Homework 5

Latent Denoising Diffusion Probabilistic Models

11-785: Introduction to Deep Learning (Fall 2024)

DUE: Nov 10 (Midterm) / Dec 10 (Final)
Writeup Version: **0.0.1**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk and work with other students for homework assignments.
  - You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.
  - You are allowed to help your friends debug, however - you are not allowed to type code for your friend
  - You are not allowed to look at your friends' code while typing your solution
  - You are not allowed to copy and paste solutions off the internet
  - Meeting regularly with your study group to work together is highly encouraged. You can even see from each other's solution what is effective, and what is ineffective. You can even "divide and conquer" to explore different strategies together before piecing together the most effective strategies. However, the actual code used to obtain the final submission must be entirely your own.
- **Overview:**
  - **Part 1**: Implementing and training Denoising Diffusion Probabilistic Models (DDPM) from scratch.
  - **Part 2**: DDIM Scheduler for inference speedup.
  - **Part 3**: Improvements of latent diffusion models based on pre-trained Vector-Quantize Variational Auto-Encoder (VQ-VAE) and Classifier-Free Guidance (CFG).
  - **Part 4**: Improving diffusion models with more advanced techniques.
- **Submission:**
  - **Report and Code**: includes your understanding of the basic implementations, more advanced techniques adopted, and evaluation results (FID, IS, Prec, Recall).
  - **Report Results**. You need to include the evaluation results for each part of this homework, i.e,. DDPM, DDPM + DDIM for inference, Latent DDPM, Latent DDPM with CFG, and with more advanced training techniques.

Checklist

Here is a checklist page that you can use to keep track of your progress as you go through the write-up and implement the corresponding sections in your starter code.

---

1. Getting started
    - ☐ Download the dataset
    - ☐ Download the starter code
    - ☐ Load the libraries, install Kaggle API, and download Kaggle dataset files
2. Implementation of DDPM
    - ☐ Understanding the mechanism of DDPM
    - ☐ Implementing DDPM from scrach
3. Implementation of DDIM Scheduler
    - ☐ Understanding why DDIM improves the inference speed.
    - ☐ Implement DDIM from scratch.
4. Implementation of Latent DDPM
    - ☐ Understanding the mechanism of using latent space and VAE for DDPM.
    - ☐ Implement latent DDPM.
5. Implementation of Classifier-Free Guidance.
    - ☐ Understanding conditional image generation and classfier-free guidance for diffusion models.
    - ☐ Implement CFG for diffusion models.
6. Advanced Training Techniques for Diffusion Models.
    - ☐ Exploring and implementing more advanced techniques for improving diffusion models.

Checkpoints

- **Midterm, Due Nov. 10th**. Initial report with DDPM and DDIM implementation and results.
- **Final, Due Dec. 10th**. Final report with all content and exploration.

Contents

1 Introduction

1.1 Overview

Diffusion models are a powerful class of deep generative models that have gained significant attention since their introduction. These models operate by progressively adding random noise to a dataset and then learning how to reverse this process, removing the noise step by step. The goal is to reconstruct the original data from the noisy version. This process, which will be detailed in later sections, is the core of diffusion modeling.

Initially developed for 2D image generation, diffusion models have since found broad application across various domains, including computer vision, robotic manipulation, and natural language processing. In this homework, you will explore the foundational concepts of diffusion models and gain hands-on experience by training your own model.

1.2 Tasks

You will be guided to implement a diffusion model that is capable of generating high quality images of any specific class that resembles the given images in the dataset. Except for the necessary implementations, you need to explore more advanced diffusion model training techniques from the recent papers or even make your own implementations. After training your model, you will need to generate a set of images, evaluate on them, obtain the Frechet Inception Distance scores (and other scores) and include them in your report. Your necessary implementation for this assignment should contain the following components:

- DDPM implementation from scratch.
- A DDIM noise scheduler that accelerates the generating process
- A Variational Autoencoder that maps the input space to the latent space for faster training.
- Following Classifier-Free Guidance during training and inference
- Evaluation of diffusion models.

The dataset you will be using is 128×128 ImageNet-100 with 100 clasess and approximately 130,000 images. The data can be downloaded here.

1.3 Homework Grading

The homework grading will be based on your report and code implementation:

- 30% - DDPM implementation
- 15% - DDIM implementation
- 20% - VAE implementation and latent DDPM
- 10% - CFG implementation
- 25% - More advanced techniques you explored

2 DDPM

2.1 Diffusion Models

Diffusion models are a class of generative models that excel at producing high-quality data, such as images, by learning to reverse a diffusion process [1]. The core idea behind diffusion models is to gradually add noise to data until it becomes indistinguishable from pure noise, and then train a model to reverse this process, thereby reconstructing the original data.

The term "diffusion" refers to the forward process where noise is incrementally added to the data. This forward process can be described mathematically as a Markov chain, where each step adds a small amount of Gaussian noise, progressively degrading the data until it becomes pure noise, as shown in Fig. 1.

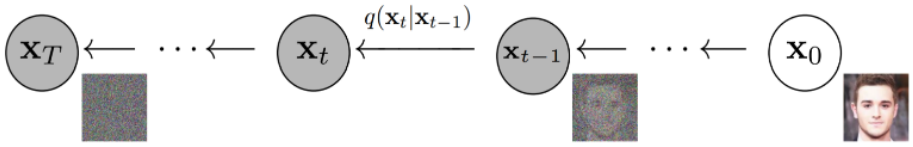

Figure 1: Forward Diffusion Process of DDPM

In the reverse process, the model is trained to remove the noise at each step, effectively reconstructing the original data from the noisy version. The success of the model lies in its ability to accurately perform this denoising at each stage, culminating in the generation of high-quality data.

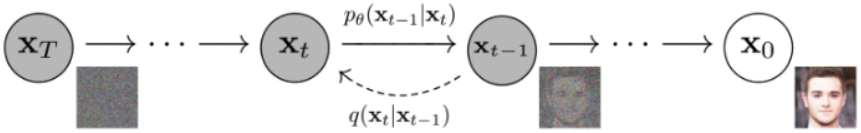

Figure 2: Reversed Diffusion Process of DDPM

Mathematically, diffusion models are latent variable models of form $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, ..., \mathbf{x}_T$ are latent variables with Gaussian noise and $\mathbf{x}_0$ as original data (image). The *reverse* process is a Markov chain with learned Gaussian transitions starting at random noise $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \quad p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \quad (1)$$

and the forward diffusion process that approximates the posterior $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ is fixed to a Markov chain that gradually adds Gaussian noise to data, according to a **variance** schedule

$\beta_1, ..., \beta_T$:

$$q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), \quad q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \tag{2}$$

Training a diffusion model is equivalent to optimizing the variational lower bound of $-\log p_\theta(\mathbf{x}_0)$ (more details in [1]). A common practice for the variance $\beta_{1:T}$ is to set them as fixed small values (pre-defined constants). Thus, for $\mathbf{x}_t$ at each timestep t in the forward process can be solved in closed-form:

$$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right)\mathbf{I}\right) \tag{3}$$

where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$ (detailed derivation in [1]).

In the reverse process, each latent variable $\mathbf{x}_{t-1}$ can be represented as a Gaussian distribution parameterized by the model's learned mean prediction:

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right) \tag{4}$$

where the mean $\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right)$ and variance $\boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)$ are typically modeled as functions of the current noisy data $\mathbf{x}_t$ and the timestep t. In DDPM, the mean is set as:

$$\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right) \tag{5}$$

where $\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)$ indicated the predicted noise from the model.

The training objective is to minimize the difference between the model's predictions and the actual noise added during the forward process. This is commonly achieved by optimizing the mean squared error (MSE) between the predicted noise and the true noise:

$$L_{\text{simple}}\left(\theta\right) := \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2\right] \tag{6}$$

The success of this training process enables the model to accurately reconstruct the original data from random noise during inference, making DDPMs powerful tools for generating high-quality samples across various applications.

2.2 Implementation

The implementation of DDPM encompasses the model architecture, training using the forward diffusion process, and inference via denoising through the reverse process. In the code framework, this involves several key components: `unet.py` for the U-Net model architecture, `train.py` for training code, `scheduling_ddpm.py` for the noise scheduler, and `ddpm.py` for the overall pipeline that integrates training and inference.

2.3 Architecture

The architecture of Denoising Diffusion Probabilistic Models (DDPMs) is crucial for their ability to generate high-quality data, particularly images. The model architecture typically follows a **U-Net** design—a specialized convolutional neural network (CNN) well-suited for tasks requiring both local and global feature extraction. We provide the fundamental modules, and your task is to integrate these modules into a complete network.

2.3.1 U-Net Structure

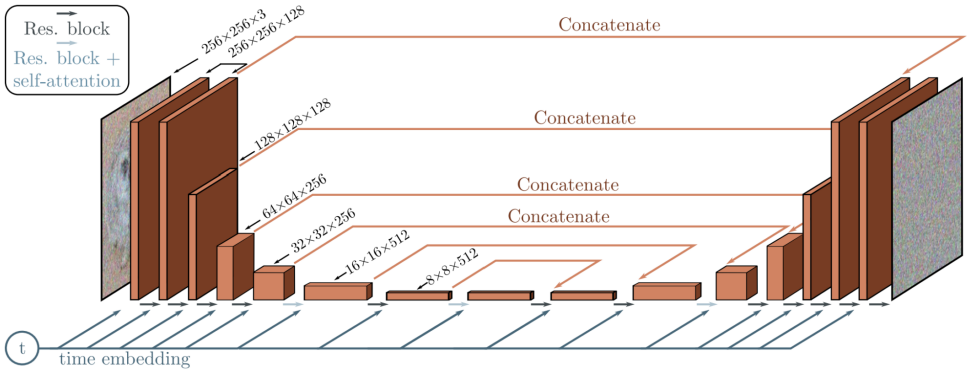

Figure 3: DDPM Architecture

The **U-Net** architecture consists of two main components: the encoder and the decoder, connected through a bottleneck.

- **Encoder (Downsampling Path):** The encoder progressively reduces the spatial dimensions of the input while extracting increasingly abstract features. This is achieved through a series of convolutional layers followed by downsampling operations, such as max-pooling. The encoder captures both fine-grained details and broader patterns within the data. It also stores intermediate feature maps, which are later utilized by the decoder.
- **Bottleneck:** Positioned between the encoder and decoder, the bottleneck processes the most compressed version of the data. It often includes layers that capture global context and complex relationships, sometimes employing attention mechanisms to enhance this capability.
- **Decoder (Upsampling Path):** The decoder mirrors the encoder but focuses on reconstructing the data to its original size. It utilizes the feature maps saved by the encoder (via skip connections) to combine detailed local features with broader context, ensuring that the final output is both detailed and coherent.

2.3.2 Positional Encoding

Since DDPMs involve processing data over many steps, it is crucial for the network to be aware of its current position within the sequence. This is achieved using **positional encodings**, which are added to the input data to inform the network of the current time step in the diffusion process.

2.3.3 Attention Mechanisms

To enhance the model's ability to focus on critical aspects of the data, modern DDPMs often incorporate **attention layers**. These layers enable the network to understand relationships

between distant parts of the input, which is particularly valuable in image generation tasks where context across the entire input is important.

2.4 Training

Training a DDPM involves teaching the model to reverse the noise addition process over a series of time steps. The process begins by corrupting the original data with progressively increasing Gaussian noise across a sequence of diffusion steps. The model, typically based on the U-Net architecture, is then trained to predict the noise added at each step, given the noisy data and the corresponding time step.

The training objective is to minimize a loss function, often the mean squared error (MSE) between the predicted noise and the actual noise added. By optimizing this loss, the model learns to accurately estimate the noise, enabling it to reconstruct the original data when the diffusion process is reversed. This ability allows the DDPM to generate high-quality data from random noise, making it a powerful generative model.

Your task is to implement the `add_noise` function within the DDPM noise scheduler and use this function to complete the training code.

2.5 Inference

Inference in DDPMs involves generating new data by reversing the noise addition process learned during training. Starting with a sample of pure Gaussian noise, the model iteratively applies the learned denoising steps, gradually transforming the noise into a coherent and high-quality output. At each step, the model predicts and removes the noise from the current noisy data, conditioned on the specific time step in the diffusion process. This reverse diffusion continues for a predetermined number of steps, progressively refining the sample until it closely resembles the original data distribution.

Remark

We suggest you test your DDPM implementation on CIFAR-10 or similar smaller datasets, instead directly training on the ImageNet-100 dataset we use for this homework. This is simply because training DDPM on raw pixel space requires extremely long training (e.g. you may expect to observe normal images after only 1M training steps).

3 DDIM Scheduler

3.1 Why DDIM?

Once problem of DDPM is that it involves a very long Markov chain in reverse process during inference, i.e., usually more than 1000 steps. This is highly inefficient. Denoising Diffusion Implicit Models (DDIM) [2] offer an efficient and deterministic alternative to the traditional Denoising Diffusion Probabilistic Models (DDPM). While DDPMs require a large number of steps to generate high-quality samples, DDIMs introduce a non-Markovian process that allows for faster sampling without sacrificing the quality of the generated data.

The key motivation behind DDIM is to reduce the number of sampling steps required for generating new data, which directly translates to faster inference times. This efficiency is achieved by leveraging a different formulation of the reverse diffusion process, where the noise schedule is modified to produce consistent outputs with fewer steps. Additionally, DDIMs provide a deterministic approach to sampling, ensuring that the same input will always produce the same output, which can be beneficial in certain applications.

3.2 Inference with DDIM

Inference using DDIM involves a carefully crafted scheduler that guides the reverse diffusion process to efficiently generate data from random noise. Unlike the standard DDPM scheduler, which operates under a probabilistic framework with many sampling steps, the DDIM scheduler reduces the number of required steps by applying a deterministic update rule.

3.2.1 DDIM Formulation

In the DDIM framework, the reverse process is formulated as follows:

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}} \left(\frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{\alpha_t}} \right) + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \sigma_t \cdot \mathbf{z}_t \tag{7}$$

where:

- $\mathbf{x}_t$ is the noisy data at timestep t.
- $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the noise predicted by the model at timestep t.
- α_t is the noise schedule.
- σ_t controls the variance of the noise added at each step.
- $\mathbf{z}_t$ is sampled from a standard normal distribution $\mathcal{N}(0, \mathbf{I})$.

In DDIM, the key modification is that σ_t can be set to zero, making the process deterministic. This deterministic update rule allows the model to generate the same output for the same input, ensuring consistency across multiple runs.

3.2.2 Implementing the DDIM Scheduler

To implement the DDIM scheduler, the following steps are required:

1. **Modify the Noise Schedule:** The noise schedule α_t used in the DDPM can be adapted to the DDIM framework. The scheduler should allow for both deterministic and stochastic sampling based on whether σ_t is set to zero or not.
2. **Reverse Diffusion Process:** The reverse diffusion process should be updated to follow the deterministic path defined by the DDIM formulation. This involves calculating $\mathbf{x}_{t-1}$ at each timestep using the DDIM update rule.

3. **Integration with the Existing Framework:** The DDIM scheduler needs to be integrated with the existing diffusion model framework. This includes setting up the scheduler in `scheduling_ddpm.py` and ensuring it works seamlessly with both training and inference code.

4. **Testing the Scheduler:** Rigorous testing should be conducted to verify that the DDIM scheduler maintains sample quality while reducing the number of steps. This involves comparing outputs generated by DDIM with those from DDPM under the same conditions.

3.2.3 Inference Process

The inference process using the DDIM scheduler starts by initializing $\mathbf{x}_T$ as random noise. The reverse process is then applied iteratively, using the DDIM update rule to gradually transform the noise into a high-quality output. Given the deterministic nature of DDIM, this process is consistent, producing the same output for the same initialization:

$$\mathbf{x}_0 = \mathbf{x}_T \rightarrow \mathbf{x}_{T-1} \rightarrow \cdots \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_0$$

Your tasks include finalizing the implementation of the DDIM scheduler, integrating it into the existing codebase, and performing thorough testing to ensure that the model generates high-quality samples efficiently.

4 Latent Diffusion Model (LDM)

4.1 Why Latent Space?

Running DDPM directly in pixel space, even with our low-resolution images, is extremely expensive. Latent Diffusion Models (LDMs) [3] leverage the concept of latent space to significantly reduce the computational cost and complexity of generating high-dimensional data, such as images. By operating in a compressed latent space, LDMs can efficiently model the underlying data distribution without directly working in the high-dimensional pixel space.

The key motivation for using latent space is that it captures the essential features of the data in a more compact and structured form. This reduction in dimensionality allows the diffusion model to focus on the most relevant aspects of the data, improving both the efficiency and effectiveness of the generation process. Furthermore, working in a latent space reduces the computational load, enabling faster training and inference times, which is particularly beneficial for high-resolution image synthesis.

4.2 VAE for the Latent Space

A Variational Autoencoder (VAE) [4, 5] is commonly used to map high-dimensional data into a lower-dimensional latent space. The VAE consists of an encoder, which compresses the input data into a latent representation, and a decoder, which reconstructs the original data from this latent representation. The latent space is typically structured such that similar

inputs are mapped to nearby points, preserving the data's essential features while reducing its dimensionality.

In the context of Latent Diffusion Models, the VAE plays a crucial role by providing a meaningful and compact latent space where the diffusion process can be applied. The VAE is trained to minimize the reconstruction error while ensuring that the latent space follows a known prior distribution (usually Gaussian). This training objective encourages the VAE to learn a latent space that is both informative and well-behaved, making it suitable for the subsequent diffusion process.

4.3 Implement the VAE

We provide a pre-trained VAE (with modules and pre-trained weights). Your task is to implement only the encoding process of VAE, which will be used to provide the input of DDPM, and the decoding process of VAE, which will be used for transforming the de-noised output of DDPM.

Remark

VAEs are of great importance nowadays for diffusion models, auto-regressive generation models, and multi-modality language models. There are also other types of VAEs, such as VQ-VAE [5], which we encourage you to explore more.

5 Classifier-Free Guidance

So far, we are dealing with unconditional image generation with diffusion models. However, in practice, conditional image generation is often more desirable.

5.1 Conditional Image Generation

Conditional image generation is a technique where the generation process is guided by additional information, such as class labels or textual descriptions. This approach allows for more controlled and specific outputs, making it possible to generate images that adhere to a given condition. In diffusion models, conditional generation is often achieved by incorporating the conditioning information directly into the model, influencing the generation process at each step.

For example, when generating images of a specific object (like a cat), the model is conditioned on the label "cat" throughout the diffusion process, ensuring that the generated image matches the specified category. This conditioning can be applied using various methods, including classifier-based guidance and the more recent classifier-free guidance.

5.2 Earlier Solution: Classifier-Guidance

In earlier approaches to conditional generation, classifier guidance was a popular technique. This method involved training a separate classifier to predict the class label from noisy

images generated during the diffusion process. The classifier's predictions were then used to adjust the reverse diffusion steps, guiding the model towards generating images that matched the desired condition.

Mathematically, the classifier guidance modifies the score function of the diffusion model by adding a term derived from the classifier's gradients with respect to the noisy image.

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t \mid y) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t) + w \cdot \nabla_{\mathbf{x}} \log p_\phi(y \mid \mathbf{x}_t) \tag{8}$$

where:

- $p_\theta(\mathbf{x}_t)$ is the unconditional diffusion model.
- $p_\phi(y \mid \mathbf{x}_t)$ is the classifier's probability of the label y given the noisy image $\mathbf{x}_t$.
- w is a weight controlling the influence of the classifier's guidance.

While effective, this approach had limitations, particularly the need to train a separate classifier and the potential for the classifier's errors to propagate into the generated images.

5.3 Classifier-Free Guidance (CFG)

Classifier-free guidance addresses some of the limitations of classifier-based methods by integrating conditioning information directly into the diffusion model, eliminating the need for a separate classifier. In this approach, the model is trained both conditionally and unconditionally, allowing it to learn how to generate images with and without conditioning information.

During inference, classifier-free guidance is implemented by interpolating between the conditional and unconditional score functions:

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t \mid y) = (1 + w) \cdot \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t \mid y) - w \cdot \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t) \tag{9}$$

where:

- $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t \mid y)$ is the score function conditioned on the label y.
- $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t)$ is the unconditional score function.
- w is a guidance weight that controls the strength of the conditioning effect.

This method simplifies the implementation by eliminating the need for a separate classifier and leveraging the model's ability to generate images with and without conditions. The guidance weight w allows for fine-tuning the balance between unconditional and conditional generation, providing flexibility in how strongly the condition influences the output.

5.4 Implementing CFG

Your tasks include writing the necessary documentation and setting up the code for classifier-free guidance. The implementation should be straightforward, as it primarily involves adjusting the score functions during the reverse diffusion process.

6 Evaluation of Diffusion Models

6.1 Overview

Evaluating the performance of generative models like diffusion models is crucial for understanding their ability to generate high-quality and realistic data. Two commonly used metrics in the evaluation of generative models are the Frechet Inception Distance (FID) and the Inception Score (IS). These metrics provide quantitative measures to assess the quality and diversity of the generated data, allowing us to compare models more systematically.

6.2 Frechet Inception Distance (FID)

The Frechet Inception Distance (FID) is a popular metric that compares the distribution of real and generated data in a feature space extracted by the Inception network. It calculates the Frechet distance between the two distributions, providing an indication of both the quality and diversity of generated samples. Lower FID values indicate that the generated data distribution is closer to that of real data, signifying better model performance.

Mathematically, FID compares the means and covariances of feature embeddings from real and generated data. Given two distributions, $\mathcal{N}(\mu_r, \Sigma_r)$ for real data and $\mathcal{N}(\mu_g, \Sigma_g)$ for generated data, the FID score is computed as:

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

6.3 Inception Score (IS)

The Inception Score (IS) is another commonly used metric for evaluating generative models. It measures both the quality and diversity of generated samples by leveraging the classification output of an Inception network. The score takes into account how confidently the model assigns labels to generated samples (indicating quality) and whether those samples cover a wide range of diverse classes (indicating diversity).

To compute IS, generated images are passed through an Inception model, and the entropy of the predicted label distribution is measured. Higher IS values correspond to higher quality and diversity of generated samples:

$$\text{IS} = \exp\left(\mathbb{E}_x\left[D_{\text{KL}}(p(y|x)||p(y))\right]\right)$$

6.4 Implementation

You will use the `torchmetrics` library to compute both FID and IS metrics. The library provides efficient implementations of these metrics, enabling easy integration into your model evaluation pipeline. You will be guided through the process of using `torchmetrics` to evaluate the diffusion models you train, helping you analyze the quality of generated data.

7 Exploring More Advanced Training Techniques

The above parts of this homework cover the very basic and necessary settings of training diffusion models. As a homework that is similar to the final project, we encourage you to explore different directions and techniques to improve on the basic diffusion models. Here are some possible directions:

- DDPM assumes a fixed variance in Gaussian transition, however you can use a learned variance which makes DDPM an iterative VAE actually.
- The noise schedule is very important for DDPM, you can explore different noise schedules other than the very basic linear schedule.
- Model size and architecture matters. You can try to enlarge the U-Net model or try more advanced models like DiT.

There are a lot more interesting and very effective techniques for improving the diffusion models, especially from recent papers. However, these directions are **prohibited**:

- Using more data.
- Fine-tuning from pre-trained diffusion models.

8 Conclusion

Good luck and enjoy the challenge!

References

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

[2] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[3] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn 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] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.