BMI / CS 771 Fall 2023:
Homework Assignment 4

Dec 2023

1 Overview

We have recently witnessed some exciting development in image generation. In this assignment, you are invited to delve into diffusion models, a key driver behind these recent advances. Specifically, you will implement Denoising Diffusion Probabilistic Models (DDPM) for **conditional image generation**. You will train DDPMs and report the results on the MNIST dataset. Although DDPMs are underpinned by complex and elegant mathematical principles rooted in diffusion processes, you'll find their implementation refreshingly approachable.

This assignment is team-based and requires cloud computing. A team can have up to 3 students. The assignment has a total of 12 points with no bonus points. Details and rubric are described in Section 4.

2 Setup

- We recommend using Conda to manage your packages.
- The following packages are needed: PyTorch ($\geq$1.11 with GPU support), torchvision, PyYaml, NumPy, and Tensorboard. Again, you are in charge of installing them.
- You can install any common packages that are helpful for the implementation. If a new package is required for your implementation, a description must be included in the writeup.
- You can debug your code and run experiments on CPUs. However, training a neural network is very expensive on CPUs. GPU computing is thus required for this project. Please setup your team's cloud instance. **Do remember to shut down the instance when it is not used!**
- Our helper code will automatically download necessary datasets.
- To complete the assignment, you will need to fill in the missing code in: *./code/libs/ddpm.py*

- Modification to other part of the helper code (besides ddmp.py) should not be needed. Similar to HW3, you can experiment with different model designs and training hyperparameters using a YAML config file.
- Your submission should include the code, results, a trained model (see Section 6) and a writeup. The submission can be generated using: *python ./zip_submission.py*

3 Overview of the Implementation

Our helper code and the implementation (located under ./code) combines many of the pieces you previously saw in HW2 and HW3. You are required to go through our helper code before attempting the implementation.

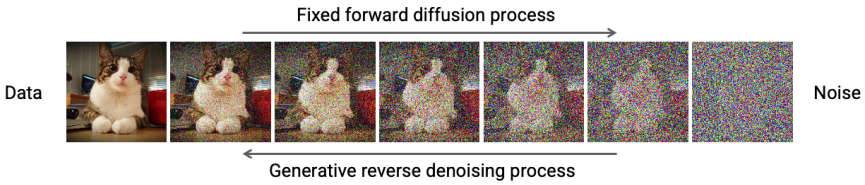

Figure 1: Overview of DDPM. DDPM consists of (a) a forward diffusion process that gradually adds Gaussian noise to an input image; and (b) a reverse denoising process that learns to remove the noise in a noisy version of the image.

Denoising Diffusion Probabilistic Models (DDPMs) are a new class of generative models that synthesize high-quality images by gradually transforming noise into structured patterns. This process, as illustrated in Figure 1, involves a series of steps where a model (often a deep model such as UNet [4]) learns to reverse a diffusion process, which adds noise to an image, thereby gradually reconstructing the original image from a noisy state. DDPMs simulate a process where they "denoise" an image, moving from random noise to a coherent, detailed image over multiple steps.

DDPMs were initially developed for unconditional generation, yet have since been adopted for text-conditioned generation (e.g., in stable diffusion [3]). In this assignment, we will consider generating images of digits conditioned on their labels, i.e., inputting "1" and generating an image of "1".

Code Organization. Simlar to HW3, the helper code has three parts.

- Source code for datasets, model architecture, DDPM, and training utilities are located under ./code/libs/.
- Configuration files (in yaml format) for the model architecture, training, and inference are stored under ./code/configs/.
- Training and evaluation interfaces are exposed in ./code/train.py.

DDMP Implementation (./code/libs/ddpm.py). This file implements a barebone DDPM as described in [1], including the forward diffusion process (q_sample), the reverse denoising process (p_sample and p_sample_loop), as well as the learning of the denoising function (compute_loss). The current implementation misses several critical pieces in order to function properly. **You will need to complete the code here (p_sample, q_sample, and compute_loss).** A good reference can be found in this link `https://huggingface.co/blog/annotated-diffusion`.

UNet Implementation (./code/libs/blocks.py and ./code/libs/unet.py). Central to DDPM is the learning of a denoising function. Practically, this denoising function is often implemented using a UNet [4]. Our helper code includes building blocks for UNet (./code/libs/blocks.py) and a reference implementation of UNet (./code/libs/unet.py). This implementation, largely following the UNet used in stable diffusion [3], combines convolutional residual blocks, self-attention blocks, and cross-attention blocks to allow conditional image generation. Completing the homework will unlikely require any modifications to these files.

Training and Other Utilities (all other files under ./code/libs). These files implement utility functions that are necessary for the assignment. You can find a similar centralized parameter configuration module as in HW3 (*./libs/config.py*). Similarly, you can use an external configuration file to control model architectures and training hyperparameters (in yaml format as ./code/configs/mnist.yaml). Our helper code also provide support to common datasets (*./libs/datasets.py*) for your exploration. Again, no modifications should be needed for these files.

Training and Evaluation Interface. Our helper code provides a training interface as described below (assuming ./code as the current directory).

- To train the model on MNIST dataset, run
  python ./train.py ./config/mnist.yaml
- By default, the training logs and checkpoints will be saved under a folder in *../logs*. Each run will have a separate folder that ends with the starting time of the experiment. You can monitor and visualize these training logs using TensorBoard. Similar to our previous assignment, we recommend copying the logs and plotting the curves locally.
  tensorboard --logdir=../logs/your_exp_folder

- You can resume from a previous checkpoint by using
  python ./train.py ./config/mnist.yaml --resume path_to_checkpoint

- Our training script will generate new images using the current model at the end of each epoch. These images (sample-#epoch.png) will be saved in the same folder where the checkpoints are saved. A visual inspection of these images will provide a good sense of the training progress.

4 Details

This assignment has two parts. An autograder will be used to grade certain parts of the assignment. **Please follow the instructions closely.**

Understanding DDPMs (4 pts). After reviewing the DDPM paper [1] and our helper code, you are required to address the following questions before attempting the implementation.

- What is the latent space of DDPMs? How can you draw samples from a DDPM?

- How does our helper code inject time and condition (image labels) into the denoising function (UNet)?

Implementing Forward Diffusion Process (1 pt). Forward diffusion process is implemented in the *q_sample* function. Please complete the code and briefly describe your implementation in the writeup.

Implementing Reverse Denoising Process (1 pt). Reverse denoising process is implemented in *p_sample* (single step) and *p_sample_loop* (full chain) functions. Please complete the code in *p_sample* and briefly describe your implementation in the writeup.

Implementing DDPM Loss (2 pts). The simplified DDPM loss is described in Algorithm 1 of the paper. Please implement this loss in the *compute_loss* function, and briefly describe your implementation in the writeup.

Experimenting with MNIST dataset (4 pts). Putting things together, we are now ready to train DDPMs on MNIST. With our default setup (30 epochs), the training will take around 1.5 hours. Digits should start to emerge in the sampled images before 10th epoch, and sampled images should begin to look fairly reasonable around 15th epoch. In your writeup, please include (a) training curves; and (b) samples from your model, and discuss these results.

Further Exploration. While we do not offer bonus points for this assignment, there are some interesting directions to explore for the sake of your curiosity.

- You might want to train DDPM on another dataset, yet this could be time consuming. For example, on CIFAR10, a larger model and a prolonged training schedule (say a day) are often needed.

- Training efficiency of DDPM can be improved with some proper design [2].
- One major drawback of the DDPM is that sampling requires many iterations. It turns out DDPM can be modified into a non-Markovian diffusion process, leading to significantly improved efficiency in sampling [5].
- A diffusion process can be modeled using a stochastic differential equation [6], leading to a family of models that are closely related to DDPMs.

5 Writeup

For this assignment, and all other assignments, you must submit a project report in PDF. Every team member should send the same copy of the report. Please clearly identify the contributions of all members. For this assignment, you will need to include (a) a response to the questions; (b) a brief description of your implementation; (c) a discussion of your results on MNIST. You can also discuss anything extra you did. Feel free to add other information you feel is relevant.

6 Handing in

This is very important as you will lose points if you do not follow instructions. Every time after the first that you do not follow instructions, you will lose 5%. The folder you hand in must contain the following:

- code/ - directory containing all your code for this assignment
- writeup/ - directory containing your report for this assignment.
- results/ - directory containing your results. **Please copy your final model (model_final.pth.tar in your experiment folder) and its corresponding config file in this folder.** Visuals including training curves and sampled images should be included in the writup and not in this folder.

Do not use absolute paths in your code (e.g. /user/classes/proj1). Your code will break if you use absolute paths and you will lose points because of it. Simply use relative paths as the starter code already does. **Do not turn in the data / logs / models folder**. Hand in your project as a zip file through Canvas. You can create this zip file using *python zip_submission.py*.

References

[1] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[2] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[3] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[4] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[5] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2020.

[6] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.