Exploring Denoising Diffusion Probabilistic Models

1 Introduction

In this assignment, you will implement a "*Denoising Diffusion Probabilistic Model*" (DDPM) [Ho et al., 2020] on a toy dataset. Your key goal in this assignment is to correctly implement a DDPM and critically analyse the results you obtain. It is strongly recommended that you go through the paper [Ho et al., 2020] *before* you start the assignment since the implementation will use notation very similar to that of the paper.

Starter code and dataset are available here: https://github.com/ashutoshbsathe/explore-diffusion

2 Data

The `data/` directory from our repo contains our toy 3D datasets – `3d_sin_5_5` and `helix`. The file `3d_sin_5_5_train.npy` contains 15686 samples which must be used for training the DDPM. Notice that each sample is a 3D vector which means the NumPy array that is loaded will have shape of $(15686, 3)$. The file `3d_sin_5_5_test.npy` contains 7781 test samples which we will use for visualization and evaluation of our DDPM after training.

3 Starter Code

The starter code contains following files:

1. *dataset.py*: This file implements our toy dataset as a custom PyTorch dataset.
2. *model.py*: Contains main implementation of DDPM. You must go through the comments of each function to understand their expected behavior. You are free to define more helper functions as a part of the main model class but make sure that core functionality is achieved via the functions we have provided.
3. *train.py*: Provides a PyTorch Lightning based training interface to train DDPM (implemented via *model.py*) on our toy dataset (implemented via *dataset.py*).
4. *eval.py*: Implements an interface to evaluate a trained DDPM model on various metrics. Also allows you to visualize the reverse diffusion process via an animation.
5. *eval_utils.py*: Various utility functions relevant for evaluation.

Specifically, your task is to complete the DDPM implementation in the *model.py* file, as per your understanding of the paper and gain insights from the results you obtain on *both* datasets.

4 Report

After the implementation is complete, study effects of various design choices of DDPM on the quality of generated samples on both datasets. We have implemented **Earth Mover's Distance** in our code which

you can use to assess the quality of the generated samples. You are free (NOT required) to implement other metrics that you feel are more suited but you *must* show results with EMD. Justify your choice of metric(s) in the report in the beginning. UWith the help of these metric(s), study following design choices in DDPMs on both datasets:

1. **Number of training steps** – *Does training longer help?* – Train at least 3 DDPMs with varying number of epochs (say $500, 1000, 2000$ respectively), keeping all other hyperparameters fixed. Produce a sample quality metric vs number of epochs plot in the report along with your comments about presence or absence of a trend in the plot. Don't forget to mention values of all other hyperparameters that you kept fixed and their possible impact on the presence or absence of a trend in your plot.
2. **Model complexity** – *How complex should our model be?* – Unlike number of training or diffusion steps, there isn't a concrete way to measure model complexity. A good proxy for measuring complexity could be simply the depth of the neural network you are using. For example: a single `torch.Linear` layer model can be considered very simple while a network with 10 `torch.Linear` layers can be considered very complex for our use case. Depending on your model architecture, you should define a complexity scale of models. Train at least 3 models with varying model complexity clearly stating your measure of model complexity and report your observations about sample quality similar to above question.
3. **Number of diffusion steps (T)** – *Are the gains marginal when increasing T?* – Train at least 5 DDPMs for $T = 10, 50, 100, 150, 200$ respectively, keeping all other hyperparameters fixed. Similar to above 2 questions, report your observations about impact of diffusion steps on sample quality. You may also use the diffusion animation function we have provided to track the trajectories of few samples throughout the diffusion process and use the trajectory to support your claims.
4. **Noise schedule** – *Which noise schedule works the best?* – The authors [Ho et al., 2020] mostly use a linear noise schedule throughout the paper. This question invites you to explore other noise schedules. At the very least, study the effect of hyperparameters `lbeta` and `ubeta` on the quality of generated samples. Try at least 5-10 different noise schedules and report your observations. This is an open ended question, credit will be given to those who explore alternate noise schedules (such as cosine) and for motivating the need of the newer noise schedules.

The main focus of this investigation are questions 3 and 4. You are advised to quickly iterate over hyperparameters to find optimal answers to questions 1 and 2 that fit within your computational budget and then spend most time on questions 3 and 4. For all the questions, you must compare results from both datasets and provide comments about differences and similarities between the results you observe.

5 Bonus

So far all our experiments have been based on a carefully curated toy dataset. However, the DDPM model is shown to be excellent at generating high quality image samples as well. In this part, we invite you to use all the insights you have gained during the assignment to train a DDPM on an image dataset called FashionMNIST [Xiao et al., 2017]. Notice that you will also have to implement dataloaders for this dataset yourself before proceeding. A straightforward approach to this could just be setting `n_dims=784` and simply retraining your best diffusion model on FashionMNIST. Report whether this straightforward approach worked well or not along with your possible explanations. You are also welcome to change the model architecture completely to something that is more popular in vision tasks.

If you are able to get a DDPM that produces good samples, study if you can get a control over the generation similar to [Radford et al., 2016]. For example, if you know 2 vectors that take you to a half sleeved shirt and to a full sleeved shirt respectively after the reverse process, is there a combination of these vectors that take

you to a three-quarters sleeved shirt? Figure 9 from the DDPM paper [Ho et al., 2020] can be a motivating starting point.

Note that bonus marks will be considered only if the entire main part of the assignment is completed.

6 Submission Structure

Once you are done with the assignment, submit (1) your best model for main assignment (2) your code for reproducing the best results (3) the report with the results you obtain (4) (optionally) code and model for the bonus part in a directory structure as follows:

```
submit/
    model.py
    train.py
    eval.py
    bonus_model.py
    bonus_train.py
    bonus_eval.py
    models/
        best_3d_sin_5_5.ckpt
        best_3d_sin_5_5.hparams.yaml
        best_helix.ckpt
        best_helix.hparams.yaml
        best_bonus.ckpt
        best_bonus.hparams.yaml
    report.pdf
```

Pack everything under `submit/` directory under a `.tar.gz` archive named as `rollno1_rollno2.tar.gz` and upload that to Moodle.

References

[Ho et al., 2020] Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.

[Radford et al., 2016] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.