Assignment 10: Diffusion Models - From Theory to Practice

1 Instructions

In this assignment, you will build and train a Denoising Diffusion Probabilistic Model (DDPM). You will bridge the gap between the theoretical mathematical formulations discussed in class and the architectural practices used to generate images.

First, you will explore the pure mathematics of the diffusion process by training a simple neural network to destroy and reconstruct a 2D point cloud. Then, you will apply these exact same principles to a more complex architecture—a time-conditioned UNet—to generate actual images.

1.1 Learning Objectives

By the end of this assignment, you should be able to:

1. **Implement the variance-preserving forward process** and explain why it converges to a standard normal distribution, in contrast to the simplified schedule from class.
2. **Train a neural network to reverse the diffusion process** by predicting noise ϵ, and articulate why this is mathematically equivalent to predicting the clean data x_0.
3. **Construct a time-conditioned UNet** and explain the architectural choices that make it suitable for image generation—including why spatial hierarchy and sinusoidal time embeddings are necessary at scale.
4. **Compare DDPM and DDIM sampling** in terms of generation quality, speed, and the trade-offs between stochastic and deterministic inference.

1.2 Submission Requirements

You must submit two components to Gradescope:

1. **The Report:** A plaintext Markdown document containing the 5 sections described in Part 3.
    - **No Images:** Describe your findings with text and data tables.
    - **Character Limit:** 7,500 characters.
2. **Supplemental Material:** Your `.ipynb` notebook containing all code, plots, and raw results.

2 Reference Guide: Bridging Class Notes to Implementation

In class, the forward process was defined using the general Markov transition:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \mu = w_\mu(t)x_{t-1}, \Sigma = w_\sigma(t)I)$$

For simplicity during derivation, the class notes set $w_\mu(t) = 1$ and $w_\sigma(t) = 1$. This resulted in the closed-form distribution $q(x_t|x_0) = \mathcal{N}(x_0, t \cdot I)$. While conceptually elegant, this simplified schedule causes the variance to grow continuously. As $T \to \infty$, the variance goes to infinity rather than converging to a stable Standard Normal prior $\mathcal{N}(0, I)$.

2.1 1. The Variance Preserving Schedule

To build a practical model that smoothly converges to $\mathcal{N}(0, I)$, we use a "Variance Preserving" schedule. Instead of setting w_μ and w_σ to 1, we define them using a noise variance parameter $\beta_t \in (0, 1)$:

- **The Variance (β_t)**: The fractional amount of noise injected at step t. We set $w_\sigma(t) = \beta_t$.
- **Signal Retention (α_t)**: To keep the total variance near 1, we scale down the mean. We define $\alpha_t = 1 - \beta_t$ and set $w_\mu(t) = \sqrt{\alpha_t}$.
- **Cumulative Signal ($\bar{\alpha}_t$)**: The total amount of the original x_0 signal that survives at step t. It is the cumulative product of all alphas up to t: $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

By substituting these specific w choices into the class equations, our collapsed closed-form transition from clean data to any noisy state t becomes:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I)$$

2.2 2. Standard Noise Schedules

A noise schedule dictates how we choose the sequence of $\beta_1, \ldots, \beta_T$.

A. The Linear Schedule (DDPM) The variance increases linearly from a small start value to a larger end value.

- **Equation:** $\beta_t = \beta_1 + \frac{t-1}{T-1}(\beta_T - \beta_1)$
- *(Standard hyperparameters:* $T = 1000, \beta_1 = 10^{-4}, \beta_T = 0.02$)

B. The Cosine Schedule (Improved DDPM) The linear schedule often destroys the image too quickly. The Cosine schedule defines the cumulative signal $\bar{\alpha}_t$ as a squared cosine wave, ensuring the destruction is spread perfectly across all T steps. We then derive β_t backwards.

- **Equation:** $\bar{\alpha}_t = \frac{f(t)}{f(0)}$ where $f(t) = \cos^2\left(\frac{t/T+s}{1+s} \cdot \frac{\pi}{2}\right)$
- **Deriving** β: $\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$
- *(Note:* $s = 0.008$ is a tiny offset to prevent singularities near $t = 0$, and β_t is typically clipped to a maximum of 0.999).

2.3 3. Implementation Algorithms

With our variables defined, the simplified algorithms from class evolve into the following practical implementation steps.

2.3.1 Algorithm 1: Training (Reweighted ELBO)

1. Sample a batch of clean data: $x_0 \sim q(x_0)$
2. Sample a random timestep for each item: $t \sim \text{Uniform}(\{1, \dots, T\})$
3. Sample random noise: $\epsilon \sim \mathcal{N}(0, I)$
4. Calculate noisy data x_t using the closed-form equation:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

5. Pass x_t and t through your neural network to get predicted noise $\epsilon_\theta(x_t, t)$.
6. Take a gradient descent step on the MSE loss: $||\epsilon - \epsilon_\theta(x_t, t)||_2^2$

2.3.2 Algorithm 2: DDPM Sampling (Stochastic)

1. Start with pure noise: $x_T \sim \mathcal{N}(0, I)$
2. Loop t from T down to 1:
    1. Predict noise: $\epsilon_{\text{pred}} = \epsilon_\theta(x_t, t)$
    2. Calculate reverse mean:

$$\mu_t = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\text{pred}} \right)$$

    3. If $t > 1$, sample stochastic noise $z \sim \mathcal{N}(0, I)$. If $t = 1$, set $z = 0$.
    4. Step backward: $x_{t-1} = \mu_t + \sqrt{\beta_t} z$
    - Implementation Note: For σ_t, you may use $\sqrt{\beta_t}$ (as in Ho et al.) or the posterior variance $\sqrt{\tilde{\beta}_t} = \sqrt{\frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t}$; both are acceptable.
3. Return x_0

2.3.3 Algorithm 3: DDIM Sampling (Deterministic)

1. Start with pure noise: $x_T \sim \mathcal{N}(0, I)$
2. Define a strided sub-sequence of timesteps $\tau = [T, T - s, T - 2s, \dots, 1]$
3. Loop i from length(τ) down to 2:
    1. $t = \tau[i]$ and $t_{\text{prev}} = \tau[i - 1]$
    2. Predict noise: $\epsilon_{\text{pred}} = \epsilon_\theta(x_t, t)$
    3. Predict the perfectly clean image:

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_{\text{pred}}}{\sqrt{\bar{\alpha}_t}}$$

4. Set stochasticity $\sigma_t = 0$ (for fully deterministic sampling).
5. Calculate direction pointing to $x_{t_{\text{prev}}}$:

$$d = \sqrt{1 - \bar{\alpha}_{t_{\text{prev}}} - \sigma_t^2} \cdot \epsilon_{\text{pred}}$$

6. Step backward: $x_{t_{\text{prev}}} = \sqrt{\bar{\alpha}_{t_{\text{prev}}}}\hat{x}_0 + d$

4. Return x_0

---

3 Part 1: The Mathematics of Diffusion (2D Toy Data)

Concept: To build an intuitive understanding of diffusion math, we will first apply it to 2D coordinates rather than images. This strips away the complexity of Convolutional Networks.

Setup:

- Generate a 2D "Swiss Roll" or "Spiral" dataset using `sklearn.datasets` (keep only 2 dimensions). Treat these (x, y) coordinates as your data x_0.

3.0.1 Task 1.1: The Closed-Form Forward Process

- Define $T = 100$ timesteps.
- Create a **Linear** variance schedule β_t. Calculate α_t and $\bar{\alpha}_t$.
- Implement the closed-form forward sampling function.
- **Notebook Output:** Pass your 2D dataset through the forward process. Plot 4 scatter plots side-by-side showing the 2D points at $t = 0, t = 33, t = 66$, and $t = 100$. Watch the shape dissolve into a standard normal distribution.

3.0.2 Task 1.2: Reversing the Process (The MLP)

- Build a simple Multi-Layer Perceptron (MLP) using PyTorch. The network should take two inputs: the 2D noisy coordinate x_t, and the scalar timestep t. (Concatenate t to x_t before passing it through the linear layers).
- Train the MLP using **Algorithm 1**.
- **Notebook Output:** Starting from random Gaussian noise, run your trained MLP through the **Algorithm 2 (DDPM Sampling)**. Plot 4 scatter plots showing the noise collapsing back into the structured 2D shape.

---

4 Part 2: Architectural Mechanics (Image Generation)

Concept: The math remains exactly the same, but generating images requires a network that respects spatial hierarchies. You will now transition from an MLP to a Time-Conditioned UNet.

Setup:

- Ensure your Colab runtime is set to GPU.
- Load the `MNIST` dataset (handwritten digits). Scale the pixel values to `[-1, 1]`.

4.0.1 Task 2.1: The Micro-UNet and Time Embeddings

- **Time Embeddings:** Unlike the MLP, you must project the scalar timestep t into a higher-dimensional continuous vector using Sinusoidal Positional Encodings (similar to a Transformer).

  > **A note on the change from Part 1:** In Part 1, we simply concatenated the raw scalar t to the input. This works fine when the network is a small MLP, the data is only 2 dimensions, and we have only $T = 100$ timesteps—the network can easily learn a direct mapping from a single number. However, when we scale to a deeper convolutional architecture with $T = 1000$ timesteps, a single scalar provides too little information for the network to distinguish between similar timesteps. Sinusoidal positional encodings project t into a rich, high-dimensional vector where each timestep has a unique signature, giving the network a much stronger signal. This is the same idea behind positional encodings in Transformers. Part 1 was not "wrong"—it was the right tool for a simpler problem.

- **The Architecture:** Implement a "Micro-UNet". It should have an encoder path, a bottleneck, and a decoder path using Conv2D layers.
- **Time Injection:** You must inject the time embeddings into the UNet. Map the time embedding to the channel dimension of your convolutional feature maps via a linear layer, and *add* it to the features inside your residual blocks.

4.0.2 Task 2.2: Training and Sampling

- Redefine $T = 1000$ for the image dataset.
- Implement the **Cosine** variance schedule.
- Train your Micro-UNet for 5-10 epochs using **Algorithm 1**.
- **DDPM Sampling:** Implement **Algorithm 2**. Sample a grid of 16 images taking the full 1000 steps.
- **DDIM Sampling:** Implement **Algorithm 3** (with $\sigma_t = 0$). Sample a grid of 16 images taking only 50 steps.
- **Notebook Output:**
  - A training loss curve.

– A 4x4 grid of images generated via DDPM.
– A 4x4 grid of images generated via DDIM.

---

5 Part 3: Content Requirements (The Report)

Your report must be organized into **exactly five sections** with Markdown headers.

5.1 Section 1: Executive Summary

- **One-Sentence Takeaway:** Summarize the core finding comparing the generation speed and visual quality between DDPM and DDIM sampling.
- **Summary Paragraph:** Outline the assignment progression, highlighting how the general mathematical transition definitions from class evolved into the functional architecture of the UNet.

5.2 Section 2: Forward Process Dynamics

- **Mechanics:** Explain the significance of the mathematical property that allows $q(x_t|x_0)$ to be computed in a single step rather than iteratively.
- **The Schedule:** Based on your visual output from Task 1.1, describe how the 2D shape degrades. Why did we need to transition from the simplified class schedule ($w_\mu = 1, w_\sigma = 1$) to the β_t variance preserving schedule?

5.3 Section 3: Architecture & Time Conditioning

- **Architectural Transition:** Why did we need to move from a simple MLP (Part 1) to a UNet (Part 2) when transitioning from 2D coordinates to images?
- **Time Injection:** Why is it strictly necessary to feed the timestep t into both networks? What would happen mechanically if the network tried to predict noise from an image without knowing t?

5.4 Section 4: The Training Objective

- **Objective Simplification:** The original VAE objective (ELBO) attempts to predict the clean data x_0. In this assignment, your models predict the noise ϵ instead. Explain mechanically why predicting the noise ϵ is mathematically equivalent to predicting x_0.

5.5 Section 5: Sampling & Inference

- **DDPM vs DDIM:** Compare the visual results of your DDPM (1000 steps) and DDIM (50 steps) generated digits.
- **Stochastic vs Deterministic:** DDPM relies on stochastic transitions (adding noise back in during the reverse step), whereas your DDIM implementation is deterministic ($\sigma = 0$). Discuss the trade-offs of having a deterministic generation process.

---

6 Grading Rubric

Each of the five sections is weighted equally (**20% each**).

Criterion	Excellent (5)	Good (4)	Satisfactory (3)	Okay (2)	Poor (1)
Section 1: Summary	Takeaway is specific and insightful. Summary elegantly connects the class math to the image architecture.	Takeaway is clear. Summary covers the main assignment arc.	Takeaway is generic. Summary is vague.	Summary misses key elements of the analysis.	Missing.
Section 2: Forward Process	Excellent explanation of the closed-form property and a deep understanding of why w_μ and w_σ had to be modified to preserve variance.	Good explanation of the schedule and step collapse.	Explains the degradation but the mathematical reasoning for the schedule change is weak.	Descriptions are incorrect or analysis is missing.	Missing.
Section 3: Architecture	Deep, mechanical analysis of UNet spatial properties and the strict necessity of time-conditioning.	Good explanation of time-conditioning necessity.	States that time is needed but lacks a discussion on how the network behaves without it.	Fails to grasp why the network needs t.	Missing.

Criterion	Excellent (5)	Good (4)	Satisfactory (3)	Okay (2)	Poor (1)
Section 4: Training Objective	Excellent, mathematically sound explanation connecting the noise prediction to x_0 estimation and the reweighted ELBO.	Good explanation. Explains the substitution well.	Mentions predicting noise is easier but lacks technical reasoning.	Evaluates tasks but ignores the objective simplification entirely.	Missing.
Section 5: Sampling	Thoughtful reflection on DDPM vs DDIM, directly comparing stochastic variance to deterministic speed trade-offs.	Good reflection on visual quality and speed differences.	Generic reflection (e.g., "DDIM is faster").	Minimal effort.	Missing.