CS 726 Assignment - 2

Kishan Prajapati(21d070048),
Tanishka Pradhan(210040159),
Akshat Taparia(210110014)

March 2025

1 Q1 DDPM on Various Datasets

1.1 Code Explanation for DDPM

1.1.1 NoiseScheduler

The `NoiseScheduler` class defines the noise schedule used during the diffusion process in DDPM. It precomputes and stores different coefficients required for both forward (adding noise) and reverse (denoising) processes.

- **Parameters**:
  - `num_timesteps`: Total number of diffusion steps, T.
  - `type`: Type of noise schedule - `linear`, `cosine`, or `sigmoid`.
- **Key Quantities Computed**:
  - β_t: Noise added at each step.
  - $\alpha_t = 1 - \beta_t$: Remaining signal at each step.
  - $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$: Cumulative product of alphas.
  - $\sqrt{\bar{\alpha}_t}, \sqrt{1 - \bar{\alpha}_t}$: Used in forward and reverse steps.

Three types of schedules can be used:

1. **Linear**: Linearly increases β_t from `beta_start` to `beta_end`.
2. **Cosine**: Uses a cosine function to define $\bar{\alpha}_t$ and computes β_t accordingly.
3. **Sigmoid**: Uses a sigmoid function to define $\bar{\alpha}_t$, similarly computes β_t.

1.1.2 DDPM Model

The `DDPM` class defines the noise prediction network $\epsilon_\theta(x_t, t)$ used in the Denoising Diffusion Probabilistic Model.

- **Inputs**:
  - $x_t \in R^{n_{\text{dim}}}$: Noisy input data at time t.
  - t: Timestep scalar indicating the current step.
- **Architecture**:
  - Time embedding: Embeds t into a 128-dimensional vector using a 2-layer MLP with ReLU.
  - Model: Concatenates x_t with the time embedding and passes through a 4-layer MLP with LeakyReLU activations to predict noise.

2 Training Function

The `train` function trains the DDPM model by minimizing the MSE between the predicted noise and the true noise added to data samples.

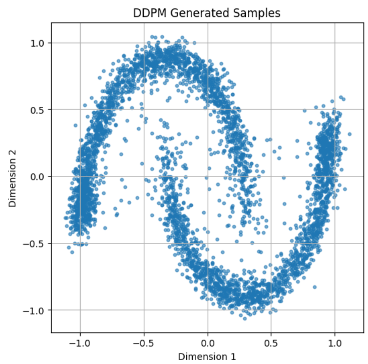
- For each batch:
  1. Sample random timesteps t.
  2. Compute $x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$.
  3. Predict $\epsilon_\theta(x_t, t)$ using the model.
  4. Compute loss: $\mathcal{L} = E[\|\epsilon - \epsilon_\theta(x_t, t)\|^2]$.
- Saves the model at each epoch.

2.0.1 Sampling Function

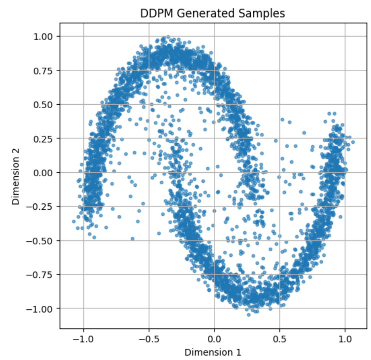
The `sample` function generates new samples by reversing the diffusion process.

- Initialize $x_T \sim \mathcal{N}(0, I)$ or use `init_noise` if provided.
- For $t = T - 1, T - 2, \ldots, 0$:
  1. Predict noise: $\epsilon_\theta(x_t, t)$.
  2. Compute the mean: $\mu_t = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$.
  3. Sample $x_{t-1} = \mu_t + \sigma_t z$, where $\sigma_t = \sqrt{\beta_t}$ and $z \sim \mathcal{N}(0, I)$.
  4. Skip the noise addition at $t = 0$.
- Returns the final denoised samples x_0 or all intermediate x_t if `return_intermediate=True`.

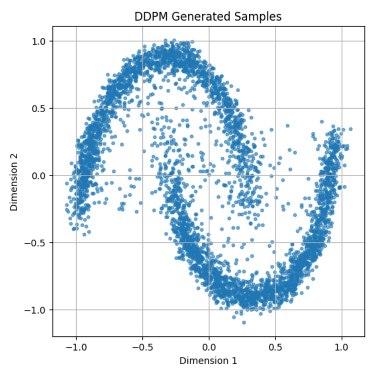
2.1 Results for Moon Datasets



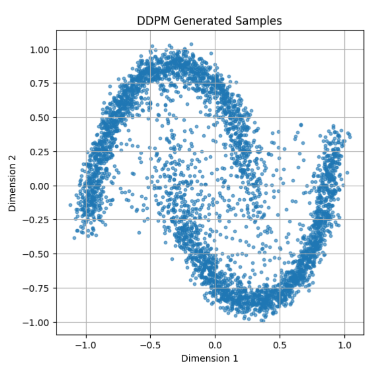
(a) T=10



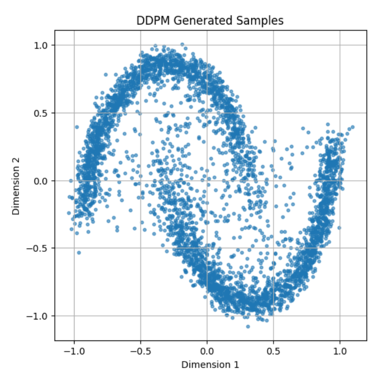
(b) T=50



(c) T=100



(d) T=150



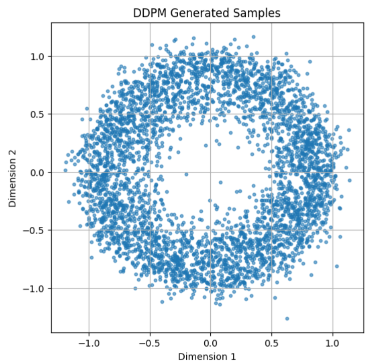
(e) T=200

Figure 1: Generated images at different diffusion steps T (Moons).

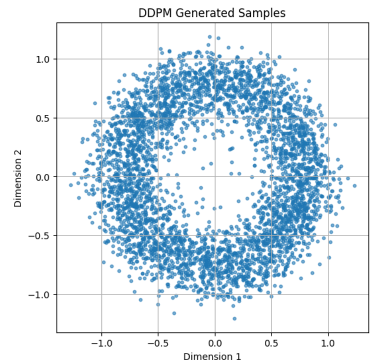
Table 1: Model Metrics for Different Diffusion Steps T (Moons)

Metric	$\mathbf{T = 10}$	$\mathbf{T = 50}$	$\mathbf{T = 100}$	$\mathbf{T = 150}$	$\mathbf{T = 200}$
Negative Log Likelihood	1.0414	0.9655	0.9633	0.9533	0.9570
Likelihood	0.3529	0.3808	0.3816	0.3854	0.3840
Gaussian Kernel Value	0.4146	0.3137	0.4675	0.4186	0.4664

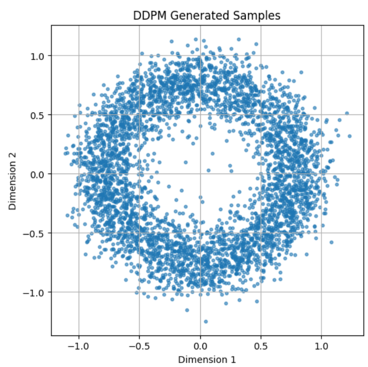
2.2 Results for Circles Dataset



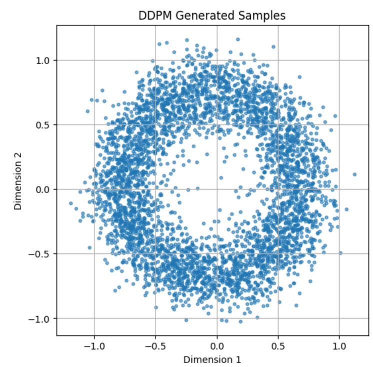
(a) T=10



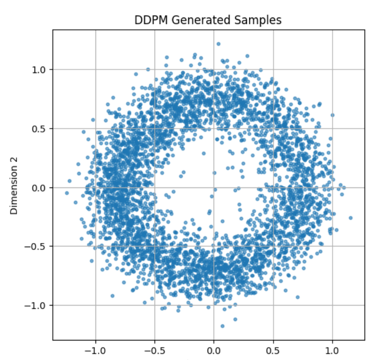
(b) T=50



(c) T=100



(d) T=150



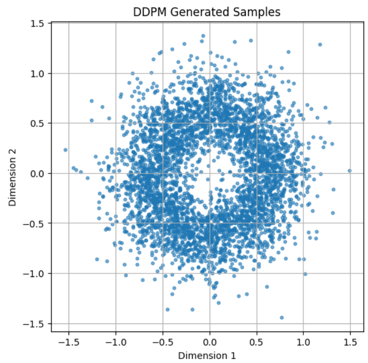
(e) T=200

Figure 2: Generated images at different diffusion steps T (Circles dataset).

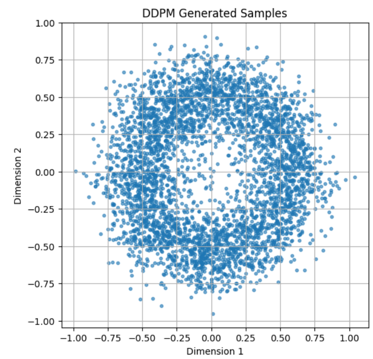
Table 2: Model Metrics at Different Diffusion Steps T

Metric	**T=10**	**T=50**	**T=100**	**T=150**	**T=200**
Negative Log Likelihood	1.05212	1.02246	1.00392	0.96619	0.99261
Likelihood	0.34920	0.35971	0.36644	0.38053	0.37061
Gaussian Kernel Value	0.35961	0.35894	0.42281	0.42389	0.41359

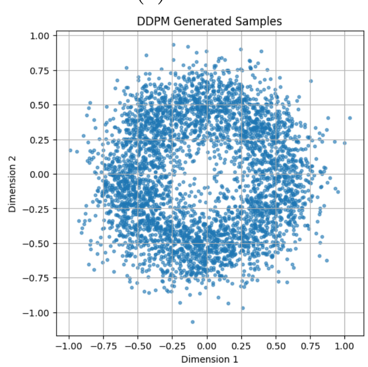
2.3 Results for Many Circles



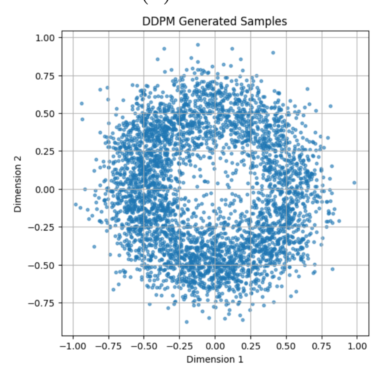
(a) T=10



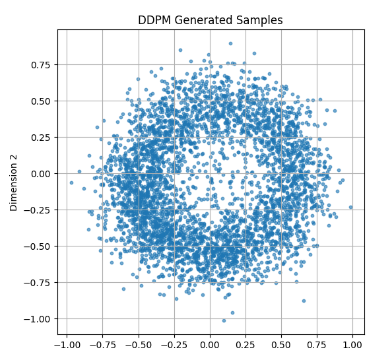
(b) T=50



(c) T=100



(d) T=150



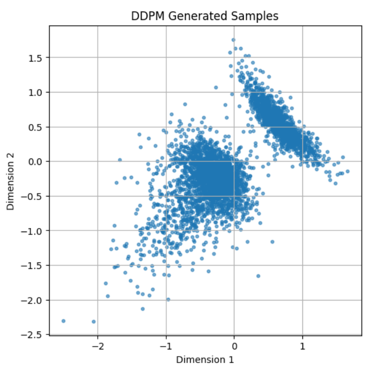
(e) T=200

Figure 3: Generated images at different diffusion steps T (Many Circles dataset).

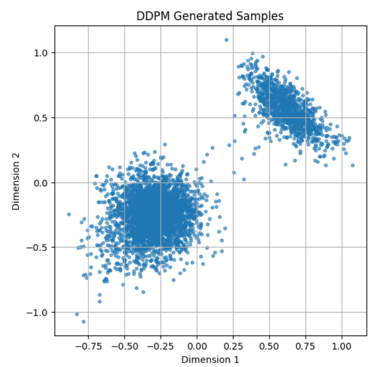
Table 3: Model Metrics Across Diffusion Steps T

Diffusion Step T	Negative Log Likelihood	Likelihood	Gaussian Kernel Value
10	0.74218	0.47608	0.49061
50	0.56781	0.56676	0.53427
100	0.56583	0.56789	0.76876
150	0.54987	0.57702	0.69144
200	0.54634	0.57907	0.61728

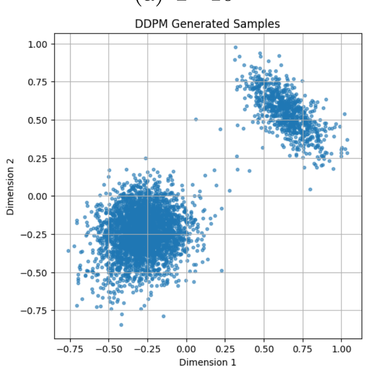
2.4 Blobs Datasets



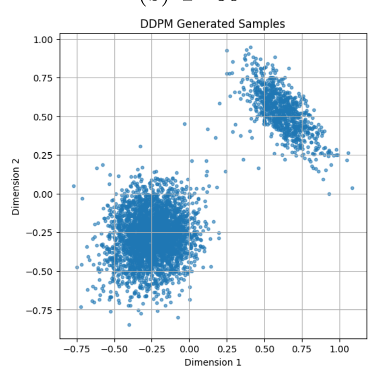
(a) T=10



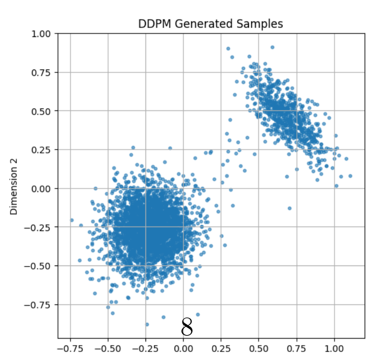
(b) T=50



(c) T=100



(d) T=150



(e) T=200

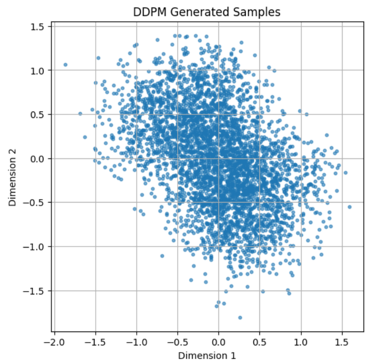
Figure 4: Generated images at different diffusion steps T (Many Circles dataset).

T	**Negative Log Likelihood**	**Likelihood**	**Gaussian Kernel Value**
10	0.4379	0.6454	0.8324
50	0.1259	0.8817	1.0663
100	0.0459	0.9551	1.2202
150	0.0653	0.9368	1.1989
200	0.0347	0.9659	1.1318

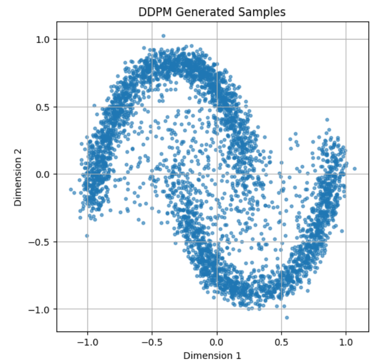
Table 4: Negative Log Likelihood, Likelihood, and Gaussian Kernel Value for various values of T

2.5 Analysis For different Noise Scheduler

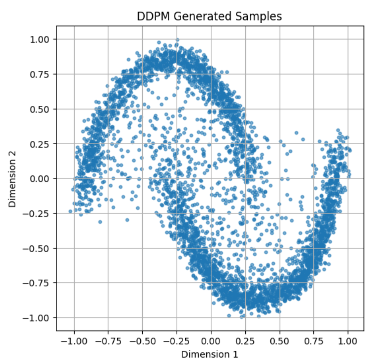
2.5.1 Cosine



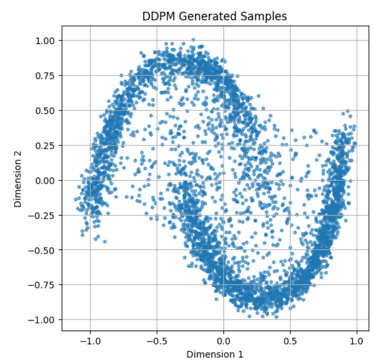
(a) T=10



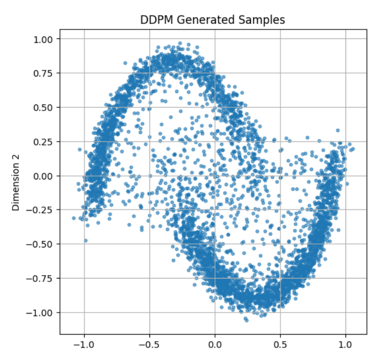
(b) T=50



(c) T=100



(d) T=150

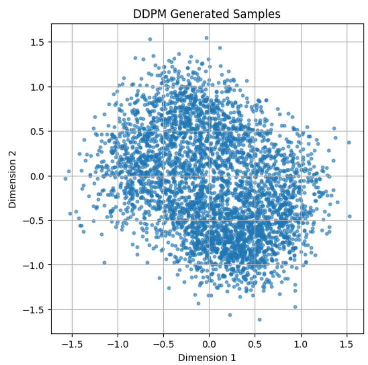


(e) T=200

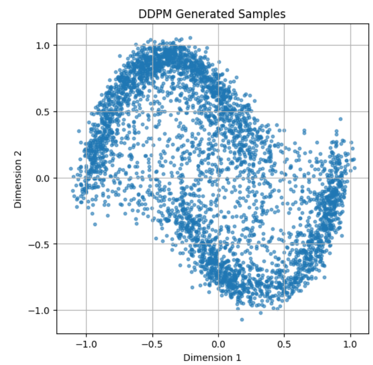
Figure 5: Generated images at different diffusion steps T (Many Circles dataset).

T	Negative Log Likelihood	Likelihood	Gaussian Kernel Value
10	0.9561	0.3844	0.3275
50	0.9352	0.3925	0.4158
100	0.9213	0.3980	0.4326
150	0.9395	0.3908	0.3991
200	0.9421	0.3898	0.4604

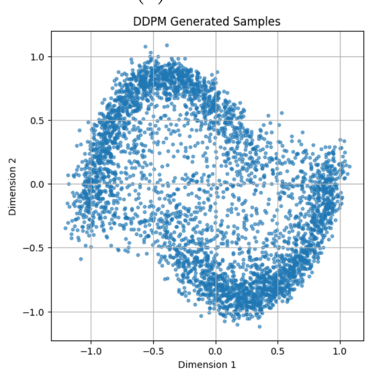
2.6 Sigmoid



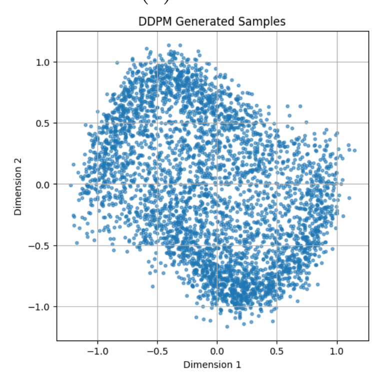
(a) T=10



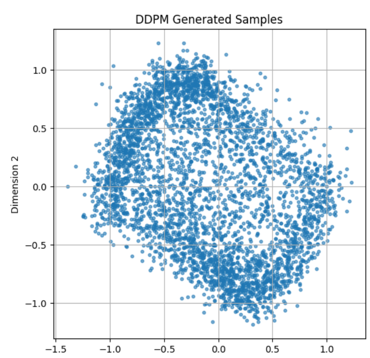
(b) T=50



(c) T=100



(d) T=150



(e) T=200

Figure 6: Generated images at different diffusion steps T (Many Circles dataset).

T	Negative Log Likelihood	Likelihood	Gaussian Kernel Value
10	1.0014	0.3674	0.4750
50	0.9324	0.3936	0.2836
100	0.9747	0.3773	0.2769
150	0.9518	0.3860	0.3725
200	0.9887	0.3721	0.4755

2.7 More Analysis on Linear, Sigmoid, Cosine

For Linear the $\mathbf{N}_L Ldependson Hyperparameters in following way$:

Table 5: Performance Metrics for Different β Values

lbeta	ubeta	NLL	Likelihood	Gaussian Kernel
1×10^{-4}	0.015	0.9570	0.3840	0.4664
1×10^{-4}	0.02	0.9611	0.3825	0.4385
1×10^{-4}	0.03	0.9347	0.3927	0.4233
1×10^{-5}	0.05	0.9307	0.3943	0.4120
1×10^{-5}	0.01	0.9564	0.3843	0.4361
1×10^{-6}	0.08	0.9464	0.3882	0.3934

For Cosine

Table 6: Performance Metrics for Different s Values (Fixed $num\_steps = 200$)

s Value	NLL	Likelihood	Gaussian Kernel
0.08	0.9421	0.3898	0.4604
0.05	0.9421	0.3898	0.4604
0.12	0.9421	0.3898	0.4604

For Sigmoid

Table 7: Performance Metrics for Different num_steps Values (Fixed $s = 0.08$)

num_steps	NLL	Likelihood	Gaussian Kernel
10	0.9561	0.3844	0.3275
50	0.9352	0.3925	0.4158
100	0.9213	0.3980	0.4326
150	0.9395	0.3908	0.3991
200	0.9421	0.3898	0.4604

The choice of noise schedule impacts how noise is added during the diffusion process. Here is a brief explanation of the three common types:

- **Linear:**
  Noise is added uniformly at each step, leading to a constant incremental change. This straightforward approach is easy to implement but may not optimally balance the trade-off between early noise injection and later refinement.

- **Cosine:**
  The cosine schedule typically introduces noise more gradually at the beginning and more aggressively in the middle, tapering off toward the end. This can help in preserving useful signal details early on while ensuring sufficient noise addition later for smoother denoising.

- **Sigmoid:**
  With a sigmoid schedule, noise addition follows an S-shaped curve — slow change at the beginning and end with a rapid change in the middle. This allows for a more controlled noise transition, potentially improving stability and sample quality by emphasizing critical phases of the diffusion process.

2.8 Conclusions and Explanation

The table (from Moons) above reports three key metrics — Negative Log Likelihood, Likelihood, and Gaussian Kernel Value — for different diffusion step settings T (10, 50, 100, 150, and 200) on the Moons dataset. The following observations can be made:

1. **Negative Log Likelihood (NLL):**
   A lower Negative Log Likelihood indicates a better model fit. From the table, we observe that as T increases from 10 to 150, the NLL decreases from 1.0414 to 0.9533, suggesting an improved model fit with more diffusion steps. However, the difference between $T = 150$ and $T = 200$ is marginal (0.9533 vs. 0.9570), indicating diminishing returns beyond a certain T.

2. **Likelihood:**
   The Likelihood metric, being the inverse of NLL in terms of model performance, increases as T increases. Specifically, likelihood improves from 0.3529 at $T = 10$ to 0.3854 at $T = 150$. This trend supports the notion that a higher number of diffusion steps allows for a more gradual noise schedule and more effective denoising. However, similar to NLL, the gains become marginal beyond $T = 150$, as seen by the slight drop at $T = 200$.

3. **Gaussian Kernel Value:**
   This metric can reflect the smoothness or the local consistency of the generated samples. The values vary with T, and while

there is not a perfectly monotonic trend, it is noticeable that at $T = 100$ and $T = 200$, the values are higher (0.4675 and 0.4664, respectively) compared to $T = 50$ (0.3137). This suggests that the model might achieve better local consistency at certain intermediate or higher values of T, though the optimal T might depend on the specific metric or quality criteria being prioritized.

Conclusion:
Increasing the number of diffusion steps T generally leads to better performance in terms of both NLL and Likelihood, indicating an improved denoising process with more gradual noise addition and removal. However, as T increases beyond a certain point (e.g., beyond 150), the improvements become marginal, which highlights a trade-off between computational cost and performance gains. The Gaussian Kernel Value suggests that there may be optimal points where sample consistency is maximized, and that the relationship between T and this metric may be less straightforward than the likelihood metrics.

3 Q2 Conditional DDPM

3.1 Difference between Guided Sampling and Conditional Sampling

Conditional Sampling

In conditional sampling, the diffusion model is trained to generate outputs based on specific information, such as a class label c in class-conditional image generation. Both during training and while sampling, the model receives this conditioning information. The model learns to predict the denoising signal (or score) for the *conditional distribution* $p(z_\lambda \mid c)$ as follows:

$$\epsilon_\theta(z_\lambda, c) \approx -\sigma_\lambda \nabla_{z_\lambda} \log p(z_\lambda \mid c)$$

This means the model learns to generate samples that match the desired condition, for example producing images of a specific class.

Guided Sampling

Guided sampling adjusts the generation process to produce higher-quality outputs, though it can reduce the variety of samples. This approach modifies the model's score during sampling to emphasize more realistic or higher-fidelity images. Two common types of guided sampling are:

Classifier Guidance

In this approach, the model's score $\epsilon_\theta(z_\lambda, c)$ is combined with the gradient from a separately trained classifier. This gradient helps steer the generated sample towards a specific class. The modified score is given by:

$$\tilde{\epsilon}_\theta(z_\lambda, c) = \epsilon_\theta(z_\lambda, c) - w\sigma_\lambda \nabla_{z_\lambda} \log p_\theta(c \mid z_\lambda)$$

Here, w is a weight that controls how strongly the classifier's guidance influences the sampling process.

Classifier-Free Guidance

Instead of relying on an external classifier, classifier-free guidance trains the diffusion model to work both conditionally and unconditionally. During sampling, a weighted combination of these two modes is used:

$$\tilde{\epsilon}_\theta(z_\lambda, c) = (1 + w)\epsilon_\theta(z_\lambda, c) - w\epsilon_\theta(z_\lambda)$$

This method simplifies the process by eliminating the need for a separate classifier while still improving the fidelity of the generated samples.

Summary Table

Aspect	Conditional Sampling	Guided Sampling
Purpose	Generate samples based on a given condition	Adjust sampling to produce higher-quality images
Score Used	$\epsilon_\theta(z_\lambda, c)$	Modified score (with classifier gradient or a mix of conditional/unconditional scores)
Classifier Needed	No	Yes for classifier guidance; not needed for classifier-free guidance
Training	Model is trained with conditioning information c	May require training a classifier or a joint conditional/unconditional model
Effect on Samples	Samples match the given condition	Samples have higher fidelity, though with less diversity

3.2 Code Explanation and Conditional DDPM implementation

This model represents a Conditional Denoising Diffusion Probabilistic Model (Conditional DDPM), where the objective is to learn how to predict noise added to a data sample at different diffusion time steps, while conditioning on both the time step and the class label. The key components and processing steps are described below.

1. Goal of DDPM

The goal of a DDPM is to model data through a process of gradually adding Gaussian noise over several time steps and learning to reverse this process by predicting the noise at each time step. By reversing the process, the model can generate new data samples from random noise.

2. Conditional Generation

In this conditional version of DDPM, the model is guided by class labels, allowing it to generate data samples conditioned on specific classes. This is achieved by embedding the class label and incorporating it into the noise prediction process.

3. Time Embedding

The model includes a time embedding module, which transforms the current diffusion time step into a higher-dimensional feature vector. This embedding helps the model understand how the noise evolves over time and to make accurate predictions depending on the current time step.

4. Class Embedding

Each class label is converted into a learnable vector embedding. This embedding captures the semantic meaning of the class and enables the model to generate samples that correspond to the specific class.

5. Input Concatenation and Padding

The noisy data sample, time embedding, and class embedding are concatenated to form a single feature vector. If the total dimensionality of the concatenated vector is less than a fixed input size (e.g., 512), zero-padding is applied to maintain a consistent input size for the neural network.

6. Noise Prediction Network

The concatenated and padded feature vector is passed through a multilayer perceptron (MLP). The MLP consists of multiple linear layers and non-linear activation functions. The output of the network is a prediction of the noise added to the data sample at the given time step, conditioned on the class label.

The `sampleCFG` function performs sample generation from a trained conditional DDPM model using Classifier-Free Guidance (CFG). Below is a step-by-step explanation of the process:

1. Initialization

- **Set the model to evaluation mode to disable dropout and batch normalization updates.**
- **Sample initial noise:**

$$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad \mathbf{x}_T \in R^{n_{\textbf{samples}} \times \textbf{model.n\_dim}}$$

- **Prepare conditional labels y (for the desired class) and unconditional labels $y_{\textbf{uncond}} = 0$ (often reserved for unconditional generation).**

2. Reverse Diffusion Process

We iterate backwards through the time steps $t = T, T-1, \ldots, 1$, applying a denoising update at each step.

2.1 Predict Noise (Conditional and Unconditional)

$$\begin{aligned}\hat{\epsilon}_{\textbf{cond}} &= \textbf{model}(\mathbf{x}_t, t, y) \\ \hat{\epsilon}_{\textbf{uncond}} &= \textbf{model}(\mathbf{x}_t, t, y_{\textbf{uncond}})\end{aligned}$$

2.2 Apply Classifier-Free Guidance (CFG)

$$\hat{\epsilon} = (1 + w) \cdot \hat{\epsilon}_{\textbf{cond}} - w \cdot \hat{\epsilon}_{\textbf{uncond}}$$

where w is the guidance scale.

2.3 Reverse Diffusion Step

Given the noise scheduler parameters α_t, β_t, and cumulative product $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, we compute:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \sqrt{1 - \alpha_t} \cdot \hat{\epsilon} \right) + I_{t>1} \cdot \sqrt{\beta_t} \cdot \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

3. Final Output

After the loop ends (i.e., at $t = 1$), $\mathbf{x}_0$ is returned, which represents the generated data sample conditioned on the specified class label y.

Key Concepts

- **Conditional DDPM: Generates samples conditioned on class labels.**
- **Classifier-Free Guidance (CFG): A technique to improve sample fidelity by interpolating between conditional and unconditional noise predictions.**
- **Noise Scheduler: Controls the diffusion process using a schedule for β_t and computes α_t and $\bar{\alpha}_t$ accordingly.**

4 Notes

- **The code contains additional arguments beyond those listed, such as options to choose between Conditional DDPM and standard DDPM, as well as which noise scheduler to use.**

5 Individual Contribution

- **We all three have worked together for the assignment.**

References

- **For Syntaxes and logic Implementation : ChatGpt and Internet is used**
- **Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising diffusion probabilistic models.* Advances in Neural Information Processing Systems, 33:6840–6851, 2020.**
- **Jonathan Ho and Tim Salimans. *Classifier-free diffusion guidance.* In NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications, 2021.**