BMI / CS 771 Fall 2024:
Homework Assignment 4

Nov 2024

1 Overview

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

This assignment is team-based and requires cloud computing. A team should have 2-3 students, unless otherwise approved by the instructor. The assignment has a total of 12 points with up to 16 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$1.13 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 MNIST dataset. We have also provided a script to download AFHQ (v2) [4]. Simply run *sh ./download_dataset.sh*
- To complete the assignment, you will need to fill in the missing code in: *./code/libs/ddpm.py* and *./code/libs/unet.py*
- You are allowed to modify any part of the code in order to improve the model design or the training / inference of the model, though modification to other part of the helper code (besides *ddmp.py* and *unet.py*) is 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 this assignment.

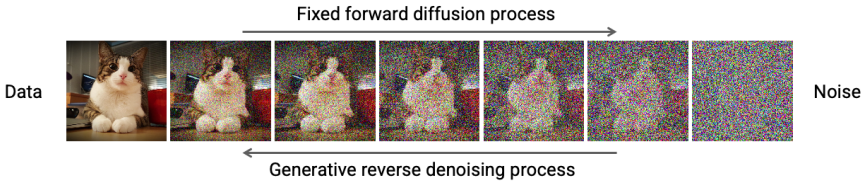

Figure 1: Overview of DDPM. DDPM consists of (a) a forward diffusion process that gradually adds Gaussian noise to an input image, resulting in white Gaussian noises at the end of the process; and (b) a reverse denoising process that learns to remove the noise from a noisy version of the image, allowing for transforming a white Gaussian into a target data distribution.

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 [8]) learns to reverse a diffusion process, which adds noise to an image. DDPMs simulate a process where the model "denoises" an image, moving from random noise to a coherent, detailed image over multiple steps. In doing so, it is capable of mapping a sample from a white Gaussian noise into a realistic image.

DDPMs were initially developed for unconditional generation, yet have since been adopted for text-conditioned generation (e.g., in Stable Diffusion [7]). In this assignment, we will consider generating images of digits or animal faces 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 DDPM as in [5] and a latent diffusion model as in [7], 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, p_sample_loop, 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 [8]. Our helper code includes building blocks for UNet (./code/libs/blocks.py) and a partial implementation of UNet (./code/libs/unet.py). Our implementation, largely following the UNet used in Stable Diffusion [7], combines convolutional residual blocks, self-attention blocks, and cross-attention blocks to allow conditional image generation. **You will need to complete the implementation.**

Autoencoder Implementation (/code/libs/tiny_autoencoder.py). We have also included an open source implementation of a lightweight autoencoder [2], distilled from Stable Diffusion models [7]. The pre-trained weights are stored under /pretrained and referenced in the AFHQ config file. This autoencoder will be used for the implementation of latent diffusion models [7].

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*). Thus, you can use an external configuration file to control model architectures and training hyperparameters (in yaml format as *./code/configs/mnist.yaml* and *./code/configs/afhq.yaml*). Our helper code also provides support to common datasets (*./libs/datasets.py*) for your exploration. *Most likely, you won't need to modify these files, even when attempting the bonus points.*

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 ./configs/mnist.yaml
- To train the model on AFHQ(v2) dataset, run
  python ./train.py ./configs/afhq.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
- Our training script will generate new images using the current model when saving a new checkpoint. The checkpointomg interval can be specified using "-c". For example, "-c 1" will create a checkpoints and generate new images at the end of every epoch. The resulting images (sample-#epoch.png) will be saved in the same folder as the checkpoints. A visual inspection of these images can help to monitor training progress.
- You can resume from a previous checkpoint by using
  python ./train.py ./configs/mnist.yaml --resume path_to_checkpoint

4 Details

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

4.1 UNet

Central to the implementation is a denoising function $\epsilon_\theta(x, c, t)$, realized using a UNet. In the vanilla version of DDPM, $\epsilon_\theta(x, c, t)$ takes the input of a noisy image (x) at time step t of the diffusion process, and a condition of image labels (i.e., context c). $\epsilon_\theta(x, c, t)$ then predicts the noise level. The estimated noise is then removed to produce a less noisy version of the image at time $t - 1$.

Understanding UNet (2 pts). We will start by reviewing the implementation of UNet. **Please address the following questions.**

- Can you describe the architecture of the encoder and decoder within the UNet?
- How does the current implementation inject time t and condition c (image labels) into the UNet?

Completing the implementation of UNet (1 pt). Next, we will move forward and complete the implementation in *./libs/unet.py*. The missing code involves the forward pass of the UNet decoder. **Please complete the code and briefly describe your implementation in the writeup.**

4.2 DDPM

Leveraging the denoising function, DDPM builds a reverse diffusion process to map an input simple distribution (Gaussian) to a target data distribution. We refer the technical details to the DDPM paper [5], as well as this excellent note [3] by Prof. Stanley Chan at Purdue University. **You will need to complete the following functions in** *./libs/ddpm.py* **and briefly describe your implementation in your writeup**

- **Forward diffusion process (1 pt)**. Forward diffusion process is implemented in the *q_sample* function.
- **Reverse denoising process (2 pts)**. Reverse denoising process is implemented in *p_sample* (single step) and *p_sample_loop* (full chain) functions.
- **Simplified DDPM loss (1 pts)**. This is implemented in *compute_loss* (see details in Algorithm 1 of [5]).

Experimenting with MNIST dataset (2 pts). Putting things together, we are now ready to train DDPMs on MNIST. With our default setup (30 epochs), the training may take an hour or two on a single T4 GPU. Digits should start to emerge in the sampled images before 10th epoch, and sampled images should begin to look fairly reasonable at the end of training. **In your writeup, please include (a) training curves; and (b) samples from your model, and discuss these results.**

4.3 Latent Diffusion Models

A major challenge in training DDPMs for high-resolution image generation is their significant computational cost. The standard DDPM defines the input Gaussian distribution over 2D grids at the target resolution. Consequently, the denoising function operates directly on these high-resolution grids, leading to substantial computational demands as resolution increases.

An effective solution is to consider Latent Diffusion Models (LDMs) [7], where the diffusion process is performed in a lower-resolution latent space rather than in the high-resolution image space. In this framework, a learned encoder maps input images into a compact, lower-resolution feature representation (the "latent space"). A DDPM is then trained to generate these latent features, significantly reducing the computational cost. Finally, the generated latent features are decoded back into high-resolution images using a learned decoder.

The encoder and decoder in LDMs are typically trained using variations of the variational autoencoder (VAE) framework and remain fixed during the DDPM training phase. This separation of tasks allows LDMs to efficiently generate high-quality, high-resolution images, by minorly extending the implementation of DDPMs. We refer to [7] for further technical details .

LDMs (1 pt). An LDM involves two key components: (1) the autoencoder, whose code and pre-trained weights are included; and (2) the DDPM, which you

have already implemented. In the DDPM class, the use_vae variable controls the use of autoencoder, and thus switches between the vanilla DDPM and LDM. You will need to implement the LDM using our help code, which should be fairly straightforward with a few minor modifications to the DDPM implementation.

Experimenting with AFHQ dataset (2 pts). Finally, we will train LDMs on the high resolution AFHQ dataset, consisting of 15,000 high-quality animal face images at 512x512 resolution. Full training on this dataset will unfortunately take substantial time and GPU resources. Here we will consider a simplified setting with a target resolution of 128x128 and a limited number of training iterations (300 epochs). With our default setup, the training may take a few hours on a single T4 GPU, and the results will not be very convincing due to insufficient training and the use of a vanilla DDPM. **In your writeup, please include (a) training curves; and (b) samples from your model, and discuss these results.** You will get the opportunity to further improve the generation quality in the bonus.

4.4 Bonus

This assignment offers several interesting directions for further exploration. To inspire creativity and encourage deeper engagement, we are offering bonus points for students who pursue these additional challenges.

Evaluating Image Generation (+4 pts). Throughout this assignment, we have relied on qualitative manual inspection of the generated images. How do we evaluate the results quantitatively? A widely used metric is Fréchet inception distance (FID) score, described in our optional lecture slides. The FID score, however, may be fooled by memorization of the training data, such that the generated images (training set) must have a similar distribution as the validation set. Bonus points will be provided for (1) implementing FID score and evaluating FID score for the MNIST experiment; and (2) demonstrating one solution to evaluate memorization. **Please include your implementation, describe your solution, and discuss the results in the writeup**.

Improving Inference Efficiency (+4 pts). DDPM inference is rather slow, as it requires running through many denoising steps (e.g., 1000). In fact, many of these steps are often skipped in practice, e.g., using the formulation of Denoising Diffusion Implicit Models [9]. Bonus points will be provided for exploring efficient inference of DDPM in the MNIST experiment, using DDIM or other efficient sampling approaches. A separate DDIM class is recommended for the implementation. **Please include your implementation, describe your solution, and discuss the results in the writeup**.

Improving DDPMs (+4 pts). DDPMs lead to a family of deep generative models. Many of these models share a similar denoising step, yet has different mathematical formulations and derivations. A recent work seeks to unified these models [6] under a simplified framework, demonstrating improved generation quality and faster training convergence. Bonus points will be provided

for exploring improved versions of DDPMs on the AFHQ dataset, using [6] or other approaches. Again, a separate DDPM class is recommended for the implementation.**Please include your implementation, describe your solution, and discuss the results in the writeup**.

Scaling up the Diffusion Model (+4 pts). You might have already noticed that our implementation of UNet has a few non-standard parts, such as the MLP used in the Transformer block. These are meant for compatibility with Stable Diffusion models. In fact, with some minor modifications to our UNet, one should be able to load Stable Diffusion 1.x checkpoints [1]. Bonus points will be provided for loading Stable Diffusion checkpoints using your implementation and demonstrating successful inference. You will need a few things to make this happen: (1) a modified UNet (a new class is preferred); (2) a CLIP text encoder from OpenAI (also on Hugging Face); (3) a function that loads Stable Diffusion checkpoints and set the model weights in our code; and (4) a proper config file for LDM. **Please include your implementation, describe your solution, and discuss the results in the writeup**.

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 and AFHQ. You can also discuss anything extra you did (e.g., for the bonus points). 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 additional results. Visuals including training curves and sampled images can be included in the writeup. This folder is intended for any results that could not fit into the report.

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] Stable diffusion. `https://github.com/CompVis/stable-diffusion`, 2022.

[2] Tiny autoencoder for stable diffusion. `https://github.com/madebyollin/taesd`, 2022.

[3] S. H. Chan. Tutorial on diffusion models for imaging and vision. *arXiv preprint arXiv:2403.18103*, 2024.

[4] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197, 2020.

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

[6] T. Karras, M. Aittala, T. Aila, and S. Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

[7] 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.

[8] 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.

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