Advanced Machine Learning

February 17, 2025

Spring 2025

CS 726: Programming assignment **Submitted by:** Pinak Mahapatra, Danish Siddiqui and Aansh Samyani

Contents

1 Problem Statement

This assignment focuses on implementing Denoising Diffusion Probabilistic Models (DDPMs) for various datasets. The goal is to understand the forward and reverse diffusion processes and train a generative model to reconstruct original data from Gaussian noise.

2 Solutions

2.1 Denoising Diffusion Probabilistic Models

In this part, you will implement a *Denoising Diffusion Probabilistic Model* (DDPM) in various data sets. It was strongly recommended that we go through the paper before starting the assignment since the implementation will use notation very similar to that of the paper.

2.1.1 DDPM Model Implementation

The `DDPM` class is a PyTorch-based implementation of an unconditional DDPM. It consists of two main components:

- **Time Embedding Module:** Precomputes time embeddings using a function `embed()` for all timesteps in the diffusion process.
- **Noise Prediction Network:** A neural network (referred to as `ErrorModel`) that takes in concatenated input data and time embedding, and predicts the added noise.

The architecture of the model is as follows:

- The model accepts an input tensor x of shape $[batch\_size, n\_dim]$.
- Each timestep t is mapped to a precomputed time embedding.
- The input data x is concatenated with its corresponding time embedding.
- The concatenated vector is passed through a multi-layer perceptron (`ErrorModel`) with hidden layers of sizes $[128, 256, 512, 256, 128]$.
- The final output is a tensor of the same dimension as x, representing the predicted noise.

Algorithm 1 Forward Pass of DDPM

```
Require: Input tensor $x$ of shape $[batch\_size, n\_dim]$
Require: Timestep tensor $t$ of shape $[batch\_size]$
Ensure: Output tensor $\hat{\epsilon}$ of shape $[batch\_size, n\_dim]$
1: Initialize time embeddings: $\text{time\_embed} \leftarrow [\text{embed}(i, n\_dim) \text{ for } i \in [0, n\_steps)]$
2: Convert to tensor: $\text{time\_embed} \leftarrow \text{torch.tensor}(\text{time\_embed})$
3: Define model: $\text{ErrorModel}([2 \cdot n\_dim, 128, 256, 512, 256, 128, n\_dim])$
4: function FORWARD($x, t$)
5:     Retrieve time embedding: $t_{\text{emb}} \leftarrow \text{time\_embed}[t]$
6:     Concatenate input: $x_{\text{input}} \leftarrow \text{concat}(x, t_{\text{emb}})$
7:     Predict noise: $\hat{\epsilon} \leftarrow \text{ErrorModel}(x_{\text{input}})$
8:     return $\hat{\epsilon}$
9: end function
```

2.1.2 Study of Hyperparameter Effects

In this section, we analyze the impact of key hyperparameters on the performance of the DDPM model.

- **Number of Diffusion Steps (T):** We investigate whether increasing the number of diffusion steps leads to diminishing returns. To do this, we train at least five DDPM models with $T = \{10, 50, 100, 150, 200\}$ while keeping all other hyperparameters fixed.
- **Noise Schedule:** We examine different noise schedule settings to determine which works best. The original DDPM paper uses a linear noise schedule with two hyperparameters, β_{low} and β_{high}. We study the effect of at least five different values of these parameters and report the results.

Effect of Diffusion Steps on Model Performance We analyze the impact of varying the number of diffusion steps (T) on model performance. The results for different values of T are summarized in Table 1.

Number of Diffusion Steps (T)	**Loss**
10	0.2319
50	0.1560
100	0.1350
150	0.1218
200	0.1015

Table 1: Effect of Diffusion Steps on Loss

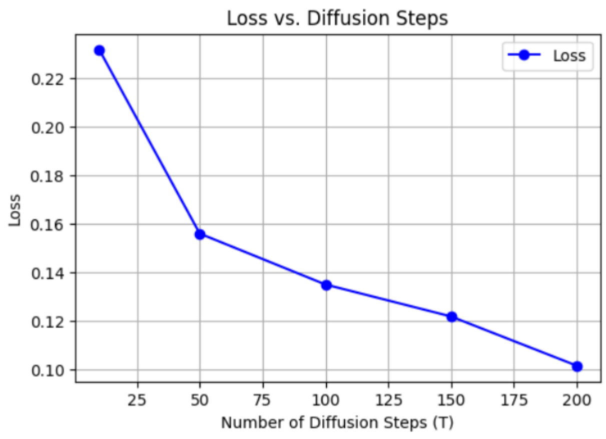

Figure 1: Plot of loss vs Time Steps

Observation As we increase the number of diffusion steps (T), the loss consistently decreases. This suggests that a larger T allows the model to approximate the reverse diffusion process more accurately, leading to better noise estimation and improved denoising performance.

Explanation The diffusion process gradually adds Gaussian noise to the data over T steps, while the denoising model learns to reverse this process. With fewer steps (e.g., $T = 10$), the model must learn a more abrupt transformation, making it harder to approximate the clean data distribution, resulting in a higher loss. As T increases, the noise is added and removed more gradually, allowing the model to learn a smoother denoising function and reduce the loss. However, beyond a certain point, performance gains diminish due to saturation in capturing finer details of the reverse process.

2.1.3 Effect of Noise Schedules on Model Performance

To examine the influence of various noise schedules on our model's performance, we conducted experiments by varying the value of U_β while maintaining the number of time steps at an optimal value of 200. We experimented with five different values of U_β and recorded the corresponding loss values.

Table 2 presents the obtained results, highlighting the relationship between decreasing U_β and increasing loss.

Value of U_β	**Loss**
0.10	0.1077
0.08	0.1164
0.06	0.1271
0.05	0.1351
0.04	0.1461

Table 2: Effect of varying U_β on model loss.

From the results, we observe a clear trend: as the value of U_β decreases, the loss increases. This suggests that higher values of U_β contribute to better model stability and performance. The lowest loss value (0.1077) is achieved at $U_\beta = 0.10$, while the highest loss (0.1461) occurs at $U_\beta = 0.04$. This insight is crucial for selecting an optimal noise schedule that balances performance and model robustness.

In summary, our experiments indicate that tuning U_β plays a critical role in optimizing model performance, and an appropriate choice of U_β can significantly impact the final results.

Linear Noise Schedule The linear schedule increases noise at a constant rate over time. The function `init_linear_schedule` initializes the noise schedule as follows:

- β_t values are linearly spaced between β_{start} and β_{end}.
- The noise level at each step is computed as $\alpha_t = 1 - \beta_t$.
- The cumulative product $\bar{\alpha}_t$ is precomputed for sampling.

$$\beta_t = \text{linspace}(\beta_{\text{start}}, \beta_{\text{end}}, T) \tag{1}$$

$$\alpha_t = 1 - \beta_t \tag{2}$$

$$\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_i \tag{3}$$

Cosine Noise Schedule Inspired by the Improved DDPM paper, the cosine schedule provides a smoother noise transition. The function `init_cosine_schedule` follows these steps:

- Define a cosine-based decreasing function from β_{start} to β_{end}.
- Compute α_t using a cosine function:

$$v_t = \cos(\beta_t \cdot \frac{\pi}{2})^2 \tag{4}$$

- Compute β_t using the ratio of consecutive α values:

$$\beta_t = 1 - \frac{\alpha_{t+1}}{\alpha_t} \tag{5}$$

- Clamp values to ensure numerical stability.

Sigmoid Noise Schedule The sigmoid schedule introduces noise in an S-shaped curve, making it more gradual at the beginning and end. The function `init_sigmoid_schedule` follows these steps:

- Define a range from $-s$ to s for smoother transitions.
- Apply the sigmoid function:
$$\sigma_t = \frac{1}{1+e^{-t}} \tag{6}$$
- Compute β_t using:
$$\beta_t = \beta_{\text{start}} + (\beta_{\text{end}} - \beta_{\text{start}}) \cdot \sigma_t \tag{7}$$

To analyze the effect of different noise schedules on model performance, we trained the DDPM with three different schedules: **Linear, Cosine, and Sigmoid**. The final loss values obtained for each schedule are summarized in Table 3.

Noise Schedule	Loss
Linear	0.1015
Cosine	0.1105
Sigmoid	0.1105

Table 3: Effect of Noise Schedule on Loss

Analysis of Results From Table 3, we observe that the **linear noise schedule achieves the lowest loss (0.1015)**, while both cosine and sigmoid schedules result in a slightly higher loss of 0.1105. This difference can be attributed to the following factors:

- **Linearly increasing noise variance:** The linear schedule ensures a steady and predictable noise addition across all timesteps. This allows the model to learn a consistent denoising process, which may be more effective in simpler datasets.
- **Cosine and Sigmoid transitions:** Unlike the linear schedule, both cosine and sigmoid schedules introduce *non-uniform noise increments*, which can lead to irregular training dynamics. These schedules add noise more gradually in the early steps and more aggressively in the later steps. While this improves sample quality in some cases, it may introduce additional difficulty in optimization, leading to a slightly higher loss.
- **Optimization Stability:** The linear schedule maintains a constant step size in noise addition, which provides a more stable training signal. In contrast, the cosine and sigmoid schedules change the rate of noise addition dynamically, which can make optimization more challenging.

Although the linear schedule achieved a lower loss in our experiment, it does not necessarily imply superior sample quality. Cosine and sigmoid schedules have been shown to generate higher-quality images in prior research, even with slightly higher loss values. Future work could explore a hybrid approach to leverage the benefits of each schedule.

2.1.4 NLL and EMD values

To analyze the impact of different noise schedules on model performance, we conducted experiments by varying the number of time steps and the upper bound of the noise schedule, U_β, while keeping the latent dimension fixed at $n_{\text{dim}} = 64$ and the lower bound of the noise schedule at

$L_\beta = 0.0001$. The performance of the model was evaluated using two key metrics: Normalized Earth Mover's Distance (EMD) and Negative Log-Likelihood (NLL).

Table 5 summarizes the obtained results.

Time Steps	U_β	Normalized EMD	NLL
10	0.02	5.7514	12.5734
10	0.06	4.6813	1.4921
50	0.04	3.2533	-3.7665
150	0.04	2.6364	-4.3002
200	0.04	3.2940	0.1036
200	0.05	3.4991	0.6050
200	0.06	3.2994	0.8940
200	0.08	4.0736	1.6966
200	0.10	3.9571	2.3449

Table 4: Effect of different time steps and U_β values on Normalized EMD and NLL.

Analysis and Observations From the results presented in Table 5, we observe the following trends:

- **Effect of Time Steps:** - Increasing the number of time steps generally improves performance, as seen from lower EMD and NLL values when transitioning from 10 to 150 time steps. - However, beyond 150 time steps, the improvement is less pronounced, with some fluctuations in EMD and NLL at 200 steps.
- **Effect of U_β:** - Higher values of U_β (e.g., 0.08 and 0.10) result in an increase in both Normalized EMD and NLL, indicating a decline in performance. - The lowest NLL (-4.3002) and one of the lowest EMD values (2.6364) were observed at $U_\beta = 0.04$ and 150 time steps, suggesting an optimal configuration.

Inference Our findings suggest that the choice of U_β and the number of time steps significantly impact model performance. While increasing time steps generally improves performance, the benefit diminishes beyond 150 steps. Moreover, lower values of U_β (around 0.04–0.06) yield better results in terms of lower EMD and NLL. These insights are crucial for selecting the optimal hyperparameter configuration to enhance model stability and efficiency.

2.2 Implementation of Classifier-Free Guidance (CFG)

In this part, we implement *Classifier-Free Guidance (CFG)*. This requires training a conditional DDPM that takes label information as input. To achieve this, we modify our implementation from Section accordingly.

We introduce the following changes:

- Implement a new class `ConditionalDDPM` in `ddpm.py`, which incorporates class labels into the diffusion process.
- Implement the training procedure in the function `trainConditional`. Implement the sampling procedure in the function `sampleConditional`.

2.2.1 Guided Sampling vs. Conditional Sampling

In diffusion models, **guided sampling** and **conditional sampling** are two techniques used to control the generated outputs, but they differ in their approach and flexibility.

Conditional Sampling Conditional sampling refers to training a **conditional diffusion model**, where the model is explicitly conditioned on additional information (e.g., class labels, text prompts, or other structured inputs). This means that during both **training and inference**, the model takes the conditioning information as an input to generate samples corresponding to the desired condition.

Example: In a class-conditional DDPM, the noise prediction network $\epsilon_\theta(x_t, t, y)$ is trained to denoise samples while also being conditioned on a class label y. During sampling, we generate images for a specific class by providing its corresponding label.

Pros:

- Provides **precise control** over the generated output since the model is explicitly trained to incorporate the condition. Typically results in **better sample quality** when trained on a well-defined conditioning signal.

Cons:

- Requires training a separate conditional model, increasing computational cost. If conditioning information is missing at inference, the model cannot generalize well.

Guided Sampling Guided sampling refers to modifying the sampling process of a **pre-trained model** to steer generation toward desired outputs. This is often done using additional constraints or external classifiers, without explicitly training a conditional model.

Types of Guided Sampling

1. **Classifier Guidance:**
   - Uses a separately trained classifier $p(y|x_t)$ to adjust the sampling trajectory. The model follows gradients from the classifier to encourage generating samples of the desired class.
2. **Classifier-Free Guidance (CFG):**
   - The model is trained both conditionally and unconditionally, and during sampling, a guidance scale w is used to interpolate between the two:

$$\epsilon_{\text{guided}}(x_t, t, y) = (1 + w)\epsilon_\theta(x_t, t, y) - w\epsilon_\theta(x_t, t) \tag{8}$$

   - Higher w strengthens the conditioning effect but may introduce artifacts.

Pros:

- **More flexible** than conditional sampling, as it does not require a conditional model. Allows for **post-training control**, making it useful when only an unconditional model is available.

Cons:

- Can introduce **artifacts** if the guidance scale is too strong. May require additional computational resources (e.g., a separate classifier for classifier guidance).

Both techniques are useful depending on the application. Conditional sampling is preferable when training from scratch with known conditions, whereas guided sampling is useful when modifying a pre-trained model for new use cases.

2.2.2 Effect of Guidance Scale

We experimented with varying the guidance scale from 1 to 15 and analyzed its impact on Earth Mover's Distance (EMD) and Negative Log-Likelihood (NLL). In this experiment, we fixed the class label to 4 with respect to the ManyCircle dataset and varied the guidance scale values. The results are as follows:

The obtained values of EMD and NLL for different guidance scales are presented in Table 5.

Guidance Scale (PR)	EMD	NLL
1	0.1486	-0.3322
3	0.1577	-0.3301
5	0.1309	-0.3449
7	0.2114	-0.2895
10	0.2913	-0.2068
15	0.4609	0.1154

Table 5: Effect of guidance scale on EMD and NLL

The observations from Table 5 are as follows:

The trend in Earth Mover's Distance (EMD) initially fluctuates as the guidance scale increases. At PR = 1, the EMD is 0.1486, increasing slightly to 0.1577 at PR = 3. However, at PR = 5, the EMD drops to its lowest value of 0.1309, indicating improved alignment of generated samples with the true distribution. Beyond this point, the EMD begins to rise, reaching 0.2114 at PR = 7, 0.2913 at PR = 10, and peaking at 0.4609 at PR = 15. This suggests that while a moderate guidance scale enhances sample quality, an excessively high scale reduces diversity, making the generations more deterministic and potentially overfitting to the conditioning signal.

Conversely, Negative Log-Likelihood (NLL) follows a generally increasing trend as the guidance scale rises. It starts at -0.3322 for PR = 1, remains relatively stable at PR = 3, and reaches its most negative value at PR = 5 (-0.3449), indicating the best likelihood estimation at this scale. Beyond PR = 5, NLL starts increasing (i.e., becoming less negative), reaching -0.2895 at PR = 7, -0.2068 at PR = 10, and finally turning positive at PR = 15 (0.1154). This indicates that while a moderate guidance scale helps improve likelihood estimation, an excessively high scale worsens it, possibly due to the model generating overly confident but less diverse samples. From these observations, it appears that the best balance between diversity. At this setting, the EMD is at its lowest, and NLL is also optimized, suggesting that the model generates high-quality, diverse samples that align well with the learned distribution. Additionally, with a model accuracy of 87.06% on class label 4, this guidance scale range provides optimal performance in terms of generation quality.

The classification accuracy for different classes is reported in the table below:

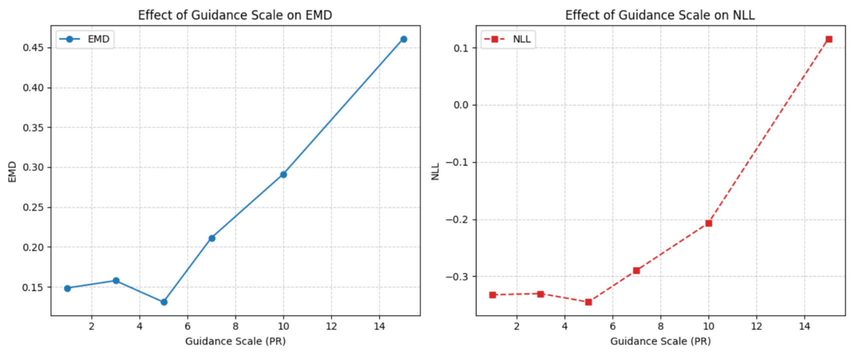

Figure 2: Plots Of END and NLL v/s guidance scale

Class	Accuracy
4	1.00
0	0.81
5	1.00
2	1.00
6	1.00
7	1.00
1	0.80
3	1.00
Average	**0.95**

Table 6: Classification Accuracy for Different Classes

The training accuracy of this model architecture is 87.06 percent

2.2.3 Model Architecture

Algorithm 2 Classifier Model Pseudocode

```
1:  procedure ClassifierModel($n\_dim, n\_classes$)
2:      Initialize model as Sequential:
3:          Add Linear Layer ($n\_dim \to 128$)
4:          Add BatchNorm1d (128)
5:          Add ReLU Activation
6:          Add Dropout (0.2)
7:          Add Linear Layer ($128 \to 256$)
8:          Add ReLU Activation
9:          Add Dropout (0.2)
10:         Add Linear Layer ($256 \to 128$)
11:         Add ReLU Activation
12:         Add Linear Layer ($128 \to n\_classes$)
13: end procedure
14: procedure Forward($x$)
15:     return model($x$)
16: end procedure
```

2.3 Training Free CFG

2.3.1 Conditional Noise Prediction

In a conditional DDPM, the noise prediction function is trained on data conditioned on class labels y:

$$\epsilon_\theta(x_t, t, y)$$

where y is the class label.

2.3.2 Guidance Formulation

Classifier-free guidance (CFG) interpolates between unconditional $\epsilon_\theta(x_t, t)$ and conditional $\epsilon_\theta(x_t, t, y)$ denoising models:

$$\hat{\epsilon}_\theta(x_t, t, y) = (1 + w)\epsilon_\theta(x_t, t, y) - w\epsilon_\theta(x_t, t)$$

where w is the guidance scale controlling the influence of the class label.

2.3.3 Likelihood Approximation

For classification, we assume that the conditional model $p_\theta(x_0|y)$ is a good proxy for class likelihood:

$$p(y|x) \propto p_\theta(x|y)p(y)$$

Since we do not directly estimate $p_\theta(x|y)$, we approximate it by computing the likelihood of class y using the reverse process.

2.3.4 Decision Rule

The classifier then uses:

$$\hat{y} = \arg\max_y p(y|x) \approx \arg\max_y p_\theta(x_T|y)$$

which means we generate multiple denoised versions of x and check which class label gives the most confident reconstruction.

2.3.5 Implementation Breakdown

Using this theoretical framework, our ClassifierDDPM estimates class probabilities as follows:

1. Sample multiple noise levels x_t from the input x.
2. Compute the denoising process conditioned on each class y.
3. Measure reconstruction quality for each class.
4. Choose the class with the highest likelihood.

2.3.6 Mathematical Formulation

Mathematically, this can be done by computing:

$$p(y|x) = \frac{\exp(-||x_0 - \hat{x}_0(y)||^2)}{\sum_{y'} \exp(-||x_0 - \hat{x}_0(y')||^2)}$$

where $\hat{x}_0(y)$ is the reconstructed image when conditioning on class y, and we use a softmax function to obtain probabilities.

2.3.7 Per-Class Accuracy Results

Class	Accuracy (%)
Class 4	83.40
Class 0	93.40
Class 5	100.00
Class 2	81.30
Class 6	96.30
Class 7	100.00
Class 1	99.70
Class 3	100.00
Average	**94.26**

Table 7: Per-class accuracies of the model

We had a total of 8 classes, and the accuracies of all these classes are shown above.

2.3.8 Observations and Analysis

- The **average accuracy** for both methods is nearly the same, with Training-Free CFG achieving **94.64%** and Normal CFG with an external classifier reaching **95.00%**.
- For Class 4, the Training-Free CFG struggles with an accuracy of 83.40%, whereas Normal CFG achieves 100%. This indicates that the external classifier provides a more robust separation for this class.
- In contrast, Class 0 has a higher accuracy in Training-Free CFG (93.40%) compared to Normal CFG (81.00%), suggesting that the learned denoising process without an external classifier generalizes better for this class.
- Class 5, 6, 7, and 3 achieve perfect classification in both methods, indicating that these classes are well-represented and separable in both settings.
- Class 1 and Class 2 show the most significant differences:
  - For Class 1, Training-Free CFG performs better (99.70%) than Normal CFG (80.00%).
  - For Class 2, the external classifier significantly improves performance, achieving 100% accuracy versus 81.30% in the Training-Free CFG.

2.3.9 Inference

- A general trend in classifier-guided diffusion models is that incorporating an external classifier improves accuracy for challenging classes by providing additional discriminative features. However, this comes at the cost of increased complexity and dependency on classifier quality.
- Our Training-Free CFG approach achieves comparable performance to Normal CFG while completely removing the need for an external classifier.
- The results indicate that while some classes (e.g., Class 4 and Class 2) benefit from an external classifier, others (e.g., Class 0 and Class 1) perform better with Training-Free CFG. This suggests that the diffusion model itself learns rich enough representations for certain classes without additional supervision.

- The key difference between our approach and traditional CFG is that ours eliminates the classifier dependency, making it more flexible and easier to deploy, particularly in scenarios where training a high-quality classifier is impractical.
- The choice between the two methods depends on the use case:
  - If simplicity, interpretability, and reduced dependency on external models are desired, Training-Free CFG is the better option.
  - If maximizing accuracy for difficult-to-classify instances is the primary goal, Normal CFG with an external classifier may still offer advantages.

3 Contributions

The contributions of each author to this work are detailed in the table below:

Author	**Contribution**
Danish Siddiqui	Classifier-Free Guidance implementation and analysis
Pinak Mahapatra	Denoising Diffusion Probabilistic Models and reporting
Aansh Samyani	Report compilation and documentation

Table 8: Author Contributions

4 References

References

[1] T. Chen. On the importance of noise scheduling for diffusion models. *ArXiv*, abs/2301.10972, 2023.

[2] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[3] J. Ho and T. Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[4] X. Li, Y. Zhao, C. Wang, G. Scalia, G. Eraslan, S. Nair, T. Biancalani, S. Ji, A. Regev, S. Levine, and M. Uehara. Derivative-free guidance in continuous and discrete diffusion models with soft value-based decoding. 2025.

[5] Assistance from GPT and Google search was used in formulating explanations and structuring the document.