

AIML 425 Assignment 4

James Thompson
Victoria University of Wellington
300680096

I. INTRODUCTION

This document reports the implementation and results of Assignment 4. I look at the differences between the Stochastic Differential Equations and Ordinary Differential Equation models. The SDE model is a score based generative model to learn a mapping from Gaussian noise to dog images. The ODE model is a normalizing flow model that learns a mapping from Gaussian to dogs and cats to dogs. For simplicity I restrict the images to be simple 2D (two pixel) images.

II. THEORY

A flow is a continuous transformation that maps data from one distribution through an iterative process of small changes. In generative modeling, flows provide a powerful framework for learning how to transport samples from a source distribution (e.g Gaussian noise) to a complex target distribution (cat images) over time $t \in [0, 1]$. The key difference between an SDE and ODE is that one is stochastic and the other deterministic.

We are not interested in a transformation of a single point but rather transformations of the distribution of probability mass. A single particle is modeled with a stochastic differential equation:

$$dx = a(x, t)dt + b(x, t)dW \quad (1)$$

where x is the state of the particle, $a(x, t)$ is the drift coefficient (a vector field), $b(x, t)$ is the diffusion coefficient (a scalar), and dW is a Wiener process increment (Gaussian noise). The first term represents deterministic drift, while the second term accounts for stochastic diffusion.

When we have many particles moving according to this SDE, the distribution of their positions evolves over time. This is known as the probability flux and can be described with the Fokker-Planck equation.

$$\frac{\partial p(x, t)}{\partial t} = -\nabla_x \cdot (a(x, t)p(x, t)) + \frac{1}{2}\nabla_x^2 (b^2(x, t)p(x, t)) \quad (2)$$

where $p(x, t)$ is the probability density at point x and time t and the rest is the same as the SDE 1 above.

A. Stochastic Differential Equations

Equilibrium is reached when the probability density no longer changes with time, i.e. $\frac{\partial p(x, t)}{\partial t} = 0$. This happens when the random diffusion balances the deterministic drift. In the context of SDEs, this looks like:

$$0 = -kT\nabla_x \cdot (p(x, t)\nabla_x \log p(x, t)) + kT\nabla_x^2 p(x, t) \quad (3)$$

where k is the Boltzmann constant and T is the temperature. This equation describes how particles in a fluid reach thermal

equilibrium through a balance of drift and diffusion. From this we can see that the score function $\nabla_x \log p(x, t)$ plays a crucial role in determining the drift that counteracts diffusion.

A Stochastic Differential Equation (SDE) generative model [1] is a continuous-time generative model that learns the score function of a data distribution. Therefore the network is aiming to learn the score function $s_\theta(x, t) \approx \nabla_x \log p(x, t)$ where θ are the parameters of the neural network.

Creating the scores is done using a variance-preserving noise schedule where clean target data y is corrupted with Gaussian noise ϵ at time t :

$$x_t = \sqrt{\alpha_t}y + \sigma_t\epsilon, \quad \text{where } \alpha_t = (1-t)^2, \sigma_t = \sqrt{1-\alpha_t} \quad (4)$$

The neural network learns to predict the score function $s_\theta(x_t, t)$ directly, rather than predicting the clean image y and computing the score from it. The training target is:

$$s_{\text{true}} = \frac{\sqrt{\alpha_t}y - x_t}{1 - \alpha_t} \quad (5)$$

This "predict the difference" approach is more effective than predicting the clean image directly, as the network learns to focus on the correction needed rather than reconstructing the entire signal.

During generation, I integrate backward in time from $t = 1$ to $t = 0$ using the Euler-Maruyama method [2]:

$$x_{t-dt} = x_t - \frac{\beta_t}{2}(x_t + 2s_\theta(x_t, t))dt + \sqrt{\beta_t dt}\epsilon \quad (6)$$

where $\beta_t = 2/(1-t)$ is the noise schedule parameter. This process reverses the forward diffusion, guided by the learned score.

B. Ordinary Differential Equations

If we take the Fokker-Planck Equation 2 and remove the equilibrium Equation 3 we get a deterministic flow:

$$\frac{\partial p(x, t)}{\partial t} = -\nabla_x \cdot ((a(x) - kT\nabla_x \log p(x, t))p(x, t)) \quad (7)$$

From this we see that the Fokker-Planck equations with coefficients $(a(x), \sqrt{2kT})$ (SDE) and $(a(x) - kT\nabla_x \log p(x, t), 0)$ (ODE) are equivalent. This means that we can remove the stochastic component and still have the same distribution of particles.

An Ordinary Differential Equation (ODE) model [3] uses this insight and can generate samples from the target distribution. It learns a continuous normalizing flow that transforms one distribution to another via:

$$\frac{dx}{dt} = v_\theta(x, t) \quad (8)$$

where $v_\theta(x, t)$ is a neural network parameterized velocity field that defines how points move over time. Unlike score-

based models, ODE models uses conditional flow matching with linear interpolation. Given samples x_0 from the source distribution and x_1 from the target distribution, we construct training pairs by linear interpolation:

$$x_t = (1 - t)x_0 + tx_1 \quad (9)$$

The true velocity field is simply $v(x, t) = x_1 - x_0$, which the neural network learns to predict. During generation, you integrate forward in time from $t = 0$ to $t = 1$ using the Euler method [2]:

$$x_{t+dt} = x_t + v_\theta(x_t, t) \cdot dt \quad (10)$$

This approach is computationally simpler than SDEs: it doesn't require sampling random noise at each step, making generation deterministic and often faster. A trade off however is that the random generation in SDEs can help "heal" small errors in the learned score function. Furthermore the ODE isn't really generative but rather a bijection while the SDE is truly generative.

C. Performance comparison

The end objective of both models is to learn how to take data from one distribution to another iteratively. The loss shows us the difference between the flows. However to understand the performance we really want to know how well it can generate samples. A simple and effective measure is Maximum Mean Discrepancy (MMD) [4] which allows us to compare the generated samples to the real target distribution. The MMD is a measure of the distance between two distributions based on samples from each distribution. It is defined as

$$\text{MMD}^2(P, Q) = \mathbb{E}[k(x, x')] + \mathbb{E}[k(y, y')] - 2\mathbb{E}[k(x, y)] \quad (11)$$

$$\text{where } k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad (12)$$

$$\text{and } x, x' \sim P, \quad y, y' \sim Q \quad (13)$$

The only parameter in this is the kernel bandwidth σ . The higher the sigma the smoother the kernel so focuses on global differences, where lower sigma focuses on local differences.

Other metrics to understand the performance can be learning efficiency and sample efficiency. This means that we can compare how many samples it takes to learn as well as how much compute time it takes to learn. This can be measured in terms of number of epochs or wall clock time.

III. EXPERIMENTS

A. Data

The data is generated by sampling from the respective distributions. The SDE models train with 100,000 examples while the ODE models train with only 10,000 examples. The Gaussian is with scale 1, Dogs are images in the uniform square with corners (-1,1) and (-2,2) and Cats are images in the uniform square with corners (2,-2) and (3,-3). At dataset creation time the target is computed using either a interpolation technique (flow matching) or a noise corruption technique (score matching). In both methods the Gaussian noise had a scale of 1. We can see examples of the data in Figure 1.

B. Models

The model architecture for both the SDE and ODE is similar. They are all fully connected feed forward networks with 3-4 hidden layers and ReLU activations. The SDE model is larger due to the less efficient training. Complete hyperparameters are found in the appendix Table I.

1) *SDE*: The SDE model is trained to learn the score function which is used to reverse the diffusion process. A visualization of the data can be found in Figure 2. The final performance is satisfactory as judged visually in Figure 3. Furthermore comparing the score field in Figure 4 with the true score field in Figure 2 shows that the model has learned a good approximation of the score function.

2) *ODE*: The ODE model is trained to learn the velocity field which models the flow of data from one distribution to another. To understand the training data we can visualize the interpolation process in Figure 5 for the Gaussian to Dogs and 8 for Cats to Dogs. For both of the mappings the performance is sufficiently good as can be seen in sample generations in Figures 6 and 9. The velocity fields learned by the models are good approximation of the true velocity fields as seen in Figures 7 and 10. Interestingly we can see in the Cat to Dog mapping the score field always points up to the left as that is all the model needs to learn (i.e take point in and move it along the left to right diagonal). This works because the t is bounded to $[0, 1]$ so the model simply learns the direction of travel not the distance it needs to travel. This is noticeably more complex than the Gaussian to Dog case where it need to learn the direction and distance.

C. SDE vs ODE

As discussed above we can quantify the performance of the models using MMD. This works by taking samples from the target distribution and comparing them to samples generated by the model. As MMD is a distance metric the lower the better and we can see that the SDE actually performs slightly better than the ODE model for Gaussian to Dog generation in Table II. It is worth noting however that the ODE model was trained using much less data, less training epochs and smaller model size. When the SDE was trained with similar parameters as the ODE it performance was unsatisfactory. This is due to the fact that the ODE is trained on actual example trajectory which is much faster to learn from then simply noisy image samples.

IV. CONCLUSION

In this assignment I implemented and trained both SDE and ODE generative models. Both models were able to learn the mappings they were trained on, with the SDE model performing slightly better in terms of MMD. However the ODE models were much more efficient to train, requiring less data, smaller models and fewer epochs. This highlights the tradeoff between model complexity and training efficiency in generative modeling.

STATEMENT

The code and report of the Assignment was solely completed by myself (Thompson James). The complete source code can be found here https://gitea.james-server.duckdns.org/james/AIML425_assignment_4, with a link to a colab notebook found here: https://colab.research.google.com/github/ljamesthompson1/AIML425_assignment_4/blob/main/main.ipynb. A complete run through of the notebook takes about 10 minutes on a GPU enabled machine.

I completed my work using the following tools:

- **Jupyter Notebook [5] and JupyterText [6]:** For interactive development.
- **LaTeX:** For writing the report.
- **VSCode [7]:** As IDE, with Copilot to help with plotting and debugging.
- **JAX [8] and Flax [9]:** For implementing the neural network and training logic.
- **Matplotlib [10] and Pandas [11]:** For data visualization and management.

REFERENCES

- [1] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-Based Generative Modeling through Stochastic Differential Equations,” Feb. 2021.
- [2] R. L. Burden, *Numerical Analysis*. Boston, MA : Cengage Learning, 2016.
- [3] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, “Flow Matching for Generative Modeling,” Feb. 2023.
- [4] A. Gretton, K. Borgwardt, M. J. Rasch, B. Scholkopf, and A. J. Smola, “A Kernel Method for the Two-Sample Problem,” May 2008.
- [5] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, “Jupyter Notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87–90.
- [6] M. Wouts, “Mwouts/jupyterx,” Aug. 2025.
- [7] “Microsoft/vscode,” Microsoft, Jul. 2025.
- [8] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, “JAX: Composable transformations of Python+NumPy programs,” 2018.
- [9] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee, “Flax: A neural network library and ecosystem for JAX,” 2024.
- [10] J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [11] The pandas development team, “Pandas-dev/pandas: Pandas.”

APPENDIX

A. Setup

B. Results

The models were trained using the hyperparameters in Table I. They were trained on a RTX A5000 GPU with a Xeon Gold 6128 CPU.

C. SDE: Gaussian to Dogs

D. ODE: Gaussian to Dogs

E. ODE: Cats to Dogs



Fig. 1. Samples from the datasets used for training. The values are normalized to [0, 1] and plotted as images to understand the data.

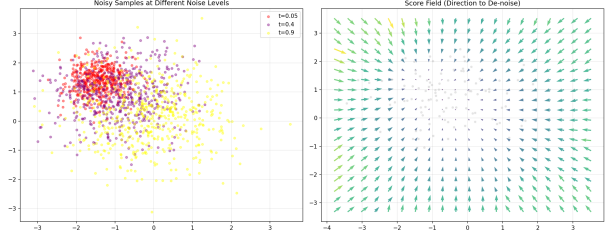


Fig. 2. The forward noise process corrupting dog images over time.

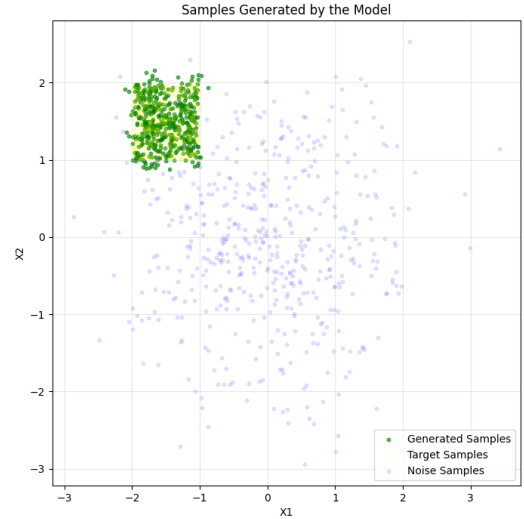


Fig. 3. Sample generations of Dogs from Gaussian noise using the SDE model.

TABLE I
HYPERPARAMETERS USED FOR EACH MODEL

Hyperparameter	SDE	ODE (Gauss to Dog)	ODE (Cat to Dog)
Learning Rate		0.0001	
Minibatch Size	512		256
Hidden Dims	512		256
Hidden Layers	4		3
Num Epochs	500		200
Optimizer		Adam	
Loss Function		MSE	
Generation dt	0.0001		0.001

TABLE II
TRAINING RESULTS

Result	SDE	ODE (Gauss to Dog)	ODE (Cat to Dog)
Runtime (seconds)	273	50	24
MMD	0.000866	0.001344	0.000560
Best validation loss	1.7442	0.4384	0.1322

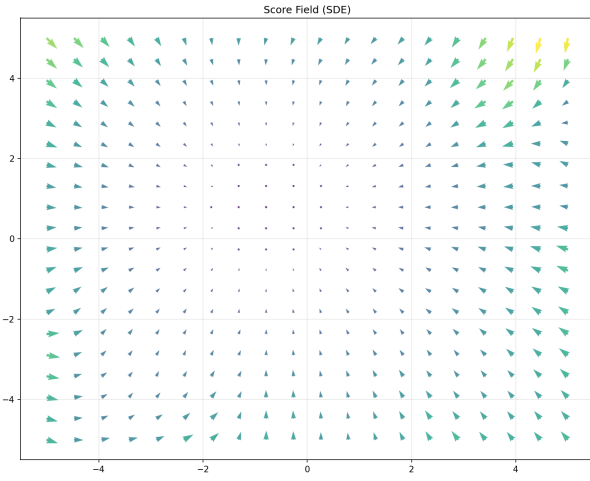


Fig. 4. The score field learned by the SDE model compared to the true score field.

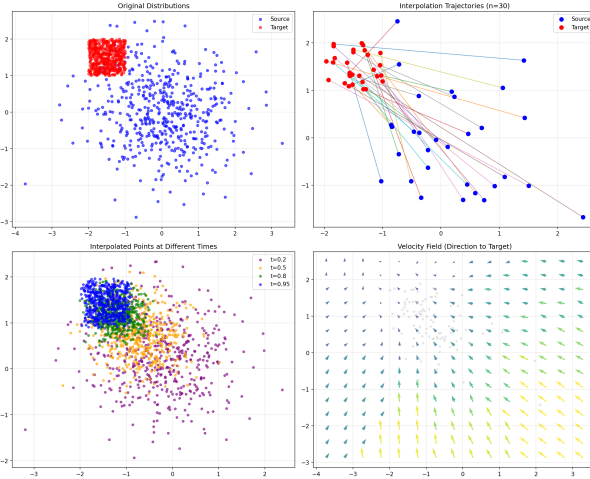


Fig. 5. Linear interpolation between Gaussian noise and Dog images used for training the ODE model.

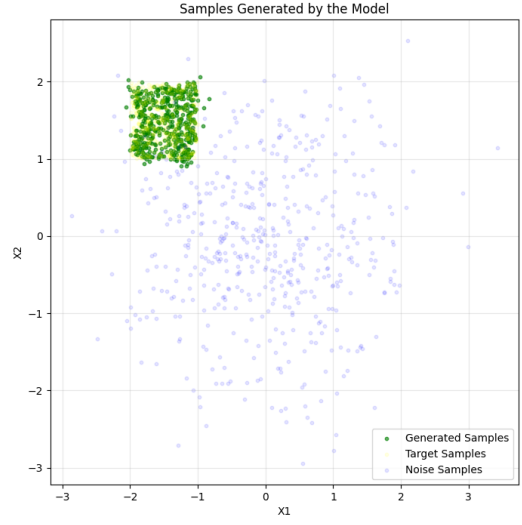


Fig. 6. Sample generations of Dogs from Gaussian noise using the ODE model.

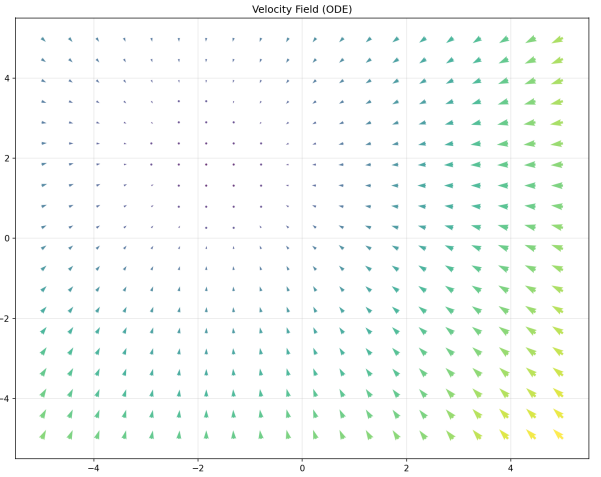


Fig. 7. The velocity field learned by the ODE model compared to the true velocity field.

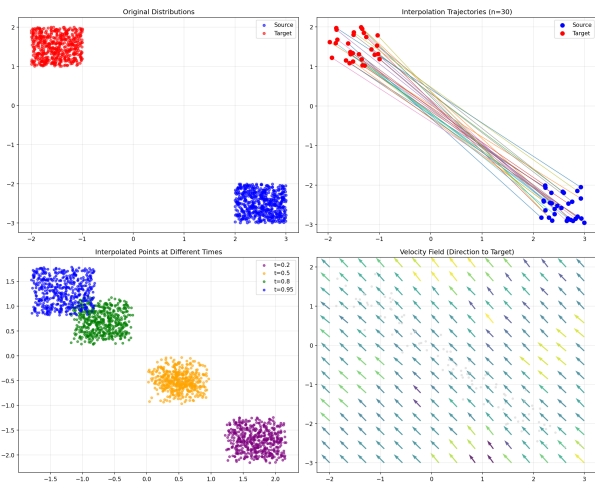


Fig. 8. Linear interpolation between Cat images and Dog images used for training the ODE model.

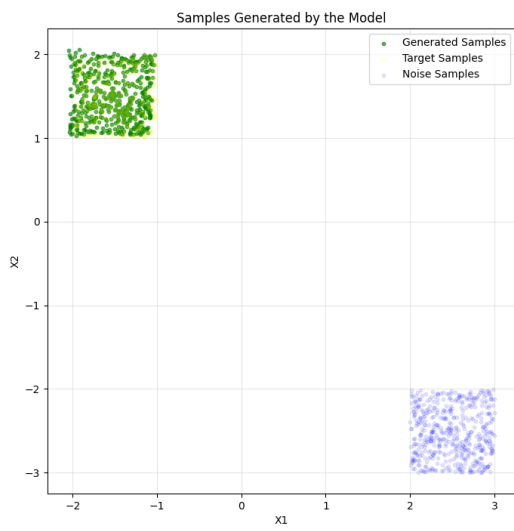


Fig. 9. Sample generations of Dogs from Cat images using the ODE model.

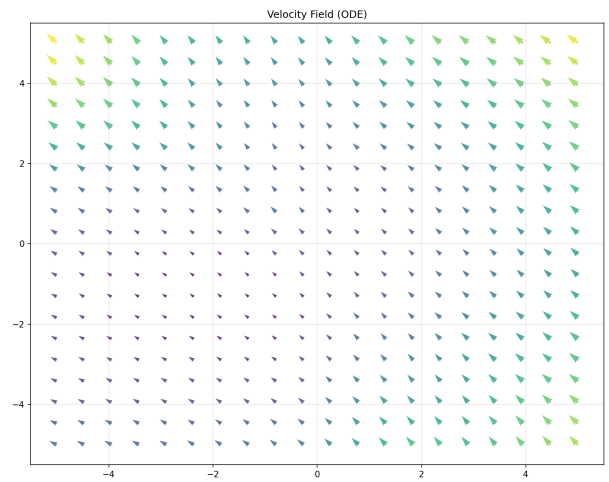


Fig. 10. The velocity field learned by the ODE model compared to the true velocity field.