Score-Based Generative Modeling through Reverse Diffusion Monte Carlo

Ethan Vertal
Statistics and Machine Learning
Carnegie Mellon University
Pittsburgh, PA 15213
evertal@andrew.cmu.edu

Mudita Sai
Statistics and Machine Learning
Carnegie Mellon University
Pittsburgh, PA 15213
mksai@andrew.cmu.edu

Supriya Shingade
Statistics and Machine Learning
Carnegie Mellon University
Pittsburgh, PA 15213
sshingad@andrew.cmu.edu

1 Introduction

We will perform class-conditional image generation using MNIST data with the intention of comparing experimental results of these new methods to the standard Denoising Diffusion Probabilistic Models or shortly, DDPM (Ho et. al., 2020) and standard score-based models. We propose a novel implementation of a Reverse Diffusion Monte Carlo (rdMC) sampler and re-implementation of Score-based Generative diffusion models via stochastic differential equations. Our hope is that the Monte Carlo sampling-based approximation methods will achieve near or better performance on this task, as measured by FID metric and the Fréchet Inception Distance (FID). The rdMC scored a 383.03 on the FID scale and averaged 10.083 bits/dim which is relatively poor in performance compared to the other score-based generative models.

2 Dataset and Task

We used the standard 60,000-image MNIST Digit dataset as our final training and test data. We started in our midway report working with a 5,200 image subset of the ImageNet Large Scale Visual Recognition Challenge (ILVRC) dataset, using data subsetted from a single class, cats, to evaluate the model's performance on a singular class-level basis. It turned out that our experimental models were too computationally expensive to permit use of colored, more detailed images for our resources, more on this in section 8. That being said, our final task focused on generating accurate digit images based on the MNIST data and minimizing both the bits/dim and FID metric[1]. Our main research questions were:

1. Will Reverse Diffusion Monte Carlo hold its own against similar state-of-the-art diffusion models for generating images?
2. How does the rdMC compare with other score-based reverse diffusion processes in terms of computational efficiency?
3. What limitations does this new model have?

[1]Our original metric also proved too expensive and inefficient for the time we had left. More on this in section 8.

3 Related Work

There exist four broad principal categories of generative models (Song et. al., 2021). The first of which are GANs that perform in a min-max adversarial fashion. Secondly, Variational Autoencoders (VAEs), autoencoders, and normalizing flows belong to the category rooted in likelihood-based approaches. The third category encompasses energy-based modeling, wherein the distribution is modeled as an energy function that is subsequently normalized. The final category includes score-based matching where the score of the energy based model is learned as a neural net. Current state-of-the-art diffusion models, such as DDPMs, leverage denoising score matching with annealed Langevin dynamics and a parameterized Markov chain trained via variational inference to generate samples approximating the underlying data distribution (Ho et al., 2020). This falls under our second category of generative models. However, it is noteworthy that these various generative models more or less make use of similar objective functions. Current literature in diffusion models showcase three equivalent objectives that optimize the model by learning a neural network to 1. predict the original image x_0, 2. predict the source noise ϵ_0 and 3. compute the score of the image, $\nabla \log p(x_t)$ (Luo, 2022). Nonetheless, a crucial element in reversing the learned diffusion model hinges upon the efficiency of convergence from any complex distribution to a normal distribution. Usually the probability distribution of our data, $p(x)$ can be defined as follows:

$$p_{\boldsymbol{\theta}}(x) = \frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(x)}$$

where $f_{\boldsymbol{\theta}}(x)$ is the energy function which can be modeled by a neural network and $Z_{\boldsymbol{\theta}}$ is a normalizing constant to ensure that $\int p_{\boldsymbol{\theta}}(x)dx = 1$. $Z_{\boldsymbol{\theta}}$ is intractable especially for complex distributions but score-based generative modeling completely avoids the need to learn this normalizing constant. By using a neural net to learn the score function $s_{\boldsymbol{\theta}}(x)$ we bypass the need to use $Z_{\boldsymbol{\theta}}$. The score function is defined as follows:

$$\begin{aligned}
\nabla_x \log p_{\boldsymbol{\theta}}(x) &= \nabla_x \log(\frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(x)}) \\
&= \nabla_x \log \frac{1}{Z_{\boldsymbol{\theta}}} + \nabla_x \log e^{-f_{\boldsymbol{\theta}}(x)} \\
&= -\nabla_x f_{\boldsymbol{\theta}}(x) \\
&\approx s_{\boldsymbol{\theta}}(x)
\end{aligned}$$

where $s_{\boldsymbol{\theta}}(x)$ is the score function computed from an arbitrary time-dependent neural network that minimizes the Fisher Divergence as defined:

$$\frac{1}{2}\mathbb{E}_{p(x)}\left[\|s_{\boldsymbol{\theta}}(x) - \nabla \log p(x)\|_2^2\right] \tag{1}$$

But the Fisher divergence cannot be directly computed, since the score of the data distribution $\nabla \log p(x)$ is unknown. Fortunately, score matching eliminates the data score using integration by parts and so our new score matching objective (which is equivalent to the above objective function) can be defined as follows:

$$\mathbb{E}_{p(x)}\left[tr(s_{\boldsymbol{\theta}}^2(x)) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(x)\|_2^2\right] + C \tag{2}$$

where minimizing the above eq. only requires knowledge of $s_{\boldsymbol{\theta}}(x)$. Once the score function is learned, samples can be generated starting from a random point and iteratively following the score through a process known as Langevin Dynamics.

$$\mathbf{x}_{i+1} = \mathbf{x_i} + c\nabla \log p(\mathbf{x}_i) + \sqrt{2c}\epsilon \tag{3}$$

The usual process is to randomly sample from the from the prior distribution with some additive noise $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})$ which prevents the model from deterministically converging to the mode and getting trapped in a local minima. However, Langevin Dynamics is prone to a multitude of problems such as when x is a low dimension manifold in a high dimensional space or when the score is not accurate in low density regions of the distribution or getting stuck a local minimas. The solution to these problems all involve adding multiple levels of Gaussian noise which allows the data sample to not be confined to a local minima/low dimensional manifold which gives rise to a method called

Annealed Langevin Dynamics (Song and Ernon 2020, [1]). To train the neural network $\mathbf{s}_\theta(\mathbf{x}, \sigma)$ we will use a noise conditional score network (Song and Ernon [2]). Thus the resulting neural net is time-dependent and noise-conditional which is as follows:

$$\arg\min_\theta \sum_{t=1}^{T} \lambda(t)\mathbf{E}_{p_{\sigma_t}}(x_t)[\|\mathbf{s}_\theta(\mathbf{x}, t) - \nabla \log p_{\sigma_t}(\mathbf{x_t})\|] \tag{4}$$

where $\lambda(t)$ is a positive weighting function that conditions on noise level t. To generalize this process to infinite number of time steps (and thereby accounting for an infinite perturbations) Stochastic Differential Equations come into play. The images are sampled simply by reversing the SDE. In a similar vein, the reverse diffusion Monte Carlo (rdMC) algorithm which also samples data from an unnormalized distribution is designed in such a way to incorporate score-based modeling (Dong et. al., 2024). But the rdMC diverges from conventional Langevin MCMC techniques in utilizing a reverse Stochastic Differential Equation (SDE), specifically the Ornstein-Uhlenbeck (OU) process, which is also able to represent a time-homogeneous continuous-time Markov process observed at non-uniformly sampled discrete times (Santos, Lin, 2024).

4 Approach

Since the forward process of a diffusion model has no learnable parameters and can be computed efficiently in one pass with the help of the reparametrization trick, our focus will be on the more interesting reverse process which in essence is the denoising process and can be implemented in a multitude of ways. This section will provide a brief introduction to the inspiration for this project in the Denoising Diffusion Probablistic Model, by Ho et. al. 2020, and our baseline algorithm in Score-Based Generative Models, as discussed by Song & Ermon 2020. Lastly, we talk about our main, novel implementation of the Reverse-Diffusion Monte Carlo sampler proposed by Huang et al. in 2024. In our final approach, we only used the Score-Based Generative Models and the rdMC sampler in our comparisons.

4.1 Denoising Diffusion Probablistic Model

Our absolute baseline model is the traditional Denoising Diffusion Probablistic Model, presented in the landmark paper by Ho et. al. in 2020. This approach involves training the model via minimization of the negative log-likelihood of $p_\theta(\mathbf{x})$. In other words, we train it via Maximum Likelihood Estimation, but since $p_\theta(\mathbf{x})$ is generally intractable, we maximize the Evidence Lower Bound (ELBO):

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} \right]. \tag{5}$$

Much of this approach is discussed in Section 3. However, it is important to recognize that the training process in this approach involves sampling from a reparameterization of a Gaussian distribution, passing that sample through a model that learns how to apply noise to these samples to replicate $\mathcal{N}(0, \mathbf{I})$, and taking the L1 loss of the model output with a random sample from said distribution. Then, using this newly trained noise model, we input noise and transform the model output via the algorithm found in Ho et. al., 2020, where they use a UNet to model the learned noise from the training process.

4.2 Score-Based Generative Modeling

As discussed in section 3, a score-based generative model differs from a DDPM in that it uses the UNet to learn the score function $s_\theta(x) \approx \nabla_x \log p_\theta(x)$ rather than the noise in the forward process. It is trained similarly to how we train DDPM, by taking the loss between the output of our UNet and Gaussian noise, but we perform perturbations on the data passed into the UNet before loss calculation to match the general forward SDE

$$d\mathbf{x}_t = f(\mathbf{x}_t, t)dt + g(t)d\mathbf{w}, \tag{6}$$

with $f(\mathbf{x}_t, t)$ as the drift coefficient and $g(t)$ being the diffusion coefficient. The reverse SDE for this would be

$$d\mathbf{x} = \big(f(\mathbf{x}, t) - g(t)^2 \nabla_x \log p_t(\mathbf{x})\big)\, dt + g(t) d\mathbf{w}. \tag{7}$$

So, to solve the reverse diffusion SDE, we use various drift and diffusion coefficients in place of f and g.

4.3 Reverse-Diffusion Monte Carlo

Rather than reparameterizing the Gaussian distribution as in the DDPM method, we are interested in using Monte Carlo Methods to perform the reverse diffusion process. And as such, we will try to implement the new reverse diffusion Monte Carlo (RdMC) sampling technique. RdMC recognizes that the reverse process is a specific Stochastic Differential Equation called an Ornstein-Uhlenbeck process, which also models the noise from a standard normal to the target distribution.

$$d\mathbf{x}_t = -\mathbf{x}_t dt + \sqrt{2} dB_t, \quad \mathbf{x}_0 \; p_0 \propto e^{f_*} \tag{8}$$

where B_t is a Brownian motion term and f_* represents the negative log-likelihood of our target distribution $p_0 = p_*$. We use the fact that this is an OU process to form the reverse equation as

$$\tilde{x}_{t+s} = e^s \tilde{x}_t + (e^s - 1) v_k + \mathcal{N}(0, (e^{2s} - 1) I_d), \text{ where } v_k \text{ is a variable for } \nabla_x \ln p_{T-t}$$

So what we have now is that,

$$v_k \approx \nabla_x \ln p_{T-t}(x) = \mathbb{E}_{x_0 \sim q_{T-t}} \left[\frac{e^{-(T-t)} x_0 - x}{(1 - e^{-2(T-t)})} \right],$$

$$q_{T-t}(x_0 | x) \propto \exp\left(-f_*(x_0) - \frac{\| x - e^{-(T-t)} x_0 \|^2}{2(1 - e^{-2(T-t)})} \right).$$

And now, using a sampling method similar to the Euler-Maruyama method we plan to implement as a simpler model, our algorithm from Huang et. al. is as shown in Algorithm 1. Our main goal here is still to estimate $v_k = \nabla_{\mathbf{x}} \log p(\mathbf{x}_t)$, but instead of directly approximating it with a UNet as in Huang et. al., we approximate v_k via taking the sample mean of n samples using the Unadjusted Langevin Algorithm to estimate its mean. Using the notation Huang et. al. and algorithms introduced in their 2024 paper that this approach is based off of, let $\{x_k^{(i)}\}_{i=1}^n$ represent samples taken in the ULA inner loop, provided in Algorithm 2 below.

Algorithm 1 RDMC: reverse diffusion Monte Carlo

1: **Input**: Initial particle $\tilde{x}_0$ sampled from $\tilde{p}_0$, Terminal time T, Step size η, η', Sample size n.
2: **for** $k = 0$ to $\left\lfloor \frac{T}{\eta} \right\rfloor - 1$ **do**
3: Set $v_k = 0$;
4: Create n Monte Carlo samples to estimate

$$v_k \approx \mathbb{E}_{x \sim q_{T-t}} \left[\frac{\tilde{x}_{kn} - e^{-(T-k\eta)} x}{(1 - e^{-2(T-k\eta)})} \right]$$

5: $\tilde{x}_{(k+1)\eta} = e^{\eta} \tilde{x}_{kn} + (e^{\eta} - 1) v_k + \xi$ where ξ is sampled from $\mathcal{N}(0, (e^{2\eta} - 1) I_d)$.
6: **end for**
7: **Return**: $\tilde{x}_{\lceil T/\eta \rceil \eta}$.

Algorithm 2 ULA inner-loop for the $q_t(\cdot|x)$ sampler (Step 4 of Algorithm 1)

Input: Condition x and time t, Sample size n, Initial particles $\{x_0^i\}_{i=1}^n$, Iters K, Step size η.
for $k = 1$ to K **do**
  for $i = 1$ to n **do**
    $x_k^i = x_{k-1}^i - \eta \left(\nabla f_*(x_{k-1}^i) + \frac{e^{-t}\left(e^{-t}x_{k-1}^i - x\right)}{1-e^{-2t}} \right) + \sqrt{2\eta}\xi_k$, where $\xi_k \sim \mathcal{N}(0, I_d)$
  end for
end for
Return: $\{x_K^i\}_{i=1}^n$.

5 Experiments

Our experiments differed from the Midway Executive Report, which can be seen in Appendix A. To answer our research questions, we pretrained the UNet on the MNIST data for 50 epochs. Each epoch took between 20 and 50 seconds with a batch size of 128, a learning rate scheduler that reduced it by 10% on a 2 epoch plateau, and with the Adam optimizer. In the following, we can see that the MSE loss quickly converges around 20, taking about 2-5 epochs to converge on each run.

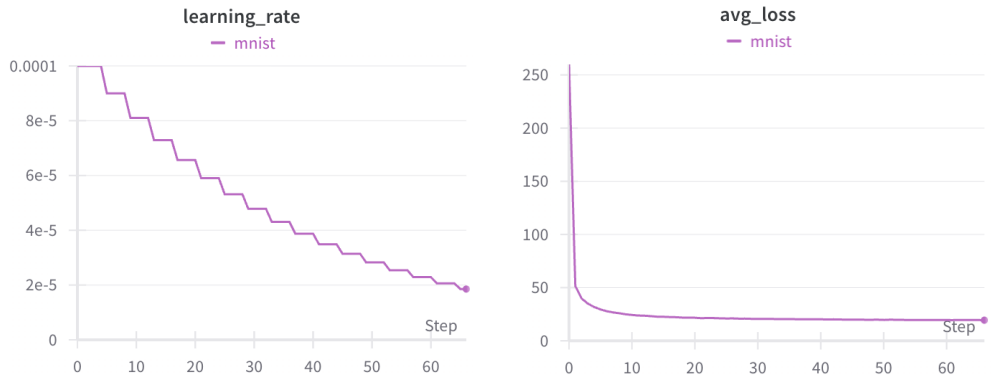


Figure 1: Weights & Biases LR and Loss plot for training UNet on 50+ epochs.

We used Google Colab GPUs to perform training and inference. Particularly, we used three different types of GPU throughout our experimentation process: T4, L4, and V100. Though GPU memory never became a bottleneck, the clock speed and number of FLOPs limited our ability to experiment thoroughly.

Then, we ran each sampler (Euler-Maruyama, Predictor-Corrector, an ODE sampler, and our rdMC implementation) for 1000 steps, with our rdMC method using 200 inner-loop steps for Monte-Carlo mean estimation. The final metrics are included in Tables 1 and 2 below. In Figure 2 (next page), we can see the images our samplers generated.

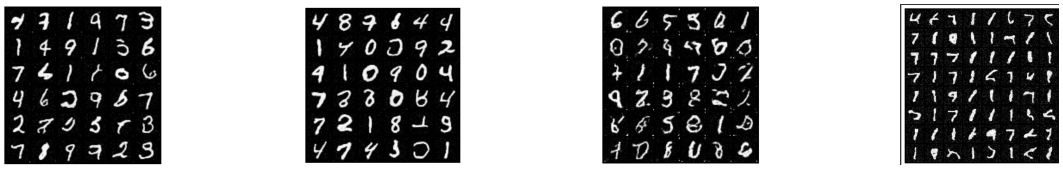
Model	**bits/dim**
Baseline (ODE Sampler)	4.09
RDMC	10.85

(a) **Table 1:** Comparison of models based on likelihood (bits/dim) score, lower is better

Model	**FID**
Euler-Maruyama	159.42
Predictor-Corrector	138.33
ODE Sampler	142.55
RDMC	383.03

(b) **Table 2:**Comparison of models based on FID score, lower is better

Unfortunately, our model yielded unsatisfactory results, achieving an FID score of 383.03. The top-performing model, the predictor-corrector sampling algorithm, achieved a marginally better FID score of 138.33, although it still remains relatively high. Additionally, when considering bits/dim, the



(a) EM (b) PC (c) ODE (d) RDMC

Figure 2: Generations from each sampler after 1000 diffusion steps.

rdMC method averaged around 10.85, whereas the baseline model averaged around 4.09, aligning consistently with our FID metrics.

6 Code Overview

Our code defines a function called `rdmc_sampler` which implements a sampling algorithm. We first initialize samples from a Gaussian distribution with mean 0 and standard deviation based on the marginal probability standard deviation from our transition kernel. A sequence of time steps from 1 to `eps` with `num_steps` is used and a step size based on the time steps is also created. The sampling process comes from looping over each time step where at each step, it computes a score using the time-dependent `score_model`, adds noise from the distribution $\sqrt{2\eta}\xi_k$, where $\xi_k \sim \mathcal{N}(0, I)$, and updates the samples accordingly. Inside the loop, there's another inner loop (ULA) which calculates `v_k`, the score function estimator, averaged over n iterations. At the end of each time step, x is updated using this `v_k` by a certain step size. The algorithm returns the denoised version of x, which our algorithm defines as `mean_x`.

7 Timeline

Table 2: Project Tasks Timeline

Task	Project Phase	Total Time Spent (hrs)	Team Members
Reading Papers	All Phases	10	All
Reading Docs	Midway, Final	2	All
Modifying DDPM	Midway	2	Ethan
Modifying SDE Samplers	Midway, Final	5	All
Implementing RDMC	Final	18	All
Writing Proposal	Proposal	4	All
Writing Midway Report	Midway	4	All
Writing Final Report	Final	4	All
Poster Content	Final	1	All
Planning SDE, RDMC Imp.	Midway, Final	4	All
Training UNet	Final	6	All
Sampling SDE/RDMC	Final	10	All
Reading related works	All Phases	6	Supriya, Mudita
Writing Metric Code	Final	9	Supriya, Mudita
Learning SDE/sampler math	All Phases	12	All

8 Research Log

To begin, we all felt that our focus would be on image generation algorithms. We all wanted to work with them since they were our favorite parts of the course to experiment with. We narrowed down on diffusion and discovered the paper for Reverse-Diffusion Monte Carlo, which was released this year

in 2024, so we decided it would be a fun and ambitious project to implement this newly proposed algorithm.

We frequently encountered unanticipated issues throughout the research process. First, the three of us were unfamiliar with differential equations. We all consulted blogs, papers, and YouTube videos to understand what a stochastic differential equation did and what forms it could take (the foundation of our approach). We then worked on developing a basic understanding of the RDMC algorithm and similar Score-Based Generative Models to begin implementation. This took longer than expected due to the combination of our lack of deep differential equation experience and the rdMC paper having confusing notation.

Once we felt confident enough to begin implementing, we quickly learned that the rdMC paper was lacking details that made this process harder than we had hoped. Specifically, many variables' first appearance was in the two algorithms in section 4, without explanation of where they came from or what they represent. We experimented with different implementations, tweaking the code dozens of times over the course of days. During this time, we also discovered that, despite our best efforts to vectorize our code, runs were taking a minimum of 20 minutes to complete. Further, when our members ran out of compute units on Google Colab, debugging became extremely difficult during pair-programming.

We regrettably left computing metrics as one of our last tasks in the coding process, since our time was consumed by debugging the rdMC sampler. We settled on an existing likelihood metric, bits/dim, and determined that this would be our metric of interest going forward to ensure we had a metric to report. Towards the end, we got our FID metrics working after waiting hours after the poster's due date for our model to take enough samples for the embedding dimension of clean-fid's implementation.

9 Conclusion

In this study, we explored the capabilities of the Reverse Diffusion Monte Carlo (rdMC) method with respect to score-based generative modeling. Our results indicate that the rdMC is not superior to established models in terms of efficiency, accuracy, and performance metrics such as bits/dim. Introduces a novel approach to managing the score function in generative modeling, but our findings suggest that it takes a longer time to reach convergence. This is particularly evident in the fact that our quickest implementation took exponentially longer than the current standard of score-based generative modeling.

Future work should focus on refining the implementation of rdMC to enhance its efficiency and effectiveness. Potential improvements include optimizing the network architecture, exploring more efficient sampling implementations, and exploring the model's performance in a higher-performance compute environment than we had available to us. By advancing the rdMC's capabilities, we hope it eventually achieves or surpasses the performance levels of current state-of-the-art generative models. More research into different score-models to learn how to better learn the score functions, especially modeling the reverse process and hyperparameter tuning on the score-model and sampling methods would go a long way in testing whether this new algorithm is in fact a viable competitor to current diffusion models.

Bibliography

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising Diffusion Probabilistic Models (arXiv:2006.11239). arXiv. http://arxiv.org/abs/2006.11239

Huang, X., Dong, H., Hao, Y., Ma, Y.-A., & Zhang, T. (2024). Reverse Diffusion Monte Carlo (arXiv:2307.02037). arXiv. http://arxiv.org/abs/2307.02037

Luo, C. (2022). Understanding Diffusion Models: A Unified Perspective (arXiv:2208.11970). arXiv. http://arxiv.org/abs/2208.11970

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-Based Generative Modeling through Stochastic Differential Equations (arXiv:2011.13456). arXiv. http://arxiv.org/abs/2011.13456

Santos, J., Lin, Y. (2023). Using Ornstein–Uhlenbeck Process to Understand Denoising Diffusion Probabilistic Model and its Noise Schedules (arXiv:2311.17673). arXiv. https://arxiv.org/abs/2311.17673

[1] Song, Y., Ermon S., (2020). Generative Modeling by Estimating Gradients of the Data Distribution (arXiv:1907.05600). arXiv. http://arxiv.org/abs/1907.05600

[2] Song, Y., Ermon S., (2020). Improved Techniques for Training Score-Based Generative Models (arXiv:2006.09011). arXiv. http://arxiv.org/abs/2006.09011

Po1, R., Yifan1, W., Golyanik2, V., Aberman, K., Barron, J. T., Bermano A., Chan, E., Dekel T., Holynski A., Kanazawa, A., Liu K., Mildenhall, B., Nießner M., Ommer B., Theobalt C., Wonka P., Wetzstein G. (2023). State of the Art on Diffusion Models for Visual Computing (arXiv:2310.07204). arXiv.http://arxiv.org/abs/2310.07204

Song, Yang. "Generative Modeling by Estimating Gradients of the Data Distribution | Yang Song." Generative Space, 5 May 2021, https://yang-song.net/blog/2021/score/.

10 Appendix

10.1 Appendix A - Original Experiments performed for Midway Report

As for experiments with the Score-Based method and rdMC, we will run the models for 20,000 steps with the goal of sampling from our 5,200 cat image dataset. As of now, we have completed training and sampling of these images for our DDPM model. The results can be seen in Figures 1 and 2 below. We started with a learning rate of 1e-3 for 10,000 training steps and then reduced it to 1e-5 for the next 5,000 steps, and reduced it further to 1e-8 for the last 5,000 steps (hence the three loss plots from Weights and Biases).

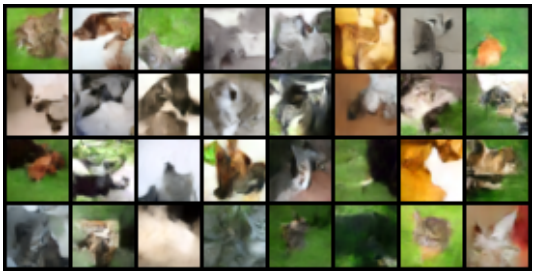
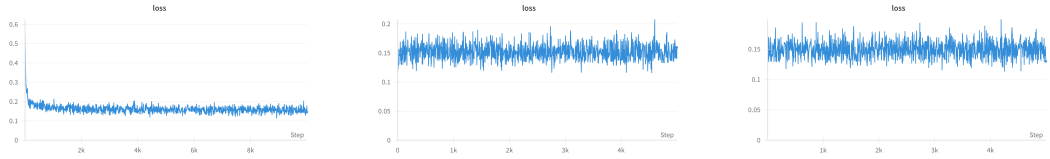


Figure 3: Samples after 20k training steps from DDPM



(a) LR: 1e-3 (b) LR: 1e-5 (c) LR: 1e-8

Figure 4: L1 Loss for all 20k steps, will include more graphs or more lines for other models

Below is a, currently empty, table containing the FID scores for all three models. This will serve as a comparison, but since we have done little experimentation so far we have no results for this yet.

10.2 Appendix B - Code

```
[ ]  #@title Define the RDMC sampler (double click to expand or collapse)

     ## The number of sampling steps.
     def rdmc_sampler(score_model,
                      marginal_prob_std,
                      diffusion_coeff,
                      batch_size=64,
                      num_steps=1000,
                      device='cuda',
                      eps=1e-3,
                      n = 300):

       t = torch.ones(batch_size, device=device)
       init_x = torch.randn(batch_size, 1, 28, 28, device=device) * marginal_prob_std(t)[:, None, None, None]
       time_steps = np.linspace(1., eps, num_steps)
       step_size = torch.tensor(time_steps[0] - time_steps[1], device=device)
       x = init_x

       two_tensor = torch.tensor(2, device=device)

       with torch.no_grad():
         for time_step in tqdm.notebook.tqdm(time_steps):
           batch_time_step = torch.ones(batch_size, device=device) * time_step

           out_x = x
           # Corrector step (ULA)
           v_k = torch.zeros_like(x)
           for i in range(n):
             score = score_model(x, batch_time_step)
             noise = torch.randn_like(x) * torch.sqrt(two_tensor * step_size)
             e_t = torch.exp(-t)[:, None, None, None] * (torch.exp(-t)[:, None, None, None] * (x - out_x))
             e_t = e_t / (1 - torch.exp(-two_tensor*t)[:, None, None, None])
             x = x + step_size * (score + e_t) + noise
             v_k += 2 * ((1 - torch.exp(-2*torch.tensor(time_step)))**-1) * (torch.exp(-torch.tensor(time_step)) * (x - out_x))
           v_k = v_k / n
           mean_x = torch.exp(step_size) * x + ((torch.exp(step_size) - 1) * v_k)
           x = mean_x + torch.randn_like(x) * torch.sqrt((torch.exp(2*step_size)-1))

         return mean_x
```

Figure 5: rdMC Sampling Algorithm