Understanding SDE-GAN

Jingyue Lu
January 9, 2022

1 Introduction

In this short report, we study a recently proposed method of using Wasserstein Gan (WGAN) to fit stochastic differential equations (SDE). The key underlying ideas of the method are introduced in [Kidger et al., 2021b] while implementation details with improved efficiency are provided in [Kidger et al., 2021a]. We refer to this method as SDE GAN from here onwards. In the following, we will mainly focus on providing explanations for the uncleared parts and details for the derivations that are omitted in the papers in the hope to paint a more comprehensive picture of SDE-GAN. In addition, we will conduct a study of applying SDE-GAN to real-world stock index data to evaluate its performance.

2 SDE-GAN: Model

SDE-GAN is a generative model specifically developed for SDEs. Like all other GAN-based models, it consists of a generator and a discriminator. The generator takes the form of Neural SDEs to model a path-valued random variable $Y_{true} : [0, T] \to \mathbb{R}^y$ while the discriminator adopts the formulation of a neural controlled differential equations (CDE) to distinguish whether a sample is real or generated. The generator and the discriminator are trained together in a zero-sum game until the generator has learnt to fool the discriminator. At this time, the generator is able to generate samples Y that have approximately the same distribution as Y_{true}. In terms of the training objective, SDE-GAN adopts the idea of WGAN by using Wasserstein distance instead of the original Jensen-Shannon divergence to measure the difference between the true data distribution and the generated data distribution. We give more details on WGAN, the generator, and the discriminator below.

2.1 WGAN

Given an optimal discriminator, vanilla GAN models encourage the learning of the generator by penalizing the Jensen-Shannon (JS) divergence between the real data distribution and the generated data distribution[1]. However, as demonstrated in [Arjovsky et al., 2017], JS divergence fails to provide a meaningful value when these two distributions are disjoint, which is likely to happen when both distributions are embedded in low-dimensional manifolds. To deal with this issue, [Arjovsky et al., 2017] proposed to replace JS divergence with Wasserstein distance.

Let p_r be the real data distribution and p_g the generated data distribution, Wasserstein distance is defined as

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]. \tag{1}$$

Here, $\Pi(p_r, p_q)$ denotes the set of all possible joint distributions between p_r and p_q. It can be shown that Wasserstein distance manages to provide a smooth measure even when p_r and p_q are disjoint.

Solving for Eq. 1 directly is intractable as it is impossible to enumerate all joint distributions of p_r and p_g. However, with a delicate application of Kantorovich-Rubinstein duality arguments, we can solve the following maximization problem instead:

$$W(p_r, p_g) = \frac{1}{k} \sup_{\|F\|_L \leq K} \mathbb{E}_{x\sim p_r}[F(x)] - \mathbb{E}_{x\sim p_g}[F(x)]. \tag{2}$$

In the context of WGAN, we can think the function f as the discriminator and $x \sim p_g$ is produced by the generator g such that $x = g(z)$ for $z \sim p_z$. The only extra requirement is that the discriminator F has to be

[1] More details on this claim can be found in the Appendix.

K-Lipschitz continuous. The final WGAN loss is

$$\min_{\theta\in\Theta}\max_{\phi\in\Phi}\mathbb{E}_{x\sim p_r}[F_\phi(x)] - \mathbb{E}_{z\sim p_z}[F_\phi(G_\theta(z))], \tag{3}$$

where p_z is the noise distribution.

In SDE-GAN, the discriminator has a recurrent nature. Depending on the number of steps T taken and the Lipschitz constant λ of the discriminator at each step, the overall discriminator has a Lipschitz constant $O(\lambda^T)$. For it to be bounded regardless the value of T, hard constraint $\lambda \leq 1$ has to be enforced. To fulfill this requirement, the authors of [Kidger et al., 2021a] have suggested weight clipping and LipSwish activation functions to replace the conventional gradient penalty.

We now give the exact forms for the generator G_θ and the discriminator F_ϕ

2.2 Generator

For the completeness of our discussion, we mention the minimum structure for constructing the generator as Neural SDEs. Recall that we want to model a path-valued random variable $Y_{true} : [0, T] \to \mathbb{R}^y$. The generator is modelled as follows:

$$X_0 = \xi_\theta(V) \tag{4}$$
$$dX_t = \mu_\theta(t, X_t)\, dt + \sigma_\theta(t, X_t) \circ dW_t \tag{5}$$
$$Y_t = \alpha_\theta X_t + \beta_\theta. \tag{6}$$

Here, V is a noise input drawn from $\mathcal{N}(0, I_v)$ and $W : [0, T] \to \mathbb{R}^w$ is a Brownian motion. The functions $\xi_\theta : \mathbb{R}^v \to \mathbb{R}^x$, $\mu_\theta : [0, T] \times \mathbb{R}^x \to \mathbb{R}^x$ and $\sigma_\theta : [0, T] \times \mathbb{R}^x \to \mathbb{R}^{x\times w}$ are neural networks. The strong solution $X : [0, T] \to \mathbb{R}^x$ to the SDE exists under mild conditions. Finally, we have $\alpha_\theta \in \mathbb{R}^{y\times x}$ and $\beta_\theta \in \mathbb{R}^y$. The modelled Y should have approximately the same distribution as Y_{true} when the generator is trained to its optimal.

2.3 Discriminator

Unlike other GAN discriminators that take point-wise values as inputs, the discriminator of SDE-GAN has to operate on variables Y that have values over a path. To satisfy this requirement, the authors of [Kidger et al., 2021b] have employed Neural CDEs, proposed in [Kidger et al., 2020]. CDEs are typically used in modeling functions of time series and has the form:

$$dz(t) = h_\theta(t, z(t))dX(t), \tag{7}$$

where $z(t)$ is a continuous path. We note that when $dX(t)$ on the right hand side is replaced by dt, we have the standard ordinary differential equation. As mentioned in [Kidger et al., 2020], the benefit of using $dX(t)$ is that the model has a direct mechanism for incorporating incoming data: we can treat X as an interpolated function of data points $(x_0, x_1, \dots)$ to represent how data changes over time. Since the differential equation Eq. 7 now change in response to the input X, we say the differential question is controlled or driven by X. When the function $h_\theta(t, z(t))$ is a neural network, we call the model Neural CDE.

Following the same idea, SDE-GAN models its discriminator as a neural CDE driven by the path-valued variable Y. The minimum structure for the discriminator is:

$$H_0 = \xi_\phi(Y_0), \tag{8}$$
$$dH_t = f_\phi(t, H_t)\, dt + g_\phi(t, H_t) \circ dY_t, \tag{9}$$
$$D = m_\phi \cdot H_T, \tag{10}$$

where $\xi_\phi : \mathbb{R}^y \to \mathbb{R}^h$, $f_\phi : [0, T] \times \mathbb{R}^h \to \mathbb{R}^h$ and $g_\phi : [0, T] \times \mathbb{R}^h \to \mathbb{R}^{h\times y}$ are neural networks and $m_\phi \in \mathbb{R}^h$. Again, the strong solution $H : [0, T] \to \mathbb{R}^x$ to the SDE exists under mild conditions. Finally, we mention that, as pointed out in Section 2.1, the discriminator has to have a maximum Lipschitz constant one.

So far, we have introduced the model structure for SDE-GAN. In the next section, we will study how to carry out the learning procedure for it.

3 SDE-GAN: Learning

Given the model structure, the forward learning for SDE-GAN is straightforward. However, in terms of the backward pass, we note that both the generator and the discriminator have a recurrent nature (they have to solve SDEs numerically). This results in high memory cost for standard back-propagation, in which case, the memory cost grows linearly with the number of steps t taken. To deal with this issue, the authors of [Kidger et al., 2021a] have used the continuous adjoint method combined with their newly developed SDE solver: the reversible Heun method. We discuss this modified back-propagation below.

3.1 Adjoint Method

Using the adjoint method for back-propagation is first proposed by Chen et al. [2018] in solving neural ordinary differential equations (ODE). Unlike standard back-propagation, the adjoint method treats the original ODE as a black-box and computes gradients by solving another ODE backward in time. Since there is no need for the adjoint method to store any intermediate quantities of forward pass for gradient computation, the adjoint method has a constant memory cost. Li et al. [2020] later generalised this idea for computing gradients of SDEs.

Mathematically, consider some Stratonovich SDE

$$dZ_t = \mu(t, Z_t)\, dt + \sigma(t, Z_t) \circ dW_t \qquad \text{for } t \in [0, T],$$

and a loss $L : \mathbb{R} \to \mathbb{R}$ on the terminal value Z_T. The adjoint process is defined as $A_t := \frac{dL(Z_T)}{dZ_t} \in \mathbb{R}^z$, which is a solution to the SDE

$$dA_t^i = -A_t^j \frac{\partial \mu^j}{\partial Z^i}(t, Z_t)dt - A_t^j \frac{\partial \sigma^{j,k}}{\partial Z^i}(t, Z_t) \circ dW_t^k. \tag{11}$$

Provided with the initial condition $A_T = \frac{dL(Z_T)}{dZ_T}$, we solve the SDE Eq. 11 numerically to get the desired gradient $A_0 = \frac{dL(Z_T)}{Z_0}$.

3.2 The Reversible Heun Method

Kidger et al. [2021a] introduced Reversible Heun method to solve the SDE Eq. 11. The key advantage of the method is that it is algebraically reversible. Normally, Z_t computed along the way of solving the SDE Eq. 11 are different from their values in the forward pass. Reversible Heun method allows the values of Z_t in forward and backward passes to match exactly, leading to precise gradient computation using the adjoint method.

Algorithms of Reversible Heun method can be found in [Kidger et al., 2021a]. Below, we give further details on gradients computation carried out in Algorithm 2 in [Kidger et al., 2021a].

Assume $\frac{\partial L(Z_T)}{\partial z_T}$ is provided. We start by calculating $\frac{\partial L(Z_T)}{\partial \hat{z}_T}$ as follows:

$$\frac{\partial L(Z_T)}{\partial \mu_T} = \frac{\partial L(Z_T)}{\partial z_T} \cdot \frac{1}{2}\Delta t, \tag{12}$$

$$\frac{\partial L(Z_T)}{\partial \sigma_T} = \frac{\partial L(Z_T)}{\partial z_T} (\frac{1}{2}\Delta W), \tag{13}$$

$$\frac{\partial L(Z_T)}{\partial \hat{z}_T} = \frac{\partial L(Z_T)}{\partial \mu_T}\frac{\partial \mu_T}{\partial \hat{z}_T} + \frac{\partial L(Z_T)}{\partial \sigma_T}\frac{\partial \sigma_T}{\partial \hat{z}_T}. \tag{14}$$

Here, $\frac{\partial \mu_T}{\partial \hat{z}_T}$ and $\frac{\partial \sigma_T}{\partial \hat{z}_T}$ are computed through standard network propagation. After that, we conduct an iteration procedure for $0 \leq n < T$. At each time step n, we are provided with the inputs:

$$\frac{\partial L(Z_T)}{\partial z_{n+1}}, \quad \frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}.$$

We first make some preliminary computations:

$$\begin{aligned}
\frac{\partial L(Z_T)}{\partial z_{n+1}}\frac{\partial z_{n+1}}{\partial \mu_n} &= \frac{\partial L(Z_T)}{\partial z_{n+1}}\cdot\frac{1}{2}\Delta t,\\
\frac{\partial L(Z_T)}{\partial z_{n+1}}\frac{\partial z_{n+1}}{\partial \sigma_n} &= \frac{\partial L(Z_T)}{\partial z_{n+1}}(\frac{1}{2}\Delta W),\\
\frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}\frac{\partial \hat{z}_{n+1}}{\partial \mu_n} &= \frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}\cdot\Delta t,\\
\frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}\frac{\partial \hat{z}_{n+1}}{\partial \sigma_n} &= \frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}(\Delta W).
\end{aligned}$$

Then, we make the following calculations:

$$\frac{\partial L(Z_T)}{\partial z_n} = \frac{\partial L(Z_T)}{\partial z_{n+1}} + 2\cdot\frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}, \tag{15}$$

$$\frac{\partial L(Z_T)}{\partial z_n}\frac{\partial z_n}{\partial \mu_n} = \frac{\partial L(Z_T)}{\partial z_n}\cdot\frac{1}{2}\Delta t, \tag{16}$$

$$\frac{\partial L(Z_T)}{\partial \mu_n} = \frac{\partial L(Z_T)}{\partial z_{n+1}}\frac{\partial z_{n+1}}{\partial \mu_n} + \frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}\frac{\partial \hat{z}_{n+1}}{\partial \mu_n} + \frac{\partial L(Z_T)}{\partial z_n}\frac{\partial z_n}{\partial \mu_n}, \tag{17}$$

$$\frac{\partial L(Z_T)}{\partial z_n}\frac{\partial z_n}{\partial \sigma_n} = \frac{\partial L(Z_T)}{\partial \hat{z}_n}(\frac{1}{2}\Delta W), \tag{18}$$

$$\frac{\partial L(Z_T)}{\partial \sigma_n} = \frac{\partial L(Z_T)}{\partial z_{n+1}}\frac{\partial z_{n+1}}{\partial \sigma_n} + \frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}}\frac{\partial \hat{z}_{n+1}}{\partial \sigma_n} + \frac{\partial L(Z_T)}{\partial z_n}\frac{\partial z_n}{\partial \sigma_n}, \tag{19}$$

$$\frac{\partial L(Z_T)}{\partial \hat{z}_n} = -\frac{\partial L(Z_T)}{\partial \hat{z}_{n+1}} + \frac{\partial L(Z_T)}{\partial \mu_n}\frac{\partial \mu_n}{\partial \hat{z}_n} + \frac{\partial L(Z_T)}{\partial \sigma_n}\frac{\partial \sigma_n}{\partial \hat{z}_n}. \tag{20}$$

Similarly, $\frac{\partial \mu_n}{\partial \hat{z}_n}$ and $\frac{\partial \sigma_n}{\partial \hat{z}_n}$ are evaluated through standard network back-propagation. We treat $\frac{\partial L(Z_T)}{\partial z_n}$, $\frac{\partial L(Z_T)}{\partial \hat{z}_n}$ as the input values for the next step.

In terms of parameters θ for $\mu(\cdot;\theta)$ and $\sigma(\cdot;\theta)$, we note that

$$\frac{\partial dL(Z_T)}{\partial \theta} = \sum_{n=0}^{T}\frac{\partial dL(Z_T)}{\partial \mu_n}\frac{\partial \mu_n}{\partial \theta} + \frac{\partial dL(Z_T)}{\partial \sigma_n}\frac{\partial \sigma_n}{\partial \theta}. \tag{21}$$

Thus, we calculate $\frac{\partial \mu_n}{\partial \theta}$, $\frac{\partial \sigma_n}{\partial \theta}$ at the same time as computing $\frac{\partial \mu_n}{\partial \hat{z}_n}$, $\frac{\partial \sigma_n}{\partial \hat{z}_n}$. Then we evaluate

$$\frac{\partial dL(Z_T)}{\partial \mu_n}\frac{\partial \mu_n}{\partial \theta} + \frac{\partial dL(Z_T)}{\partial \sigma_n}\frac{\partial \sigma_n}{\partial \theta}$$

and add it to the sum of previous summands. Only the updated partial sum is recorded and carried into the next step $n-1$.

4 Experiments

We evaluate SDE-GAN on 5 years of Hang Seng index(HSI) data from 1st Dec, 2016 to 30th Nov, 2021[2]. We will use the adjusted close values. Denote HSI as S and assume it satisfies

$$dS/S = a\,dt + s\,dW, \tag{22}$$

where a, s are constants.

[2]There is one day with null data in the dataset. We checked and found the stock market closed on that day due to weather conditions, so we directly dropped the null data.

4.1 Data Preprocessing

To facilitate the learning process, instead of directly fitting S, we study $\log(S)$. By Itô's Lemma, we have

$$d\log(S) = (a - \frac{s^2}{2})\,dt + s\,dW. \tag{23}$$

Furthermore, as suggested by the authors of [Kidger et al., 2021b], we normalise the data $\log(S)$. It should be noted that the mean and standard deviation used for normalisation should be those computed on the training dataset. Let μ_{train} and σ_{train} be the mean and standard deviation of the training dataset. The final model we try to fit is

$$d\frac{\log(S) - \mu_{train}}{\sigma_{train}} = \frac{1}{\sigma_{train}}(a - \frac{s^2}{2})\,dt + \frac{s}{\sigma_{train}}dW. \tag{24}$$

Since SDE-GAN is developed for Stratonovich SDEs, one last step is to transform the above Itô equation into its corresponding Stratonovich form. In this case, given that a, s are constants, both Itô equation and Stratonovich equation have the same drift and diffusion terms.

4.2 SDE-GAN:Training

We first split all HSI data points into training, validation, and testing datasets. The split ratio we used is 3:1:1. Then, we treat the HSI of a month as one process sample. That is, every 22 (t-size in the running script) consecutive HSI points are treated as one data sample. We form process samples within each dataset.

In terms of the SDE-GAN architecture, we followed the one authors Chen et al. [2018] provided in the example script in the torchsde library. However, since we only have roughly 1000 process samples (8192 samples are used in the example script), we decrease the architecture size by using 3 input noise dimensions and 8 hidden units. Other layer configurations are kept the same as the example script. Model weights are initialised using Kaiming initialiser with scales adjusted so the untrained weights have similar variances to the training data.

4.3 Results

We have tested various combinations of initial learning rates, weight decay rate, stochastic weight averaging starting epoch, and batch size. Detailed plots can be viewed by running the tensorboard on the github directory (https://github.com/JingyueLu/tfsde/tree/main). Here, we give a snapshot of all losses. Based on

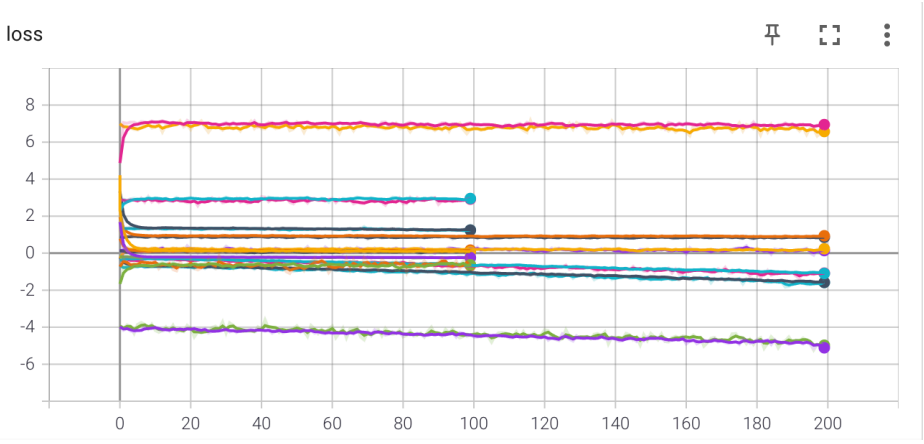

Figure 1: A snapshot of all training/validation losses we experimented. For all experiments, training loss curve stays very close to its corresponding validation loss curve. The purpose here is to show the overall trend, so we removed all labels for visibility.

the Figure 1, we make the following observations. Firstly, all experiments converge within the first epoch,

which could be caused by the fact that the training dataset is small. Results are sensitive to the initial starting value. For the same set of parameters, the model may converge to a number close to 0 or a number far away from zero. This might be the nature of GAN models. Training loss is very close to the validation loss in all experiments. Although not shown in the plot, the final test loss is also close to the final training loss. This indicates the model may not suffer from overfitting nor underfitting. To further improve the model's performance, we need to increase the dataset. Among all combinations of parameters tested, we found the setting: batch-size=128, weight-decay=0.0, generator-learning-rate=2e-4, discrminator-learning-rate=1e-4 gives the best performance. Thus, we use the same set of parameters to train the final model on the combined datasets of the training dataset and the validation dataset. The training loss is shown in Figure 2 and the final test loss is 0.0724.

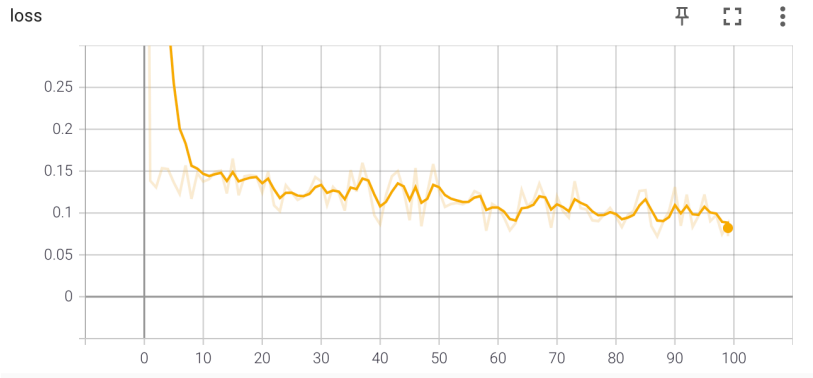

Figure 2: The training loss for the model trained on the new training dataset, which combines the original training dataset and the validation dataset.

5 Discussions

In this report, we have studied and evaluated the method SDE-GAN. Before we end, we mention several limitations of our study. Firstly, the authors of [Kidger et al., 2021a] have developed new ways for speeding up the Brownian motion sampling. Since the explanation in the paper is clear and straightforward, we decide not to discuss these improvements. Secondly, in terms of the experiments, we have only used a simple measure of test loss to evaluate the models' performance. Comprehensive measurements together with larger datasets may help us gain a better understanding of SDE-GAN in the future.

6 Appendix

In this section, we collect some basic definitions and facts that might be useful for readers.

KL(Kullback-Leibler) Divergence

KL divergence is a statistical distance that measures how one probability distribution q diverges is different from a second reference probability distribution p.

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)}\, dx.$$

D_{KL} achieves the minimum zero when $p(x) == q(x)$ everywhere.

Remark. KL divergence is asymmetric. In cases where $p(x)$ is close to zero, but $q(x)$ is significantly non-zero, the q's effect is disregarded.

Jensen-Shannon Divergence

Jenson-Shannon Divergence is a symmetrized and smoothed version of the KL divergence. It is defined as

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2}).$$

Generative Adversarial Network (GAN)

Denote the generator as G and the discriminator as D. Moreover, let p_z denote the data distribution over noise input z; p_g the generator's distribution over x; and p_r the data distribution over real samples x.

The standard training objective for GAN is

$$\begin{aligned}\min_G \max_D L(D,G) &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \\ &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1-D(x))].\end{aligned}$$

It can be computed that for the well-defined loss function

$$L(G,D) = \int_x \left(p_r(x)\log(D(x)) + p_g(x)\log(1-D(x))\right),$$

the optimal value for D is $D^*(x) = \frac{p_r(x)}{p_r(x)+p_g(x)}$. When the generator is trained to optimal, the distribution p_g is very close to p_r, which leads to $D^*(x) = \frac{1}{2}$. As a result, when both G and D are at their optimal, we have

$$\begin{aligned}L(G,D^*) &= \log\frac{1}{2}\int_x p_r(x)\, dx + \log\frac{1}{2}\int_x p_g(x)\, dx \\ &= -2\log 2.\end{aligned}$$

To connect the loss with JS divergence, we note that

$$\begin{aligned}D_{JS}(p_r||p_g) &= \frac{1}{2}D_{KL}(p_r||\frac{p_r+p_g}{2}) + \frac{1}{2}D_{KL}(p_g||\frac{p_r+p_g}{2}) \\ &= \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_r(x)}{p_r(x)+p_g(x)}\, dx\right) + \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_g(x)}{p_r(x)+p_g(x)}\, dx\right) \\ &= \frac{1}{2}\left(\log 4 + L(G,D^*)\right).\end{aligned}$$

It follows,

$$L(G,D^*) = 2D_{JS}(p_r||p_g) - 2\log 2.$$

The loss of GAN quantifies the similarity between p_r and p_q with JS Divergence. The optimal G^* gives zero JS divergence, resulting $L(G^*,D^*) = -2\log 2$, which is consistent with above analyses.

Itô's Process

An Itô process or stochastic integral is a stochastic process on $(\Sigma, \mathcal{F}, \mathbb{P})$ adopted to $\mathcal{F}_t$ which can be written in the form

$$X_t = X_0 + \int_0^t \mu_s \, ds + \int_0^t \sigma_s \, dW_s,$$

where $\mu, \sigma \in \mathcal{L}_2$. Equivalently, we can write it as

$$dX_t = \mu_t \, dt + \sigma_t \, dW.$$

Itô's Lemma

Let X_t be an Itô process $dX_t = \mu_t \, dt + \sigma_t \, dW_t$. Suppose $g(x) \in C^2(\mathbb{R})$ is a twice continuously differentiable function. Suppose $g(X_t) \in \mathcal{L}_2$. Then $Y_t = g(X_t)$ is again an Itô process and

$$dY_t = \frac{\partial g(X_t)}{\partial x} dX_t + \frac{1}{2} \frac{\partial^2 g(X_t)}{\partial x^2} (dX_t)^2.$$

That is,

$$dY_t = \left(\frac{\partial g(X_t)}{\partial x} \mu_t + \frac{1}{2} \frac{\partial^2 g(X_t)}{\partial x^2} (X_t) \sigma_t^2 \right) dt + \frac{\partial g(X_t)}{\partial x} \sigma_t dW_t.$$

Itô integral vs Stratonovich Integral

Suppose X_t solves the Stratonovich equation

$$dX_t = \mu_t \, dt + \sigma_t \circ dW_t.$$

Then X_t solves the Itô equation

$$dX_t = \left(\mu_t + \frac{1}{2} \sigma_t \frac{\partial \sigma_t}{\partial x} \right) dt + \sigma_t \, dW_t.$$

Conversely, suppose X_t solves the Itô equation

$$dX_t = \mu_t \, dt + \sigma_t \, dW_t.$$

Then X_t also solves the Stratonovich equation

$$dX_t = \left(\mu_t - \frac{1}{2} \sigma_t \frac{\partial \sigma_t}{\partial x} \right) dt + \sigma_t \circ dW_t.$$

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In International conference on machine learning, pages 214–223. PMLR, 2017.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. arXiv preprint arXiv:1806.07366, 2018.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. arXiv preprint arXiv:2005.08926, 2020.

Patrick Kidger, James Foster, Xuechen Li, and Terry Lyons. Efficient and accurate gradients for neural sdes. arXiv preprint arXiv:2105.13493, 2021a.

Patrick Kidger, James Foster, Xuechen Li, Harald Oberhauser, and Terry Lyons. Neural sdes as infinite-dimensional gans. arXiv preprint arXiv:2102.03657, 2021b.

Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In International Conference on Artificial Intelligence and Statistics, pages 3870–3882. PMLR, 2020.