Logistic-Normal Diffusion on the Probability Simplex

Griffin Floto
Department of Computer Science
University of Toronto
griffin.floto@mail.utoronto.ca

Abstract

Score based stochastic differential equation (SDE) diffusion models learn to reverse the progressive noising of a data distribution to create a generative model and have demonstrated impressive performance across a wide range of tasks. The method relies on a Gaussian distributed Ornstein–Uhlenbeck (OU) process, which can naturally model continuous data in $\mathbb{R}^n$, but does not naturally transfer to discrete data. To apply score based diffusion to discrete data, we propose logistic-normal diffusion which is defined on the probability simplex. Using the probability simplex naturally creates an interpretation where points correspond to categorical probability distributions. We use a process that is the softmax of the commonly used OU diffusion, creating a one to on correspondence between diffusion on $\mathbb{R}^n$ and the n dimensional unit simplex. We find that our methodology also naturally extends to include diffusion on the unit cube which has applications for bounded image generation. Furthermore, the connection between Logistic-Normal and Gaussian Diffusion allows our model to utilize recent advances in SDE score matching models.

1 Introduction

Diffusion models Sohl-Dickstein et al. [2015] Ho et al. [2020] Song and Ermon [2019] have emerged as a well-established class of generative models, finding applications in image Dhariwal and Nichol [2021], speech Jeong et al. [2021], and video Singer et al. [2022] domains. Score-based generative stochastic differential equation (SDE) models Song et al. [2021] are a continuous time diffusion model with additional desirable properties. For example, the methodology allows for flexible reverse time solutions with arbitrary ordinary differential equation (ODE) solvers, as well as exact likelihood evaluation via a connection with probability flow ODEs Chen et al. [2018]. All diffusion models operate by progressively adding noise to data samples, which transforms a complex data distribution into a simpler, easy-to-sample distribution. Parameterized models are then optimized to reverse the noising process and generate new samples from the underlying data distribution.

In comparison to other popular methods, such as Generative Adversarial Networks Goodfellow et al. [2014], and Variational Autoencoders Kingma and Welling [2022], continuous time diffusion models present a compelling advantage as they have an exact likelihood interpretation and do not require adversarial training that other state-of-the-art generative models require. That is, diffusion models enjoy the benefit of having a more stable training process that avoid non-overlapping data and generated distributions [Yang et al., 2023]. Time continuous diffusion models are also advantageous over normalizing flows Rezende and Mohamed [2016] an appealing alternative that is also capable of computing the exact likelihood. Normalizing flows face practical restrictions when computing the determinant of the Jacobian from the change of variables formula , whereas the continuous time interpretation allows for a far more flexible implementation by integrating the ODE divergence through time Chen et al. [2018].

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Most work with diffusion models uses Gaussian distributions, which is natural for data in $\mathbb{R}^n$, however this presents a problem for discrete data distributions. We propose a simple solution by associating n discrete categories with the corners of the n-dimensional probability simplex, and in turn defining a continuous time diffusion process in that space. By shifting from categories themselves, to the space of probabilities over categories, we effectively turn a discrete problem into a continuous one. Given a function σ that is twice-differentiable and a typical Gaussian diffusion process $\mathbf{x}_t$, we can find the solution to the process given by $\sigma(\mathbf{x})$ by using Ito's Lemma. In other words, we simply make use of a map that is twice differentiable between $\mathbb{R}^n$ and the probability simplex, which allows us to use all the continuous time diffusion methodology to train a model on the probability simplex. This is in contrast to a number of recent works which design continuous time diffusion based on the Beta and Dirichlet distributions Zhou et al. [2023], Avdeyev et al. [2023], Richemond et al. [2022], with a more complicated implementation process.

2 Backgound

2.1 Score-Based Generative Modeling with SDEs

Score matching as formulated by Song et al. [2021] is interested in constructing a diffusion process $\{\mathbf{x}_t\}_{t=0}^T$ that is indexed by a continuous time variable t. The goal is to have $\mathbf{x}_0 \sim p_0$ be a dataset of i.i.d. samples, and $\mathbf{x}_T \sim p_T$ be a distribution that is easy to sample from. The diffusion process can be described by an Ito SDE

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\mathrm{d}\mathbf{w} \tag{1}$$

where $\mathbf{w}$ is the standard Wiener process (also know as Brownian motion), $\mathbf{f}(\mathbf{x}, t) : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$ is the drift term and $\mathbf{G}(\mathbf{x}, t) : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^{d\times d}$ is the diffusion matrix. The process maps a data distribution, $p_{t=0}(\mathbf{x}_t) \in \mathbb{R}^d$ into some limiting distribution $p_{t=1}(\mathbf{x}_t)$ that is easy to sample from and independent from the data distribution. This is called the time forward diffusion process, and is typically defined a priori, and not learnt. Classical results in the theory of stochastic processes then tell us that the time reverse of this process is itself an SDE and obeys

$$\mathrm{d}\mathbf{x} = \left\{\mathbf{f}(\mathbf{x}, t) - \nabla \cdot [\mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top] - \mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right\} \mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\mathrm{d}\bar{\mathbf{w}} \tag{2}$$

where time now flows backwards from $t = 1$ to $t = 0$ and $\nabla \cdot \mathbf{G}(\mathbf{x}) := [\nabla \cdot \mathbf{g}_1(\mathbf{x}), \cdots, \nabla \cdot \mathbf{g}_d(\mathbf{x})]^\top$ for a matrix-valued function $\mathbf{G}(\mathbf{x}) = [\mathbf{g}_1(\mathbf{x}), \cdots, \mathbf{g}_d(\mathbf{x})]^\top$. In common implementations like Gaussian diffusion, the diffusion matrix is set to a position independent scalar, which results is the following reverse time sde

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}\mathbf{x} - \frac{1}{2}g(t)^2 \nabla \log p_t(\mathbf{x})\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$$

The goal of diffusion models is to approximate the score $\nabla_x \log\, p_t(\mathbf{x}_t)$ and use the reverse SDE to sample from the generative model. The score can be approximated by $\mathbf{s}_\theta(\mathbf{x}_t, t)$ which provides the following objective

$$\theta^* = \mathrm{argmin}_\theta \mathbb{E}_{t\sim U[0,1]} \mathbb{E}_{\mathbf{x}_0 \sim p_0(\mathbf{x})} \mathbb{E}_{\mathbf{x}_t \sim p_{0t}(\mathbf{x}_t|\mathbf{x}_0)} \lambda(t) \left[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log\, p_{0t}(\mathbf{x}_t|\mathbf{x}_0)\|_2^2\right] \tag{3}$$

where $\lambda(t)$ is a weighting function and $p_{st}(\mathbf{x}_t|\mathbf{x}_s)$ is the transition kernel from $\mathbf{x}(s)$ to $\mathbf{x}(t)$. We note that a number of other objectives can be used to learn the score function Song et al. [2021]. A common practice when using diffusion models is to discretize time into uniform steps, which is equivalent to markov chain probabilistic model approach from Ho et al. [2020].

2.2 The Logistic-Normal Distribution on the Probability Simplex

We would like to define a diffusion process on the unit simplex. Also known as the probability simplex, this is the set of points that sum to 1 and are positive $\left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; \sum_{k=1}^d \mathbf{x}_k = 1,\ \mathbf{x}_k \geq 0\right\}$.

The probability simplex $\mathbb{S}^d$ is a $d-1$ dimensional subset of $\mathbb{R}^d$, given that the final component of vectors on the simplex can be written as $\mathbf{x}_d = 1 - \sum_{k=1}^{d-1} \mathbf{x}_k$. We can then consider the simplex as being fully determined by the set of points that are positive and sum to *less than or equal* to 1:

$$\mathbb{S}^d := \left\{ \mathbf{x} \in \mathbb{R}^{d-1} \;\middle|\; \sum_{k=1}^{d-1} \mathbf{x}_k \leq 1,\ \mathbf{x}_k \geq 0 \right\}.$$

where we can always recover the final redundant component of the vector.

The logistic-normal distribution is an example of a probability distribution over the probability simplex. It is defined as the probability distribution of a random variable whose multinomial logit is a normal distribution, or equivalently it is the distribution of the softmax function applied to a Gaussian. The probability density function of the logistic normal is

$$\begin{aligned} p(\mathbf{x}; \mu, \mathbf{\Sigma}) = & \frac{1}{|(2\pi)^{d-1}\mathbf{\Sigma}|} \frac{1}{(1 - \|\mathbf{x}\|_1) \prod_{i=1}^{d-1} \mathbf{x}_i} \\ & \exp\left(-\frac{1}{2} \left[\log\left(\frac{\mathbf{x}}{1 - \|\mathbf{x}\|_1} \right) - \mu \right]^\top \mathbf{\Sigma}^{-1} \left[\log\left(\frac{\mathbf{x}}{1 - \|\mathbf{x}\|_1} \right) - \mu \right] \right) \end{aligned} \tag{4}$$

where $\mathbf{x} \in \mathbb{S}^d$. In the $d = 2$ dimensional case, the distribution can be understood as mapping a Gaussian distribution on $\mathbb{R}$ to $[0, 1]$ via the sigmoid function.

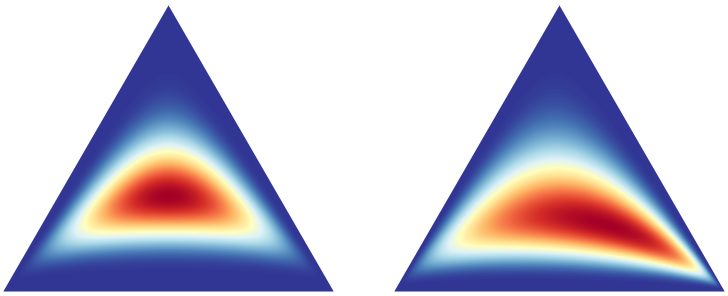


Figure 1: Examples of the Logistic-Normal distribution probability distribution function on $\mathbb{S}^3$ with parameters $\mu = [0, 0],\ [0.2, 0.35]$ and $\sigma = [0.5, 0, 5],\ [0.6, 0.8]$ respectively.

To constructively sample from this distribution, we map a point $\mathbf{z} \in \mathbb{R}^{d-1}$ to a point in the probability simplex $\mathbf{x} \in \mathbb{S}^d$ using the additive logistic transformation $\sigma : \mathbb{R}^{d-1} \to \mathbb{S}^d$ defined by

$$\mathbf{x} = \sigma(\mathbf{z}) := \frac{e^{\mathbf{z}}}{1 + e^{\|\mathbf{x}\|_1}} \tag{5}$$

Conversely, the unique inverse map from $\mathcal{S}^d$ to $\mathbb{R}^{d-1}$ is:

$$\mathbf{z} = \sigma^{-1}(\mathbf{x}) := \log\left[\frac{\mathbf{x}}{1 - \|\mathbf{x}\|_1} \right]$$

The logistic-normal distribution is similar to the more common Dirichlet distribution defined on the probability simplex. It should be noted that these distribution families are never exactly identical for any choice of parameters.

3 Method

3.1 Logistic-Normal Diffusion

To perform score-based generative modelling on the probability simplex, we can use the commonly used Gaussian diffusion and push it forward to the simplex $\mathbb{S}^d$. Typically, Gaussian diffusion is defined as the following OU process on $\mathbb{R}^d$:

$$\mathrm{d}\mathbf{z} = -\frac{1}{2}\beta(t)\mathbf{z}\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}$$

where the function $\beta(t)$ is a user-selection function that controls the signal to noise ratio during the process, and $\mathbf{w}$ is a standard Brownian motion.

Given the observation that $\mathbb{S}^d$ is a $d-1$ dimensional subset of $\mathbb{R}^d$, we would like $\sigma(\mathbf{z})$ to map vectors in $\mathbb{R}^{d-1}$ to the first $d-1$ dimensions of the simplex. We will use the convention where only the first $d-1$ components of $\mathbf{z}$ are written, with the understanding that the last component is always fully determined. We would like the diffusion process on the probability simplex to also take the form of an Ito SDE $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\,\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\,\mathrm{d}\mathbf{w}$. To derive this form, we can use Ito's Lemma to find the forward process SDE for $\mathbf{x}$:

$$\mathrm{d}\mathbf{x}_i = \sigma_i(\mathbf{z}) = \left\{-\frac{1}{2}\beta(t)(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))^\top\mathbf{z} + \frac{1}{2}\beta(t)\,\mathrm{Tr}\left[H_z\sigma_i(\mathbf{z})\right]\right\}\,\mathrm{d}t + \sqrt{\beta(t)}(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))^\top\,\mathrm{d}\mathbf{w}_i \quad (6)$$

where H_z is the Hessian with respect to $\mathbf{z}$. This relationship shows that there are equivalent processes that operate on $\mathbb{R}^{d-1}$ with Gaussian diffusion and $\mathbb{S}^d$ with logistic-normal diffusion. Derivations of these terms are in Appendix A.1. In summary we can write the drift term for the forward time process as:

$$\mathbf{f}(\mathbf{x}) = \frac{1}{2}\beta(t)\left[\mathbf{x} - 2\mathbf{x}^2 + \|\mathbf{x}(1-2\mathbf{x})\|_1\,\mathbf{x} - \frac{1}{\sqrt{\beta(t)}}\mathbf{G}(\mathbf{x}, t)\sigma(\mathbf{x})^{-1}\right]$$

and the diffusion matrix as:

$$\mathbf{G}(\mathbf{x}, t) = \sqrt{\beta(t)}J_{\mathbf{z}}\sigma(\mathbf{z})$$

$$(J_{\mathbf{z}}\sigma(\mathbf{z}))_{i,j} = \begin{cases}\mathbf{x}_i(1-\mathbf{x}_i) & \text{if } \; i = j\\ -\mathbf{x}_i\mathbf{x}_j & \text{if } \; i \neq j\end{cases}$$

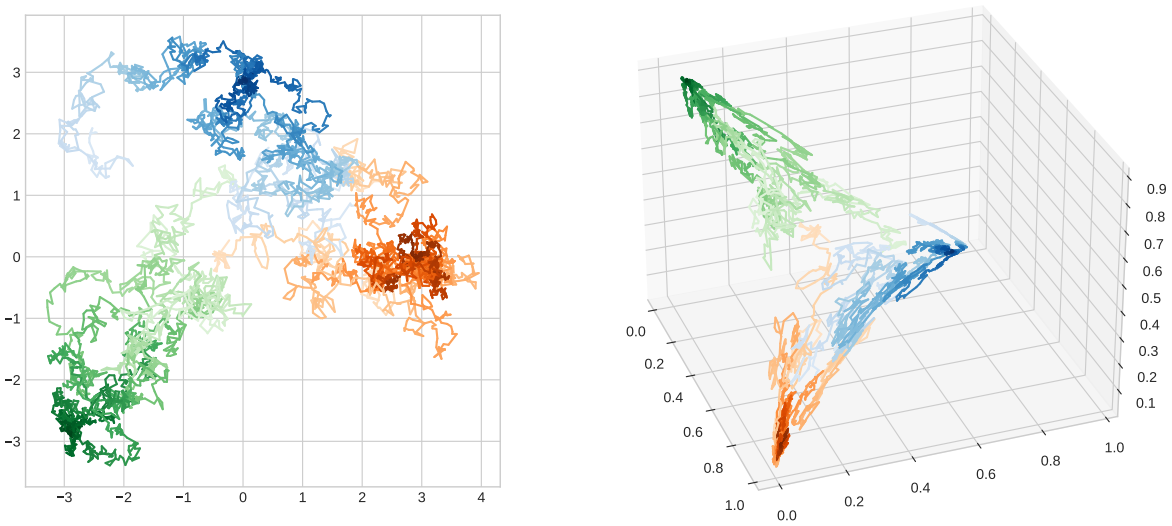
With the definition of the forwards time process on the probability simplex, we can now derive reverse time process $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t - \frac{1}{2}\nabla\cdot[\mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top]\mathrm{d}t - \frac{1}{2}\mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top\nabla_x\log p_t(\mathbf{x})\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\mathrm{d}\bar{\mathbf{w}}$. The full process for this is worked out in Appendix A.2 and summarized below. First, the diffusion matrix divergence can be written as:

$$\nabla_{\mathbf{x}}\cdot[\mathbf{G}_t(\mathbf{x})\mathbf{G}_t(\mathbf{x})^\top] = \beta\mathbf{x}\left[-(d-2)\mathbf{x} + (d+2)\|\mathbf{x}\|_2 - 2\|\mathbf{x}\|_1 + 2\right]$$

and finally the log derivative, or score or the logistic normal distribution:

$$\nabla_{\mathbf{x}}\log p(\mathbf{x}, \mu, v) = \frac{\mathbf{x} - ||\mathbf{x}||_1\mathbf{1}}{||\mathbf{x}||_1\mathbf{x}} - \frac{1}{v||\mathbf{x}||_1}\sigma^{-1}(\mathbf{x}) - \frac{\mathbf{1}^\top[\sigma^{-1}(\mathbf{x}) - \mu]}{v||\mathbf{x}||_1}\mathbf{1}$$

We are then in a position to run identical diffusion processes in either the real space, or the probability simplex.



Gaussian Diffusion Simplex Diffusion

Figure 2: Example of identical forward time diffusion process on $\mathbb{R}^2$ and $\mathbb{S}^3$, where initial points are on the simplex corners, corresponding to 3 discrete categories. The SDE path becomes lighter as time progresses. For visual clarity, only the first 50% of the process is shown.

3.2 Probability Flow and Connection to Neural ODEs

The reverse process for continuous time score based SDE models can be used to compute the exact likelihood of the model. For any diffusion process, there exists a corresponding deterministic process that satisfies an ODE, which in this case is an example of a neural ode Chen et al. [2018] and takes the following form

$$\mathrm{d}\mathbf{x} = \underbrace{\left\{\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}\nabla \cdot [\mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top] - \frac{1}{2}\mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top \mathbf{s}_\theta(\mathbf{x}, t)\right\}}_{\bar{\mathbf{f}}_\theta(\mathbf{x}_t, t)} \mathrm{d}t \tag{7}$$

By using the instantaneous change of variables formula, neural ODEs allow for the exact likelihood to be computed by

$$\log p_0(\mathbf{x}_0) = \log_T(\mathbf{x}_T) + \int_0^T \nabla \cdot \bar{\mathbf{f}}_\theta(\mathbf{x}_t, t)\mathrm{d}t$$

where $\mathbf{x}_t$ is obtained via the solution to the reverse time ODE 7. In practice, we estimate this quantity by using the Skilling-Hutchinson trace estimator $\nabla \cdot \bar{\mathbf{f}}_\theta(\mathbf{x}_t, t) = \mathbb{E}_p(\epsilon)[\epsilon^\top \nabla \bar{\mathbf{f}}_\theta(\mathbf{x}_t, t)\epsilon]$, where $\nabla \bar{\mathbf{f}}_\theta(\mathbf{x}_t, t)$ is the Jacobian of $\bar{\mathbf{f}}_\theta(\mathbf{x}_t, t)$. This estimator is unbiased and can be averaged to obtain an arbitrarily small error. Furthermore, the Jacobian can be obtained via reverse-mode automatic differentiation.

3.3 Categorical Data and Bounded Intervals

Discrete data can be naturally modelled by associated n discrete categories with each corner of the simplex. An example of this can be seen in Figure 2 in the case of $n = 3$ categories. In practice, data points cannot be mapped exactly on the probability simplex boundary do to finite numerical precision. We instead select a small number $\epsilon_b \geq 0$ such that discrete categories are mapped from a

one-hot vector $\mathbf{x}(k)$ to $\left(1-\epsilon_b-\frac{\epsilon_b}{d-1}\right)\mathbf{x}(k)+\frac{\epsilon_b}{d-1}\mathbf{1}$, which data points slightly towards the interior of the probability simplex.

Our model also has the additional benefit that the case of $\mathbb{S}^2$ corresponds to diffusion on the unit interval. This is ideal for image based data, where each pixel is on the interval $[0,1]^3$ for each of the RGB channels. In the typical case when image based diffusion models are used with Gaussian noise, sampling errors often compound and result in pixel values that are outside the valid data range of the unit cube. To mitigate this problem, thresholding is often performed to keep generated images to reasonable values via knowledge of the data distribution constraints Ho et al. [2020] Dhariwal and Nichol [2021]. While thresholding is popular in many image based diffusion models, it is theoretically unsound as there is a disconnect between the training and generative processes. Our methodology provides a natural solution to this problem where the diffusion is constrained to the appropriate data domain.

4 Related Work

The first proposed diffusion model by Sohl-Dickstein et al. [2015] included the modelling of discrete data using a binomial diffusion process. The model uses a probabilistic modelling approach with discrete time intervals. Austin et al. [2021] improved upon this approach by proposing a number of alternative transition kernels. One drawback of this approach is that transition kernels scale quadratically with the number of categories, however solutions such as using low rank matrices to help combat this effect are proposed. Campbell et al. [2022] propose the use of continuous time transition kernels, which allows for the the use of high performance samplers that can out-perform the discrete time methods previously mentioned. A drawback of these methods is that unlike the continuous time SDE approaches, these methods use a bound on the log likelihood and have a more rigid sampling procedure.

Dirichlet Diffusion Avdeyev et al. [2023] is the first continuous time SDE diffusion methodology and uses the connection between the Dirichlet distribution and discrete. The target application for the method is biological sequence generation, however Dirichlet Diffusion can be successfully applied to other problems as well. Avdeyev et al. [2023] use a Jacobi diffusion process to create a diffusion process that is Beta distributed, which is then extended to the probability simplex by using a stick-breaking construction. Sampling from the Jacobi diffusion process is more expensive than the Gaussian case, which was addressed by pre-computing diffused sampled and log transition density gradients.

Categorical SDEs with Simplex Diffusion [Richemond et al., 2022] use a diffusion process of Gamma random variables to sample from a Dirichlet distribution over the simplex. The Dirichlet distribution is an appealing choice as it is the conjugate prior of the categorical distribution . The forward process used is the Cox-Ingersoll-Ross process, which is defined by the SDE $\mathrm{d}\theta = b(a-\theta)\mathrm{d}t + \sigma\sqrt{2b\theta}dw$, where $\theta(t=0) \geq 0$ and $a, b, \sigma > 0$. Simplex Diffusion was attempted to be used for language modelling Dieleman et al. [2022]. Unfortunately, the very high dimensional vocabulary caused an uneven data corruption process which caused problems during training that could not be resolved.

Reflected Diffusion Lou and Ermon [2023] is a method of performing diffusion on the unit cube $[0,1]^d$ that is motivated by applications to pixel-based diffusion models. The authors address this problem by using a reflected diffusion process that reflects particle trajectories into the interior of a data domain Ω that would normally extend outside the domain. The model achieves impressive performance on image generation tasks.

Beta diffusion Zhou et al. [2023] takes an alternative approach to the thresholding problem in image generation. Similar to the Dirichlet and Simplex Diffusion papers, their method results in beta distributions in both the forward marginals and reverse conditionals. The authors take the discrete probabilistic model approach and show that optimization with kl-divergence upper bounds is more effective than the traditional approach that uses reweighted evidence lower bounds. While all Dirichlet and Beta methods are able to produce samples on the bounded unit interval, this is the only work that uses this approach for image generation.

5 Experiments

5.1 Implimentation Details

For all experiments we choose a noise to signal schedule based on the denoising diffusion probabilistic models (DDPM) implementation Ho et al. [2020], which is adapted to the continuous domain Song et al. [2021]. By taking the discrete sequence to the limit, we use $\beta(t) = \beta_{\min} + t(\beta_{\max} - \beta_{\min})$, where $t \in [0, 1]$, $\beta_{\min} = 0.1$, $\beta_{\min} = 20$. This results in the perturbation kernel of a logistic normal distribution with the parameters of $\mu = \left[e^{\frac{1}{4}t^2(\beta_{\max} - \beta_{\min}) - \frac{1}{2}t\beta_{\min}}\right] \mathbf{x}_0$ and $v = 1 - e^{-\frac{1}{2}t^2(\beta_{\max} - \beta_{\min}) - t\beta_{\min}}$.

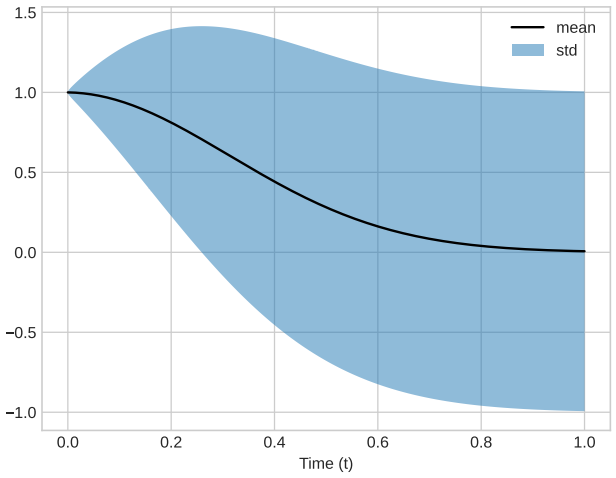


Figure 3: Mean and standard deviation parameters of the logistic normal distributions perturbation kernel when using $\beta(t)$.

No data augmentations were performed for either experiment. We use a UNet Ronneberger et al. [2015] model for the score model, with approximately 140M parameters for the CIFAR-10 experiment and 35M for the MNIST experiment. For both experiments we use $T = 2000$ discrete time steps when computing the log-likelihood and generative samples.

5.2 Binarized MNIST

We first benchmark our dataset on binarized MNIST and compare to other likelihood based methods in Table 1. We compare to DDSM Avdeyev et al. [2023], CR-VAE Li et al. [2023], Locally Masked PixelCNN Jain et al. [2020], PixelRNN van den Oord et al. [2016], EoNADE Uria et al. [2014], MADE Germain et al. [2015] and NADE Uria et al. [2016].

Table 1: Binarized MNIST benchmark performance.

METHOD	NLL (NATS) ↓
LOGISTIC-NORMAL DIFFUSION	77.92
DDSM	78.04
CR-VAE	**76.93**
LOCALLY MASKED PIXELCNN	77.58
PIXELRNN	79.20
PIXELCNN	81.30
EONADE	84.68
MADE	86.43
NADE	88.33

5.3 Unit Interval CIFAR10

In this experiment we treat each pixel of a data point in CIFAR-10 as and RGB value in $[0, 1]$. Considering the case where the logistic-normal distribution is in $\mathbb{S}^2$, we can forget about the implicit variables that is $\mathbf{x}_1 = 1 - \mathbf{x}_0$ and perform one-dimensional diffusion with the logit-normal distribution. For this experiment we re-scale pixel values to be on the interval $[0.1, 0.9]$ for numerical stability.

We compare to wide variety of diffusion implementations and use the Frechet Inception Distance as a measure of image generation quality. The following methods are considered: DDPM Ho et al. [2020], VDM Kingma et al. [2023], Improved DDPM Nichol and Dhariwal [2021], TDPM+ Zheng et al. [2023], VP-EDM Karras et al. [2022], Soft Diffusion Daras et al. [2022], Blurring Diffusion Hoogeboom and Salimans [2022], Cold Diffusion Bansal et al. [2022], Inverse Heat Dispersion Rissanen et al. [2023], D3PM Austin et al. [2021], τ-LDR-10 Campbell et al. [2022], Poisson Diffusion Chen and Zhou [2023], Zhou et al. [2023].

Table 2: Unit Interval CIFAR-10 FID score comparison with models from various different diffusion spaces

Diffusion Space	Model	FID ↓
Gaussian	DDPM	3.17
	VDM	4.00
	Improved DDPM	2.90
	TDPM+	2.83
	VP-EDM	**1.97**
Gaussian + Blurring	Soft Diffusion	3.86
	Blurring Diffusion	3.17
Deterministic	Cold Diffusion	8.92
	Inverse Heat Dispersion	18.96
Categorical	D3PM	7.34
	τLDR-10	3.74
Count	JUMP (Poisson Diffusion)	4.80
Range-Bounded	Beta Diffusion	3.06
	Logistic-Normal Diffusion	3.72

Overall, we find the our Logistic-Normal diffusion model is capable of competitive performance on both discrete and unit interval generative modelling tasks. Given that we do not make sure of advances in diffusion methodology, it is expected that methods like Improved DDPM out-performs the Logistic-Normal diffusion model. We would like to note however, that modification made to Gaussian OU diffusion can easily be integrated into our proposed model. A benefit of our model is faster samping than comparable models such as Beta Diffusion Zhou et al. [2023] or Dirchlet Diffusion Avdeyev et al. [2023] due to our simple OU process. While our method is suitable for the binary and low-class discrete modelling case, it scales linearly with the number of discrete categories and it is unclear how the model will perform in the case of a high dimensional probability simplex.

6 Conclusion

We introduce Logistic-Normal Diffusion, which inherits benefits from the continuous time SDE diffusion framework such as exact likelihood computation and flexible reverse time sampling. The framework makes use of direct correspondence between diffusion in $\mathbb{R}^n$ the probability simplex using Ito's Lemma. Due to the tight relationship, Logistic-Normal Diffusion is able to make use of recent improvements Karras et al. [2022] Nichol and Dhariwal [2021] to the standard diffusion methodology. We find the initial results of the model promising and believe that it can be extended to cases outside computer vision, such as reinforcement learning with discrete environments Janner et al. [2022].

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Pavel Avdeyev, Chenlai Shi, Yuhao Tan, Kseniia Dudnyk, and Jian Zhou. Dirichlet diffusion score model for biological sequence generation, 2023.

Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie S. Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise, 2022.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models, 2022.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018.

Tianqi Chen and Mingyuan Zhou. Learning to jump: Thinning and thickening latent counts for generative modeling, 2023.

Giannis Daras, Mauricio Delbracio, Hossein Talebi, Alexandros G. Dimakis, and Peyman Milanfar. Soft diffusion: Score matching for general corruptions, 2022.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H. Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. Continuous diffusion for categorical data, 2022.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, 2020.

Emiel Hoogeboom and Tim Salimans. Blurring diffusion models, 2022.

Ajay Jain, Pieter Abbeel, and Deepak Pathak. Locally masked convolution for autoregressive models, 2020.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis, 2022.

Myeonghun Jeong, Hyeongju Kim, Sung Jun Cheon, Byoung Jin Choi, and Nam Soo Kim. Diff-tts: A denoising diffusion model for text-to-speech, 2021.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models, 2022.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models, 2023.

Hongming Li, Shujian Yu, and Jose Principe. Causal recurrent variational autoencoder for medical time series generation, 2023.

Aaron Lou and Stefano Ermon. Reflected diffusion models, 2023.

Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2016.

Pierre H. Richemond, Sander Dieleman, and Arnaud Doucet. Categorical sdes with simplex diffusion, 2022.

Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation, 2023.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. abs/1505.04597, 2015.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-a-video: Text-to-video generation without text-video data, 2022.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator, 2014.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation, 2016.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks, 2016.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications, 2023.

Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Truncated diffusion probabilistic models and diffusion-based adversarial auto-encoders, 2023.

Mingyuan Zhou, Tianqi Chen, Zhendong Wang, and Huangjie Zheng. Beta diffusion, 2023.

A Pushing Gaussian Diffusion on the Probability Simplex

A.1 Deriving the SDE with Ito's Lemma

We would like our SDE to be in the following form to be able to use the score matching method:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\,\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\,\mathrm{d}\mathbf{w}.$$

To derive this form, we will use Ito's Lemma to find the SDE for $\mathbf{x}$:

$$\mathrm{d}\mathbf{x}_i = \sigma_i(\mathbf{z}) = \left\{-\frac{1}{2}\beta(t)(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))^\top \mathbf{z} + \frac{1}{2}\beta(t)\,\mathrm{Tr}\left[H_z\sigma_i(\mathbf{z})\right]\right\}\,\mathrm{d}t + \sqrt{\beta(t)}(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))^\top\,\mathrm{d}\mathbf{w}$$

where H_z is the Hessian of $\sigma(\mathbf{z})_i$ with respect to $\mathbf{z}$. To simplify this equation, we will first work with the gradient of the i-th component of $\sigma(\mathbf{z})$ and then the Hessian. The time dependence is dropped during derivations for visual clarity.

A.1.1 Gradient of $\sigma_i(\mathbf{z})$ leading to Diffusion Matrix Term

$$\begin{aligned}
\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}) &= \nabla_{\mathbf{z}}\frac{e^{\mathbf{z}_i}}{1+\sum_{k=1}^{d-1}e^{\mathbf{z}_k}} \\
(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))_j &= \frac{\partial}{\partial \mathbf{z}_j}\frac{e^{\mathbf{z}_i}}{1+\sum_{k=1}^{d-1}e^{\mathbf{z}_k}}
\end{aligned}$$

There are two different cases to consider, when $j = i$ and when $j \neq i$. We consider the first of these below. For convenience, we will use $\alpha(\mathbf{z}) = 1 + \sum_{k=1}^{d-1} e^{\mathbf{z}_k}$.

$$\begin{aligned}
(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))_i &= \frac{\partial}{\partial \mathbf{z}_i}\frac{e^{\mathbf{z}_i}}{\alpha(\mathbf{z})} \\
&= \alpha(\mathbf{z})^{-2}\left[\alpha(\mathbf{z})\frac{\partial}{\partial \mathbf{z}_i}e^{\mathbf{z}_i} - e^{\mathbf{z}_i}\frac{\partial}{\partial \mathbf{z}_i}\alpha(\mathbf{z})\right] \\
&= \alpha(\mathbf{z})^{-2}\left[e^{\mathbf{z}_i}\alpha(\mathbf{z}) - e^{2\mathbf{z}_i}\right] \\
&= e^{\mathbf{z}_i}\alpha(\mathbf{z})^{-1}\left[\alpha(\mathbf{z}) - e^{\mathbf{z}_i}\right] \\
&= \sigma_i(\mathbf{z})\left[1 - \sigma_i(\mathbf{z})\right] \\
&= \mathbf{x}_i(1 - \mathbf{x}_i)
\end{aligned}$$

Next, the case when $j \neq i$:

$$\begin{aligned}
(\nabla_{\mathbf{z}}\sigma_i(\mathbf{z}))_j &= \frac{\partial}{\partial \mathbf{z}_j}\frac{e^{\mathbf{z}_i}}{\alpha(\mathbf{z})} \\
&= -\alpha(\mathbf{z})^{-2}\frac{\partial}{\partial \mathbf{z}_j}\alpha(\mathbf{z})e^{\mathbf{z}_i} \\
&= -\alpha(\mathbf{z})^{-2}e^{\mathbf{z}_i}e^{\mathbf{z}_j} \\
&= -\sigma_i(\mathbf{z})\sigma_j(\mathbf{z}) \\
&= -\mathbf{x}_i\mathbf{x}_j
\end{aligned}$$

At this point, we can notice that $G_t(\mathbf{x})_i = \sqrt{\beta(t)}\nabla_{\mathbf{z}}\sigma_i(\mathbf{z})^\top$ and can write out the full diffusion matrix $G_t(\mathbf{x})$ as:

$$\begin{aligned}
G(\mathbf{x}) &= \sqrt{\beta}J_{\mathbf{z}}\sigma(\mathbf{z}) \\
(J_{\mathbf{z}}\sigma(\mathbf{z}))_{i,j} &= \begin{cases}\mathbf{x}_i(1-\mathbf{x}_i) & \text{if } \ i = j \\ -\mathbf{x}_i\mathbf{x}_j & \text{if } \ i \neq j\end{cases}
\end{aligned}$$

A.1.2 Hessian of $\sigma_i(\mathbf{z})$ leading to Drift Term

Next we deal with the trace Hessian term:

$$\mathrm{Tr}[H_X \sigma_i(\mathbf{z})] = \sum_{j=1}^{d-1} \frac{\partial^2}{\partial \mathbf{z}_j^2} \sigma_i(\mathbf{X})_j$$

which again can be split into two cases. First we deal with the case when $j = i$

$$\begin{aligned}
\frac{\partial^2}{\partial \mathbf{z}_i^2} \sigma_i(\mathbf{z}) &= \frac{\partial}{\partial \mathbf{z}_i} \sigma_i(\mathbf{z})(1 - \sigma_i(\mathbf{z})) \\
&= \sigma_i(\mathbf{z})(1 - \sigma_i(\mathbf{z}))(1 - 2\sigma_i(\mathbf{z})) \\
&= \mathbf{x}_i(1 - \mathbf{x}_i)(1 - 2\mathbf{x}_i)
\end{aligned}$$

Then the case where $j \neq i$

$$\begin{aligned}
\frac{\partial^2}{\partial \mathbf{z}_j^2} \sigma_i(\mathbf{z}) &= -\frac{\partial}{\partial \mathbf{z}_j} \sigma_i(\mathbf{z}) \sigma_j(\mathbf{z}) \\
&= -\sigma_i(\mathbf{z}) \sigma_j(\mathbf{z})(1 - 2\sigma_j(\mathbf{z})) \\
&= -\mathbf{x}_i \mathbf{x}_j (1 - 2\mathbf{x}_j)
\end{aligned}$$

We can now combine these terms together and simplify:

$$\begin{aligned}
\mathrm{Tr}[H_X \sigma_i(\mathbf{z})] &= \mathbf{x}_i(1 - \mathbf{x}_i)(1 - 2\mathbf{x}_i) + \sum_{j \neq i} -\mathbf{x}_i \mathbf{x}_j (1 - 2\mathbf{x}_j) \\
&= \mathbf{x}_i(1 - \mathbf{x}_i)(1 - 2\mathbf{x}_i) - \mathbf{x}_i \left[-\mathbf{x}_i(1 - 2\mathbf{x}_i) + \sum_j \mathbf{x}_j (1 - 2\mathbf{x}_j) \right] \\
\mathrm{Tr}[H_X \sigma(\mathbf{z})] &= \mathbf{x}(\mathbf{1} - \mathbf{x})(\mathbf{1} - 2\mathbf{x}) + \mathbf{x}^2(\mathbf{1} - 2\mathbf{x}) - \|\mathbf{x}(\mathbf{1} - 2\mathbf{x})\|_1 \, \mathbf{x} \\
&= \mathbf{x}(\mathbf{1} - 2\mathbf{x}) - \|\mathbf{x}(\mathbf{1} - 2\mathbf{x})\|_1 \, \mathbf{x}
\end{aligned}$$

Now we could like to simplify terms to get the drift term $\mathbf{f}_t(\mathbf{x})$. We must work with the following:

$$\begin{aligned}
\mathbf{f}(\mathbf{x}) &= -\frac{1}{2}\beta(t)(\nabla_{\mathbf{z}} \sigma_i(\mathbf{z}))^\top \mathbf{z} + \frac{1}{2}\beta(t) \, \mathrm{Tr}\left[H_z \sigma_i(\mathbf{z})\right] \\
&= \frac{1}{2}\beta(t) \left[\mathbf{x} - 2\mathbf{x}^2 + \|\mathbf{x}(1 - 2\mathbf{x})\|_1 \, \mathbf{x} - \frac{1}{\sqrt{\beta(t)}} \mathbf{G}(\mathbf{x}) \sigma(\mathbf{x})^{-1} \right]
\end{aligned}$$

In summary, the forwards process for the diffusion on the simplex, where we re-introduce the time dependence is:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t) \, \mathrm{d}t + \mathbf{G}(\mathbf{x}, t) \, \mathrm{d}\mathbf{w}$$

$$\mathbf{G}(\mathbf{x}, t) = \sqrt{\beta(t)} J_{\mathbf{z}} \sigma(\mathbf{z})$$

$$(J_{\mathbf{z}} \sigma(\mathbf{z}))_{i,j} = \begin{cases} \mathbf{x}_i(1 - \mathbf{x}_i) & \text{if} \quad i = j \\ -\mathbf{x}_i \mathbf{x}_j & \text{if} \quad i \neq j \end{cases} \mathbf{f}(\mathbf{x}, t)$$

$$= \frac{1}{2}\beta(t)\left[\mathbf{x} - 2\mathbf{x}^2 + \|\mathbf{x}(1-2\mathbf{x})\|_1\,\mathbf{x} - \frac{1}{\sqrt{\beta(t)}}\mathbf{G}(\mathbf{x},t)\sigma(\mathbf{x})^{-1}\right]$$

A.2 Reverse Process

The reverse diffusion process is:

$$\mathrm{d}\mathbf{x} = \left\{\mathbf{f}_t(\mathbf{x}) - \nabla_{\mathbf{x}}\cdot[\mathbf{G}_t(\mathbf{x})\mathbf{G}_t(\mathbf{x})^\top] - \mathbf{G}_t(\mathbf{x})\mathbf{G}_t(\mathbf{x})^\top\nabla_{\mathbf{x}}\log p_t(\mathbf{x})\right\}\mathrm{d}t\ + \mathbf{G}_t(\mathbf{x})\,\mathrm{d}\mathbf{w}$$

where (insert definition of matrix divergence) we are able to use the fact that $\mathbf{G}_t(\mathbf{x})^\top = \mathbf{G}_t(\mathbf{x})$. We will also drop the dependence on t for visual clarity.

A.2.1 Diffusion Matrix Divergence Term

First, we will simplify the divergence term. To do this, we will begin with the following:

$$\nabla_{\mathbf{x}}\cdot[\mathbf{G}(\mathbf{x},t)\mathbf{G}(\mathbf{x},t)^\top]_i = \sum_j \frac{\partial}{\partial \mathbf{x}_j}[\mathbf{G}(\mathbf{x},t)\mathbf{G}(\mathbf{x},t)^\top]_{i,j}$$

where we will again split the summation into two cases, when $i = j$ and when $i \neq j$. We begin with the case when $i = j$, where the diffusion matrix is first expanded:

$$\begin{aligned}
\mathbf{G}(\mathbf{x})^2_{i,i} &= \sum_k \mathbf{G}(\mathbf{x})_{i,k}\mathbf{G}(\mathbf{x})_{k,i} \\
&= \mathbf{G}^2_{i,i} + \sum_{k\neq i}\mathbf{G}_{i,k}\mathbf{G}_{k,i} \\
&= \beta\mathbf{x}_i^2(1-\mathbf{x}_i)^2 + \beta(t)\mathbf{x}_i^2\sum_{k\neq i}\mathbf{x}_k^2 \\
&= \beta\mathbf{x}_i^2\left[(1-\mathbf{x}_i)^2 + \sum_{k\neq i}\mathbf{x}_k^2\right]
\end{aligned}$$

and then the derivative computed:

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{x}_i}\mathbf{G}(\mathbf{x})^2_{i,i} &= \beta\frac{\partial}{\partial \mathbf{x}_i}\mathbf{x}_i^2\left[(1-\mathbf{x}_i)^2 + \sum_{k\neq i}\mathbf{x}_k^2\right] \\
&= 2\beta\mathbf{x}_i\left[(1-\mathbf{x}_i)^2 + \sum_{k\neq i}\mathbf{x}_k^2\right] - 2\beta\mathbf{x}_i^2(1-\mathbf{x}_i) \\
&= 2\beta\mathbf{x}_i\left[(1-\mathbf{x}_i)(1-2\mathbf{x}_i) + \sum_{k\neq i}\mathbf{x}_k^2\right]
\end{aligned}$$

Next, the case when $i \neq j$. Again, we begin by expanding the diffusion matrix:

$$\begin{aligned}\mathbf{G}(\mathbf{x})^2_{i,j} &= \mathbf{G}_{i,i}\mathbf{G}_{i,j} + \mathbf{G}_{i,j}\mathbf{G}_{j,j} + \sum_{k\neq i,j}\mathbf{G}_{i,k}\mathbf{G}_{k,j} \\ &= \beta\left[-\mathbf{x}_i^2\mathbf{x}_j(1-\mathbf{x}_i) - \mathbf{x}_j^2\mathbf{x}_i(1-\mathbf{x}_j) + \mathbf{x}_i\mathbf{x}_j\sum_{k\neq i,j}\mathbf{x}_k^2\right] \\ &= -\beta\mathbf{x}_i\mathbf{x}_j\left[\mathbf{x}_i(1-\mathbf{x}_i) + \mathbf{x}_j(1-\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right]\end{aligned}$$

and then the sum of derivative terms:

$$\sum_{j\neq i}\frac{\partial}{\partial\mathbf{x}_j}\mathbf{G}(\mathbf{x})^2_{i,j} = \beta\sum_{j\neq i}\frac{\partial}{\partial\mathbf{x}_j} - \mathbf{x}_i\mathbf{x}_j\left[\underbrace{\mathbf{x}_i(1-\mathbf{x}_i) + \mathbf{x}_j(1-\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2}_{\mathbf{a}(\mathbf{x})}\right]$$

we can approach this by using the product rule. First we will compute the derivative of $\mathbf{a}(\mathbf{x})$:

$$\begin{aligned}\frac{\partial}{\partial\mathbf{x}_j}\mathbf{a}(\mathbf{x}) &= \frac{\partial}{\partial\mathbf{x}_j}\left[\mathbf{x}_i(1-\mathbf{x}_i) + \mathbf{x}_j(1-\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right] \\ &= (1-2\mathbf{x}_j)\end{aligned}$$

and then the derivative of the product:

$$\begin{aligned}\sum_{j\neq i}\frac{\partial}{\partial\mathbf{x}_j}\mathbf{G}(\mathbf{x})^2_{i,j} &= \beta\sum_{j\neq i}\frac{\partial}{\partial\mathbf{x}_j} - \mathbf{x}_i\mathbf{x}_j\mathbf{a}(\mathbf{x}) \\ &= \beta\sum_{j\neq i}[-\mathbf{x}_i\mathbf{a}(\mathbf{x}) - \mathbf{x}_i\mathbf{x}_j(1-2\mathbf{x}_j)] \\ &= -\beta\mathbf{x}_i\sum_{j\neq i}\left[\mathbf{x}_i(1-\mathbf{x}_i) + \mathbf{x}_j(1-\mathbf{x}_j) + \mathbf{x}_j(1-2\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right] \\ &= -(d-2)\beta\mathbf{x}_i^2(1-\mathbf{x}_i) - \beta\mathbf{x}_i\sum_{j\neq i}\left[\mathbf{x}_j(1-\mathbf{x}_j) + \mathbf{x}_j(1-2\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right] \\ &= -(d-2)\beta\mathbf{x}_i^2(1-\mathbf{x}_i) - \beta\mathbf{x}_i\sum_{j\neq i}\left[\mathbf{x}_j(2-3\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right]\end{aligned}$$

Now we can combine the two cases together to get the full divergence term:

$$\nabla_{\mathbf{x}} \cdot [\mathbf{G}_t(\mathbf{x})\mathbf{G}_t(\mathbf{x})^\top]_i = \sum_j \frac{\partial}{\partial \mathbf{x}_j}[\mathbf{G}_t(\mathbf{x})\mathbf{G}_t(\mathbf{x})^\top]_{i,j}$$
$$= \underbrace{2\beta\mathbf{x}_i \left[(1-\mathbf{x}_i)(1-2\mathbf{x}_i) + \sum_{k\neq i}\mathbf{x}_k^2\right]}_{\mathbf{b}_1(\mathbf{x})_i} - \underbrace{(d-2)\beta\mathbf{x}_i^2(1-\mathbf{x}_i)}_{\mathbf{b}_2(\mathbf{x})_i}$$
$$\underbrace{-\beta\mathbf{x}_i \sum_{j\neq i}\left[\mathbf{x}_j(2-3\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right]}_{\mathbf{b}_3(\mathbf{x})_i}$$

To complete this section, we would like to vectorize the equation that we have just derived. We will work on each part of the equation separately. First, we will consider the term $\mathbf{b}_1(\mathbf{x})_i$:

$$\mathbf{b}_1(\mathbf{x})_i = 2\beta\mathbf{x}_i\left[(1-\mathbf{x}_i)(1-2\mathbf{x}_i) + \sum_{k\neq i}\mathbf{x}_k^2\right]$$
$$= 2\beta\mathbf{x}_i\left[1-3\mathbf{x}_i+2\mathbf{x}_i^2-\mathbf{x}_i^2+\sum_k \mathbf{x}_k^2\right]$$
$$\mathbf{b}_1(\mathbf{x}) = 2\beta\mathbf{x}[\mathbf{x}^2 - 3\mathbf{x} + (1+\|\mathbf{x}\|_2^2)\mathbf{1}]$$

The next term can be vectorized as:

$$\mathbf{b}_2(\mathbf{x})_i = (d-2)\beta\mathbf{x}_i^2(1-\mathbf{x}_i)$$
$$\mathbf{b}_2(\mathbf{x}) = (d-2)\beta\mathbf{x}^2(\mathbf{1}-\mathbf{x})$$

and the final term as:

$$\mathbf{b}_3(\mathbf{x})_i = \beta\mathbf{x}_i\sum_{j\neq i}\left[\mathbf{x}_j(2-3\mathbf{x}_j) - \sum_{k\neq i,j}\mathbf{x}_k^2\right]$$
$$= \beta\mathbf{x}_i\sum_{j\neq i}[\mathbf{x}_j(2-3\mathbf{x}_j) + \mathbf{x}_i^2 + \mathbf{x}_j^2 - \|\mathbf{x}\|_2^2]$$
$$= (d-2)\beta(\mathbf{x}_i^2 - \|\mathbf{x}\|_2^2)\mathbf{x}_i + \underbrace{\beta\mathbf{x}_i\sum_{j\neq i}[\mathbf{x}_j(2-3\mathbf{x}_j)+\mathbf{x}_j^2]}_{\mathbf{b}_3'(\mathbf{x})_i}$$

where the first term can be easily vectorized. We then continue to work on the second term $\mathbf{b}_3'(\mathbf{x})_i$:

$$\mathbf{b}_3'(\mathbf{x})_i = \beta\mathbf{x}_i\sum_{j\neq i}[2\mathbf{x}_j - 2\mathbf{x}_j^2]$$
$$= \beta\mathbf{x}_i\left[2\mathbf{x}_i^2 - 2\mathbf{x}_i + 2\|\mathbf{x}\|_1 - 2\|\mathbf{x}\|_2^2\right]$$
$$\mathbf{b}_3'(\mathbf{x}) = 2\mathbf{x}^2(\mathbf{x}-1) + 2\mathbf{x}(\|\mathbf{x}\|_1 - \|\mathbf{x}\|_2^2)$$

and write both terms together as:

$$\mathbf{b}_3(\mathbf{x}) = (d-2)\beta(\mathbf{x}^2 - \|\mathbf{x}\|_2^2)\mathbf{x} + 2\mathbf{x}^2(\mathbf{x}-1) + 2\mathbf{x}(\|\mathbf{x}\|_1 - \|\mathbf{x}\|_2^2)$$

Finally, we can combine these terms together to get the full divergence term:

$$\begin{aligned}\nabla_{\mathbf{x}} \cdot [\mathbf{G}_t(\mathbf{x})\mathbf{G}_t(\mathbf{x})^\top] &= \mathbf{b}_1(\mathbf{x}) - \mathbf{b}_2(\mathbf{x}) - \mathbf{b}_3(\mathbf{x}) \\ &= \beta\mathbf{x}\left[-(d-2)\mathbf{x} + (d+2)\|\mathbf{x}\|_2 - 2\|\mathbf{x}\|_1 + 2\right]\end{aligned}$$

A.2.2 Score Derivation

The final term that we need to derive is the score term. We will begin by writing out the full equation for the score:

$$\log p(\mathbf{x}) = -\log[Z] - \underbrace{\log\left[(1 - \|\mathbf{x}\|_1)\prod_{i-1}^{d-1}\mathbf{x}_i\right]}_{s_a(\mathbf{x})} - \underbrace{\frac{1}{2v}\left\|\log\left[\frac{\mathbf{x}}{1-\|\mathbf{x}\|_1}\right] - \mu\right\|_2^2}_{s_b(\mathbf{x})}$$

We deal with the gradients, starting with the second term (the first one has no gradient).

$$\begin{aligned}g &:= -\nabla_x \log\left[\prod_{i=1}^{d} x_i\right] \\ g_i &= -\frac{\partial}{\partial x_i}\left(\sum_{i=1}^{d-1}\log\left[x_i\right] + \log\left[a - \sum_{k=1}^{d-1} x_k\right]\right) \\ &= -\frac{1}{x_i} + \frac{1}{a - \sum_{k=1}^{d-1} x_k} \\ &= \frac{1}{x_d} - \frac{1}{x_i} \\ &= \frac{x_i - x_d}{x_i x_d}\end{aligned}$$

Next, we deal with the exponential term:

$$\begin{aligned}h &:= -\frac{1}{2v}\nabla_x\left\|\log\left[\frac{\bar{x}_d}{x_d}\right] - \mu\right\|_2^2 \\ h_i &= -\frac{1}{2v}\frac{\partial}{\partial x_i}\left(\sum_{k=1}^{d-1}\left(\log\left[\frac{x_k}{x_d}\right] - \mu\right)^2\right) \\ &= -\frac{1}{2v}\sum_{k=1}^{d-1}\left(\frac{\partial}{\partial u}u^2\frac{\partial}{\partial x_i}u\right), u = \log\left[\frac{x_k}{x_d}\right] - \mu\end{aligned}$$

We can just focus on $\beta := \frac{\partial}{\partial u}u^2\frac{\partial}{\partial x_i}u$ for now

$$\begin{aligned}\beta &:= \frac{\partial}{\partial u}u^2\frac{\partial}{\partial x_i}u \\ &= 2u\left(\frac{\partial}{\partial x_i}\log\left[x_k\right] - \frac{\partial}{\partial x_i}\log\left[a - \sum_{k=1}^{d-1} x_k\right]\right) \\ &= 2u\left(\delta_{ik}\frac{1}{x_i} + \frac{1}{x_d}\right)\end{aligned}$$

Combining terms we get:

$$
\begin{aligned}
h_i &= -\frac{1}{v}\sum_{k=1}^{d-1}\left(\delta_{ik}\frac{1}{x_i}+\frac{1}{x_d}\right)\left(\log\left[\frac{\bar{x}_d}{x_d}\right]-\mu\right) \\
&= -\frac{1}{vx_d}\sum_{k=1}^{d-1}\left(\log\left[\frac{x_k}{x_d}\right]-\mu\right)-\frac{1}{vx_i}\left(\log\left[\frac{x_i}{x_d}\right]-\mu\right) \\
&= -\frac{1}{vx_d}\sum_{k=1}^{d-1}\gamma_\mu^k(\mathbf{x})-\frac{1}{vx_i}\gamma_\mu^i(\mathbf{x})
\end{aligned}
$$

where we write $\gamma_\mu^i(\mathbf{x}) = \log\left[\frac{x_i}{x_d}\right]-\mu$

For the final results, we must combine the h and g terms together to get:

$$
\nabla_x \log p_a(x)_i = -\frac{1}{vx_d}\sum_{k=1}^{d-1}\gamma_\mu^k(\mathbf{x})-\frac{2}{vx_i}\gamma_\mu^i(\mathbf{x})+\frac{x_i-x_d}{x_ix_d}
$$

which can be vectorized as:

$$
\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \frac{\mathbf{x}-||\mathbf{x}||_1\mathbf{1}}{||\mathbf{x}||_1\mathbf{x}}-\frac{1}{v||\mathbf{x}||_1}\gamma(\mathbf{x})-\frac{\mathbf{1}^\top[\gamma(\mathbf{x})-\mu]}{v||\mathbf{x}||_1}\mathbf{1}
$$