CS5787: Exercises 4

`https://github.com/shl225/CS5787_HW4`

Sean Hardesty Lewis
shl225

1 Theory [50 pts]

1. [20 pts] In the context of a diffusion generative model, the forward process adds noise, which corresponds to a given time step (usually between 0 and 1000), to an initial data point x_0. A naïve approach is to sequentially add small amounts of noise:
$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t,$$
where $\epsilon_t \sim \mathcal{N}(0, I)$ is Gaussian noise and β_t is a predefined variance schedule (a small positive value that increases with t). Show that we can also obtain x_t in one step:
$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t,$$
where $\alpha_t = \prod_{i=1}^{t}(1-\beta_i)$.

   Proof:
   Starting with the recursive formula:
$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t.$$
   By unrolling the recursion:
$$x_t = \left(\prod_{i=1}^{t}\sqrt{1-\beta_i}\right)x_0 + \sum_{s=1}^{t}\left(\sqrt{\beta_s}\epsilon_s \prod_{i=s+1}^{t}\sqrt{1-\beta_i}\right).$$
   Let $\alpha_t = \prod_{i=1}^{t}(1-\beta_i)$, so $\sqrt{\alpha_t} = \prod_{i=1}^{t}\sqrt{1-\beta_i}$.
   The sum of scaled Gaussian noises remains Gaussian:
$$\epsilon = \sum_{s=1}^{t}\left(\sqrt{\beta_s}\epsilon_s \prod_{i=s+1}^{t}\sqrt{1-\beta_i}\right) \sim \mathcal{N}(0, 1-\alpha_t).$$
   Therefore:
$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t,$$
   where $\epsilon_t \sim \mathcal{N}(0, I)$.

2. Next, we derive the reverse process.
   1. [5 pts] Write the parameterization of the reverse process.

      The reverse process is defined as:
$$x_{t-1} = \mu_\theta(x_t, t) + \sigma_t\epsilon,$$
      where $\epsilon \sim \mathcal{N}(0, I)$, $\mu_\theta(x_t, t)$ is the model's predicted mean, and σ_t is the known variance.
   2. [15 pts] Derive the expression for $\mu_\theta(x_t, t)$ in terms of the predicted noise $\epsilon_\theta(x_t, t)$.

      We can estimate x_0 from x_t:
$$\hat{x}_0 = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, t)\right).$$

So the mean $\mu_\theta(x_t, t)$ is:

$$\mu_\theta(x_t, t) = \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\hat{x}_0 + \frac{\sqrt{1-\beta_t}(1-\alpha_{t-1})}{1-\alpha_t}x_t.$$

Which simplifies to:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\varepsilon_\theta(x_t, t)\right).$$

3. [5 pts] Write out the loss function used to train diffusion models.

   The loss function used is:

$$L = \mathbb{E}_{x_0,\varepsilon,t}\left[\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2\right],$$

   where $x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\varepsilon$.

4. [5 pts] Suggest one potential modification to this diffusion model and how it might improve / impact training.

   A potential modification is to learn the variance σ_t during training by parameterizing it as $\sigma_\theta(x_t, t)$. This allows the model to adapt the noise level dynamically, potentially improving sample quality and convergence speed.

2 Practical [50 pts]

3. [20 pts] Implement the forward diffusion process and visualize how a given data point is progressively corrupted by Gaussian noise. Implement the forward diffusion process using PyTorch. Start with 5 clean images of your choice, and at each time step, add Gaussian noise to the images according to the variance schedule β_t. Plot the original image and a series of noisy images at different time steps (e.g., t = 0, 10, 50, 100, 500) to visually demonstrate how the noise gradually corrupts the data. Additionally, calculate the mean squared error between the noisy images and the original images at each time step to quantify the level of corruption.

We apply a forward diffusion process to images using PyTorch, simulating the corruption of clean images through the incremental addition of Gaussian noise. The Gaussian noise is added according to a predetermined variance schedule β_t. This process demonstrates how an image's data integrity deteriorates over time under the influence of stochastic noise.

The implementation involves defining specific time steps at which the corruption is visualized: t = 0, 10, 50, 100, and 500. For each time step:

- We compute $\sqrt{\alpha_t}$ and $\sqrt{1-\alpha_t}$, where α_t is derived from the variance schedule β_t.
- Gaussian noise is sampled from a standard normal distribution and scaled by $\sqrt{1-\alpha_t}$.
- This noise is added to the original images scaled by $\sqrt{\alpha_t}$, resulting in progressively noisier images.

This results in a collection of noisy images and a series of mean squared error (MSE) values quantifying the noise's impact.

The series of images below demonstrates the visual progression of noise corruption over time. Each row represents one of the five original images, and each column corresponds to a time step in the diffusion process.

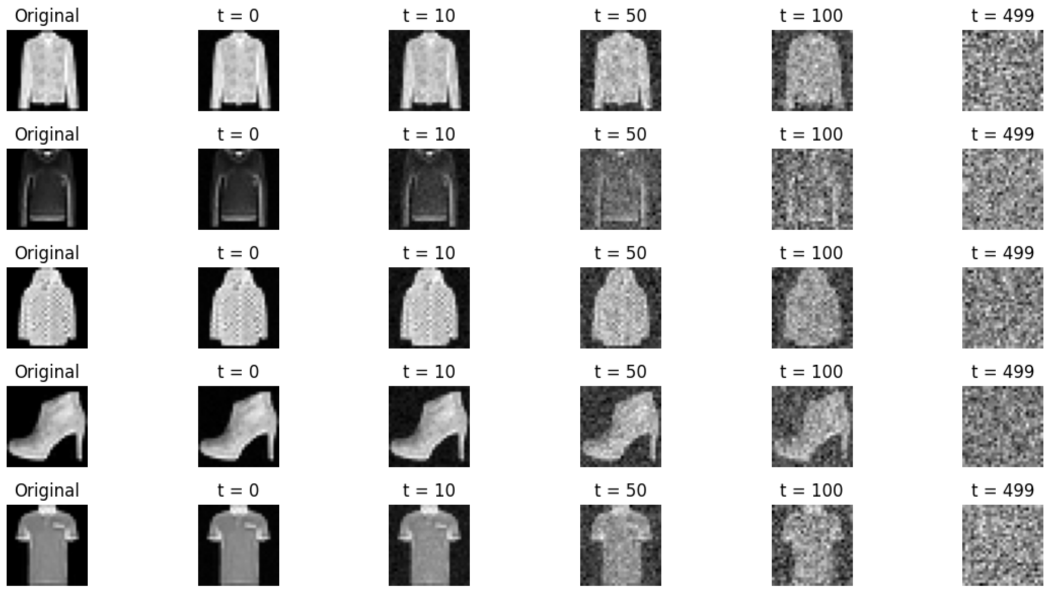

Figure 1: Visual demonstration of image corruption at different time steps in the forward diffusion process. From left to right: original image, and images after 0, 10, 50, 100, and 500 time steps.

We quantify the corruption by calculating the mean squared error (MSE) between the original images and their noisy counterparts at each time step. The plot below shows how the MSE increases as the time step number increases, reflecting the increasing deviation from the original images due to added noise.

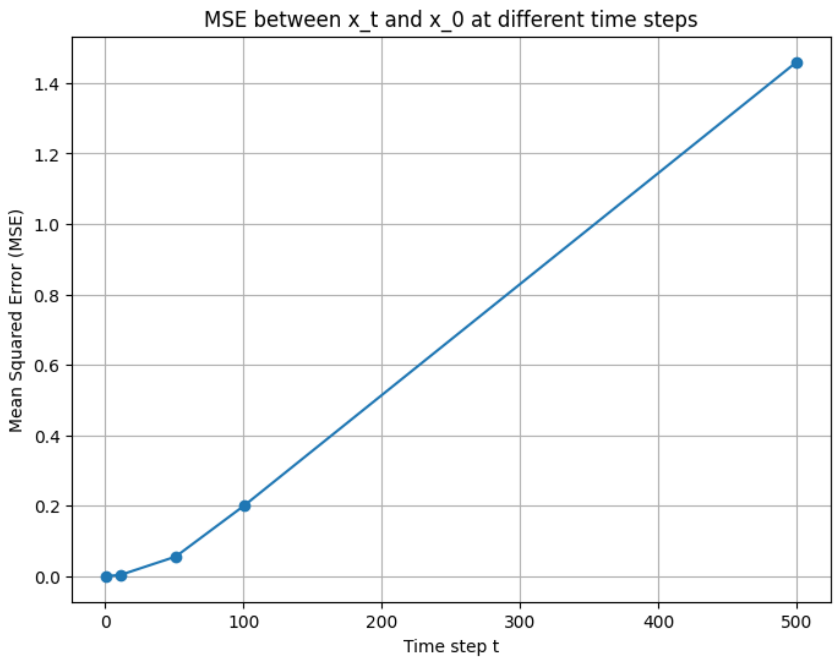

Figure 2: Plot of Mean Squared Error (MSE) as a function of time steps, demonstrating the increase in error corresponding to increased image corruption.

To provide a more detailed analysis, we calculate the MSE for each image at every single time step from 0 to 500. The following plot illustrates the overall trend in the MSE throughout the entire forward diffusion process, providing insights into the corruption dynamics across the entire spectrum of the process.

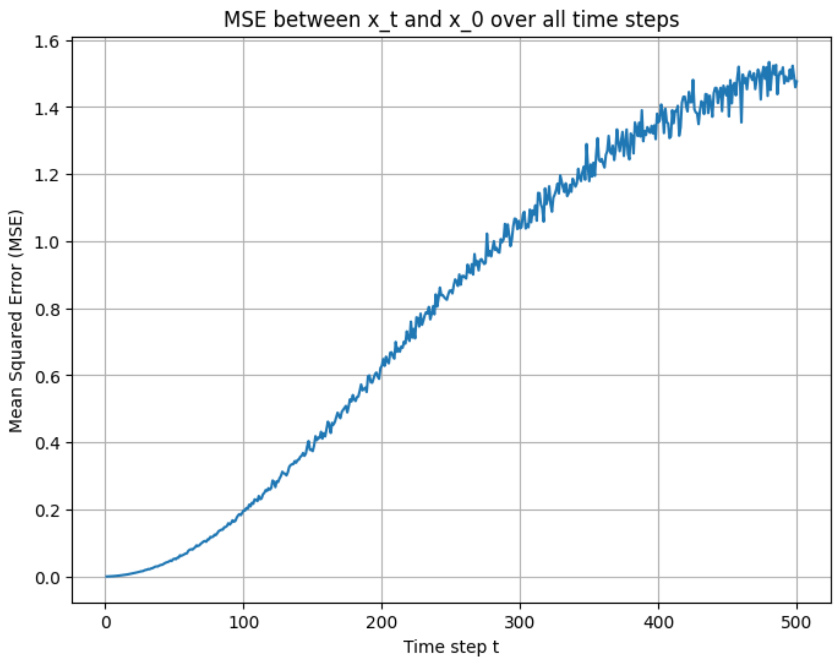

Figure 3: Comprehensive plot of MSE across all time steps, showing the trend of increasing corruption as noise is added progressively.

4. [20 pts] DDIM is a follow-up work to DDPM that speeds up inference by skipping steps. Specifically, it generalizes the forward diffusion process from Markovian to non-Markovian. Whereas DDPM sampled images in 1000 steps, DDIM can sample images with comparable quality in just 50. Check out the DDIM repo (https://github.com/ermongroup/ddim) and play around with the number of steps. Plot images using different sampling steps (e.g. 100, 50, 10). Additionally, the generative process in DDIM is deterministic, so start with a fixed initial state $x_t \sim \mathcal{N}(0, I)$) for a more direct comparison.

   DDIM (Denoising Diffusion Implicit Models) offers an accelerated and deterministic approach to image generation, compared to its predecessor, DDPM (Denoising Diffusion Probabilistic Models). While DDPM typically operates over a sequence of 1000 sampling steps, DDIM significantly reduces this requirement, allowing for high-quality image sampling in as few as 50 steps.

   In our code, we employed the CIFAR-10 model pre-trained on the DDIM architecture to generate images at various sampling steps. Specifically, we compared outputs at 100, 50, and 10 steps as per the instructions (with a few extra steps I added). This range was selected to investigate the model's performance at higher speeds and to see how decreasing the number of steps affects the visual quality of generated images.

   The results below show the model's capability to produce discernible/coherent images even at reduced step counts, highlighting DDIM's efficiency. However, as the number of steps decreases, subtle details and color consistency in the images begin to degrade slightly, showcasing the trade-off between speed and accuracy.

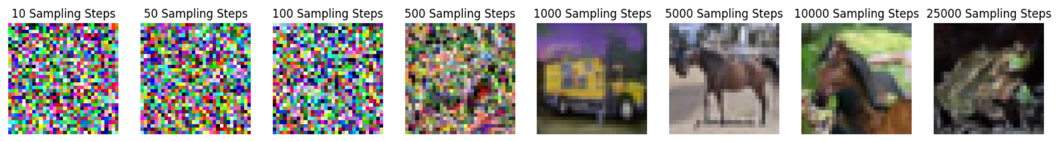

Figure 4: Comparison of images generated by the DDIM model at different sampling steps. The images reveal the trade-off between generation speed and image quality.

5. [20 pts] There are many generative applications based on diffusion models and we encourage you to try them out. Here we provide an example of this, where you can follow the on google colab: https://github.com/google/prompt-to-prompt/. You are also welcome to look around and find a specific paper/project that interests you and reproduce results. In your submission, other than selecting your own images and changing some of the inputs (e.g. text inputs), briefly explain what surprised you with these methods, and what you think they are lacking.

   We implemented the `prompt-to-prompt_stable.ipynb` notebook from the repository `https://github.com/google/prompt-to-prompt`. By using the different text inputs given, we explored how the model manipulates images based on prompt modifications.

   - **Original Prompt:** A painting of a squirrel eating a burger
   - **Modified Prompt:** A painting of a lion eating a burger

   The generated image successfully replaced the squirrel with a lion while preserving the rest of the scene, including the burger. The style and composition remained consistent between the two images.

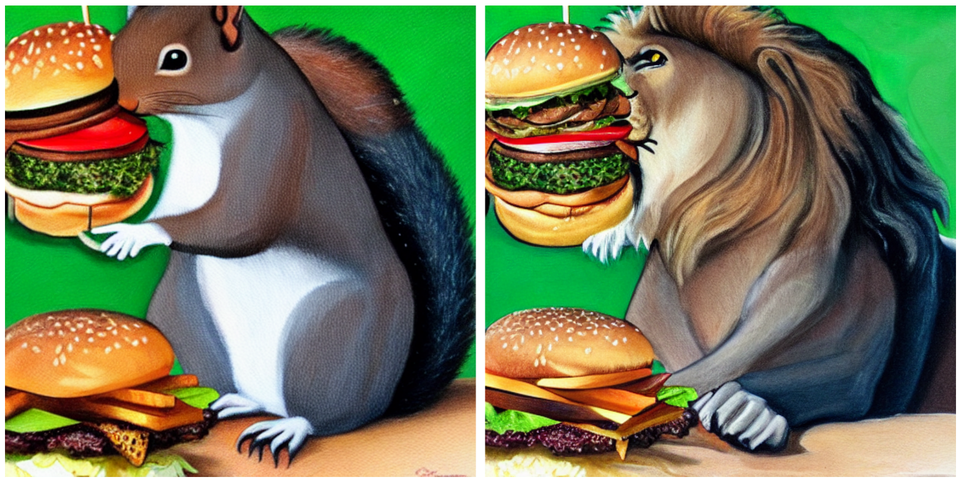

Figure 5: Left: Painting of squirrel eating burger. Right: Painting of lion eating burger.

   - **Original Prompt:** A photo of a house on a mountain
   - **Modified Prompt:** A photo of a house on a mountain at winter

   The model adeptly changed the environment to depict a winter scene, adding snow to the mountain and surroundings while keeping the house consistent.

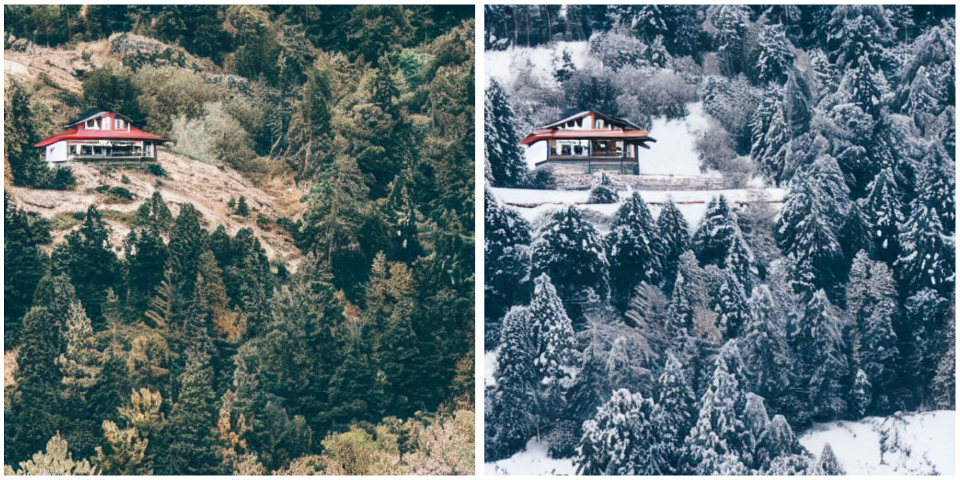

Figure 6: Left: Photo of house on mountain. Right: Photo of house on mountain in winter.

Google's model allowed for precise alterations in the image corresponding to specific changes in the prompt. This level of control is impressive, especially since it does not require manual intervention or additional training. The style and aesthetic of the images remained consistent even after modifying the prompts, which was beneficial for maintaining visual coherence in a series of images. Despite the complexity of the task, the images are generated relatively quickly (at least, on my GPU), showcasing the efficiency of diffusion models in handling prompt-based edits.

The model sometimes struggled with prompts that involved complex interactions between multiple objects. For example, ensuring the lion interacts naturally with the burger can be challenging. Edits involving abstract or less concrete concepts, such as emotions or metaphors, are less successful. While the main elements are preserved, some fine details might be lost or altered. In the lasagne (lasagna) example, the texture of the lasagne might not be as detailed as expected. The quality of the output heavily relies on the specificity and clarity of the prompt. Ambiguous or complex prompts can lead to less accurate images.

The prompt-to-prompt method demonstrates a powerful capability in editing images through textual modifications, giving us a high degree of control and consistency. It excels in straightforward substitutions and environmental changes while maintaining stylistic elements. But, it exhibits limitations with complex object interactions, abstract concepts, and preserving fine details. Future improvements could focus on enhancing the model's understanding of nuanced prompts and improving the preservation of intricate image features.

For more details, code, and relevant citations, please see the Github.