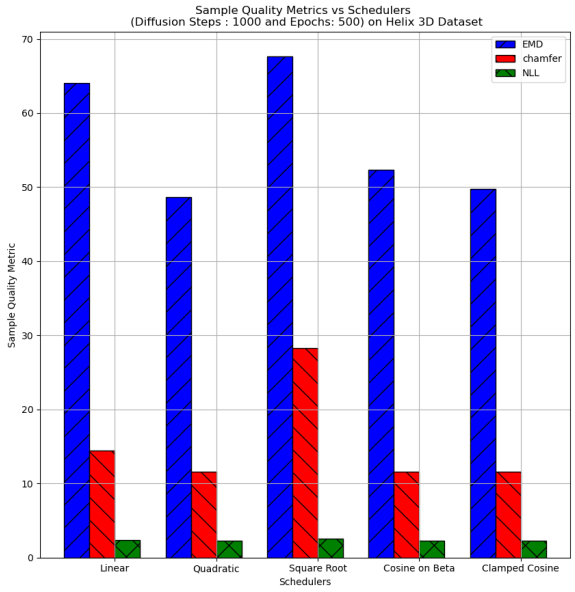

Fig. 9 Sample quality based on noise scheduler for Helix dataset

As evident from Fig. 8 & Fig. 9, it can be clearly seen that the cosine similar schedules work better. The linear and square root schedulers give low loss values. Still, when the samples are generated, they tend to be concentrated towards the mean rather than trying to imitate the respective distribution.

Conclusion

After carrying out the various experiments on the given parameters, we concluded that a DDPM model performs better when the number of training steps and the number of epochs are large. An NN with a higher number of hidden layers works better to generate parameters for the data distribution. A cosine based scheduler which allows asymptotic clamping towards the end points of betas works better for learning the underlying data distribution.

Based on this information that we acquired, we trained our model for the following specifications :

- Epochs - 2000
- Complexity - 5 hidden layers
- Diffusion Steps - 1000
- Noise Scheduler - Cosine clamped for $\bar{\alpha}_t$

The rest of the hyperparameters stayed the same as before. We trained and evaluated this final model on both datasets, and the results met our expectations. The model was able to learn the data distribution quite well, and the samples generated were very similar to the actual ground truth. The corresponding evaluation metrics for both datasets can be seen in table 1.

Dataset	EMD	Chamfer	NLL
Sin 3D	59.03120	30.28151	2.625234
Helix	51.63412	10.88840	2.253511

Table 1 Evaluation metrics of best model for both datasets

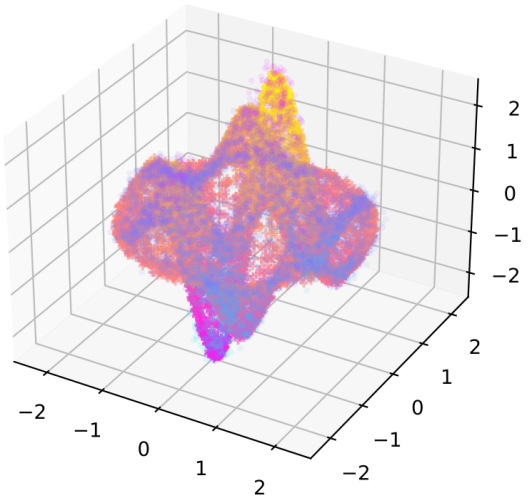

Fig. 10 Best model for Sin 3D dataset

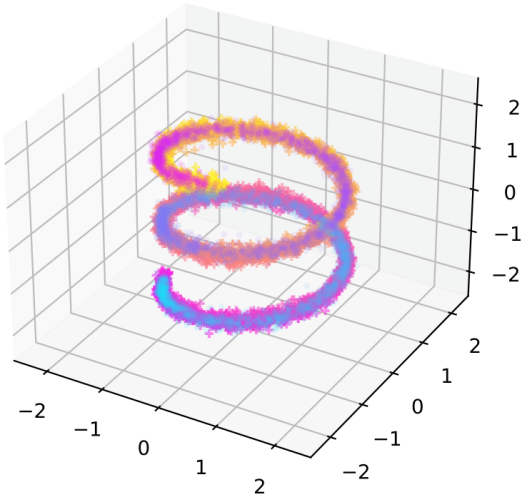

Fig. 11 Best model for Helix dataset

As evident from Fig. 10 & Fig. 11, the observations that we made regarding the DDPM based models do hold true for the given datasets.

A few observations that we made through the duration of this assignment is that the Chamfer distance was working as a better display of evaluation metric than the Earth Mover's Distance, i.e. we found that when the Chamfers distance for a model was lower, the samples were of better quality even though the EMD was equal or higher than the other models. Moreover, the Negative Log Likelihood didn't provide much help for evaluation, and this stands with the observations taught in class that Diffusion models don't evaluate well on log-likelihoods.

In conclusion, Diffusion based models show great promise in learning underlying data distribution and generate samples based on it in a manner that adheres to the structure, as it is clearly evident from Fig. 10 & Fig. 11, where the model has learned the ground truth data distribution pretty well.

References

[Nichol and Dhariwal 2021] Nichol, Alex, and Prafulla Dhariwal. 2021. "[2102.09672] Improved Denoising Diffusion Probabilistic Models." arXiv. https://arxiv.org/abs/2102.09672.