Paper review: *Score-Based Generative Modeling through Stochastic Differential Equations*

Grégoire Retourné
ENSAE, France
gregoire.retourne@gmail.com

Abstract

This memoir focuses on generative modeling [2] [6] [4], and more precisely score based generative modeling. While other downstream tasks such as classification or even detection [12] [15] [7] have been successfully tackled with deep learning, generative modeling has proven to be more challenging. First promising results were obtained with Generative Adversarial Networks (GANs) [5], but they are known to be hard to train (as it is a highly unstable process) and to suffer from mode collapse. More recently, diffusion models [8] have been proposed, and have shown great results on image generation, take for instance OpenAI DALLE-2 [13] a stunning text-to-image model. However, diffusion models are computationally expensive, and still cannot attain the same generation speed as GANs. Similarly to diffusion models, score based generative models [17] attempt to generate images by iteratively applying a sequence transformations to a noise sample, to go from the noise distribution (which is known) to the data distribution (which is learnt). This denoising *process is done by estimating a particular gradient field (know as the score) on the perturbed data distribution, to then learn how to revert the noise corruption step. High-level wise, this process iteratively transports a sample from a particular distribution, to another one, which is an interesting theory that is already applied to other tasks [3] [10]. In this study, inspired by the work and code provided by Song et al. [17], we conducted experiments using a score based generative model trained on the FashionMNIST dataset, that uses Stochastic Differential Equations (SDE) to revert the noise corruption step. Our code is made available at* `https://github.com/greg2451/score-based-generative-modeling.git` *and allows to easily train a score based generative model on FashionMNIST, and to play with image generation using various SDE solvers.*

1. Score-Based Generative Modeling: classical approach

1.1. The score function

In order to generate samples from a distribution, we first need a way to represent it. A common technique is to directy model the probability density function (p.d.f.) using

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

where $f_\theta(\mathbf{x})$ is a real-valued function parameterized by a learnable parameter θ, and Z_θ is a normalizing constant such that the integral of $p_\theta(\mathbf{x})$ over $\mathbf{x}$ is 1.

We then train θ by maximizing the log-likelihood of the data:

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i). \tag{2}$$

But the issue in equation 2 is that it requires to compute Z_θ, which is often feasible. One particular solution in this case, is to restrict the class of estimators to tractable ones, where the normalizing constant is equal to one by design, but this is a rather big limitation.

Another less restrictive solution is to estimate the gradient of the log-likelihood, which is called the score function, and which does not depend on the normalizing constant:

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = -\nabla_\mathbf{x} f_\theta(\mathbf{x}) - \underbrace{\nabla_\mathbf{x} \log Z_\theta}_{=0} = -\nabla_\mathbf{x} f_\theta(\mathbf{x}). \tag{3}$$

Then, to train the model, we minimize the Fisher divergence between the model and the data distributions:

$$\mathbb{E}_{p(\mathbf{x})}[\|\nabla_\mathbf{x} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2] \tag{4}$$

The issue here is that if we are trying to estimate a distribution function, it is highly unlikely that we have access to the score function of the data distribution $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, which is required to compute the Fisher divergence. This is where score matching [19] comes in, which is a family of methods that minimize the Fisher divergence without knowledge of the ground-truth data score. Finally, this whole process allows great flexibility in the choice of the model, since no assumptions are made on the form of $f_\theta(\mathbf{x})$, except that it should be correct in terms of dimensions.

1.2. Langevin dynamics and its pitfalls

Once a score function has been estimated, let's use it to generate samples from the distribution, using an iterative procedure called Langevin dynamics [16].

This method provides a Markov Chain Monte Carlo [18] (MCMC) procedure, that first starts with a sample drawn from a prior distribution (from which we have knowledge), and then iterates the following equation:

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i, \quad i = 0, 1, \cdots, K, \tag{5}$$

where $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a Gaussian noise, and ϵ is a step size. When K is sufficiently large, $\mathbf{x}_K$ converges to a sample from $p(\mathbf{x})$ under some regularity conditions.

This allows us to sample solely from the score function! However, in practice, this approach did not have much success. The culprit is the score function, which is unreliable in low density regions. In fact, this is due to the loss function, the Fisher divergence, which is a l2-distance pondered by the density of the data, the latter being very small in low density regions.

1.3. Introducing noise perturbation

The previous approach fail to the score estimation function not being able to grasp the low density regions of the data. To overcome this issue, the authors proposed to learn the aforementioned score function on perturbed data, instead of the original data. This allows to populate the low-density regions with points, and thus alleviate the inefficiency in these regions. Indeed, too much noise would mean losing the initial signal, but too little noise would not be enough to populate the low-density regions, there is a trade-off to find. To do so, the authors propose to use multiple scales of noise perturbations at the same time, ensuring to learn jointly from the original data and the perturbed data. They learn a single scoring function being a weighted sum of the scoring functions of each perturbed data. The loss used is a weight sum of the Fisher divergences for all noise scales, the weight often being bigger for perturbations with bigger noise (and in their case, proportional to the variance of the noise).

2. Infinite number of noise scales using Stochastic Differential Equations

Intuitively, finding the right amount of noise to inject in the process is difficult, so having a high number of noise scales helps a lot. The authors here have managed to do the best thing in that regard: have an infinite number of noise scales, by using a continuous-time stochastic process, which is a solution of a stochastic differential equation (SDE) [17].

2.1. Continuously perturbing the data distribution

When we had a finite number of noise scales, the problem was applied to sequences. Now, since we want to have an infinite number, it becomes an infinitesimal problem, hence we can use differential equations, and functions of time. The noising equation is now:

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

Each new variables are roughly their equivalent in a continuous setting. For example, $\mathrm{d}\mathbf{w}$ is an infinitesimal white noise, and $\mathrm{d}t$ is an infinitesimal time step. Tailoring the drift coefficient $\mathbf{f}$ and diffusion coefficient g allows us to control the amount of noise injected at each time step, and we can easily find the equivalent of the previous noise scales.

Similarly, the reverse process [1] is (continuous equivalent of equation 5) is:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}. \tag{7}$$

The goal is to solve this equation backward in time, continuously from the step T to 0.

Importantly, to solve this reverse SDE, we need to estimate the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ of the distribution at each intermediate time step. Indeed, since now we have an infinite number of noise scales, the loss function becomes:

$$\mathbb{E}_{t \in \mathcal{U}(0,T)} \mathbb{E}_{p_t(\mathbf{x})}[\lambda(t) \|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t)\|_2^2], \tag{8}$$

This gives us all the theoretical tools to tailor a solution to the problem, but how de wo do in practice ?

2.2. Solving SDEs in practice

There are practical solutions to solve SDEs.

First, any numerical SDE solvers can be used. For instance, the Euler-Maruyama method is a simple way to solve SDEs, but it is not very accurate, and it is not stable for large time steps. As this method approximates the continuous process with a discrete one, it will simply solve the previous discrete equation (equation 5) with a small time step Δt, hence it is not very interesting. There are

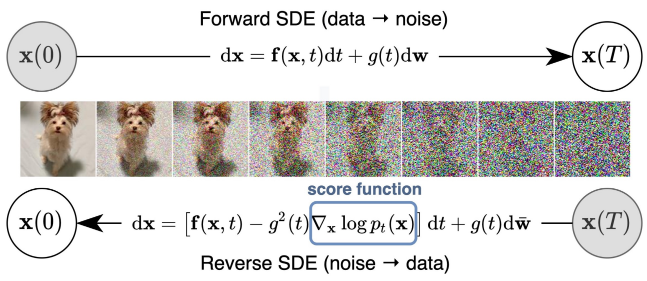
Forward SDE (data → noise)
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\,\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$
$\mathbf{x}(T)$
Reverse SDE (noise → data)

Figure 1. Sampling and reverse-sampling with SDE

other methods, such as the Milstein method or the stochastic Runge-Kutta method, which are more adapted to solve reverse SDEs.

Second, the authors proposed to use a predictor-corrector method. It consists in using a predictor to estimate the next step, based on any classic numerical method, and then use a corrector which will use the score function to enhance the prediction. This method proved to attain state-of-the-art results, and is very efficient.

Finally, the authors proposed to solve this SDE as an Ordinary Differential Equation (ODE). Indeed, any SDE can be transformed into an ODE, which is called a probability flow ODE.

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t. \qquad (9)$$

The last method gives access to exact log-likelihoods computation, which in terms allows to do fully controllable generation.

3. Experiments

The code made available by the authors at https://github.com/yang-song/score_sde_pytorch, allowed us to reproduce some of their experiments on a smaller dataset, the Fashion-MNIST [20] dataset. The code of our experiment is available here at https://github.com/greg2451/score-based-generative-modeling.git and allows to easily train a score based generative model on FashionMNIST [20], and to play with image generation using various SDE solvers. All instructions are available in the `README` of the repository.

3.1. Training

We trained a score-based generative model on the FashionMNIST [20] dataset, the score function is a convolutional[12] model based on a U-Net [14] architecture with 1,115,425 trainable parameters.

We used the following hyperparameters:

- batch size: 64
- learning rate: 10^{-4}
- number of epochs: 100
- sigma: 25 (for the marginal probability)

The training took 10 minutes on a standard cloud GPU to attain a loss of approximately 21. This loss value was deemed sufficient to generate similar images to the original dataset with high fidelity. The weights of this trained model are available in the repository, in the `pretrained_models` folder. This model is optimized for inputs of size 28x28 in black and white (1 channel).

We trained a second model, this time for images in color (3 channels) of size 32x32, on CIFAR-10 [11] with the same hyperparameters. This model is adapted from another architecture usually used for image in-painting [9] and modified to include cross-channel connections, this time with 1,092,643 trainable parameters. However, we did not manage to train this model to a satisfactory loss (value stuck around 1000), hence generated images were not interpretable (strange noise patterns). As we did not have enough time to investigate the issue, we did not release those weights, but the architecture is available in the repository in the `models` folder.

3.2. Generation

After finishing the training of the score function, we then proceeded to generate images from the model. We used the three methods proposed by the authors, namely the Euler-Maruyama method, the predictor-corrector method, and the probability flow ODE method.

We then studied the impact of the hyperparameters for each method, all experiments were run on the FashionM-NIST [20] dataset with a batch size of 64.

3.2.1 Euler-Maruyama

This method was the fastest to generate images, with approximately 200 denoising steps per second, which gave us great fidelity images in between 5 and 10 seconds. The resulting images were very similar to the original dataset, with a few artifacts, but overall very good results, and nice sharpness. Images generated by three different number of steps are shown in figure 2, respectively 100, 250 and 2000 steps.

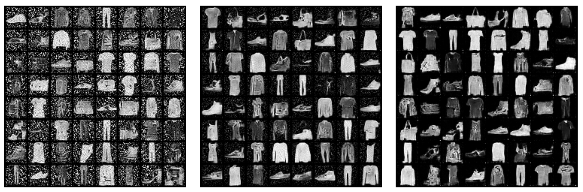

Figure 2. Images generated by the Euler-Maruyama method with 100, 250 and 2000 steps.

3.2.2 Predictor-corrector

As expected, the cost is twice as high as the Euler-Maruyama method, as it requires two model evaluations per step, so we had approximately 100 denoising steps per second. This method has two hyperparameters to tune, the number of total steps and the signal-to-noise ratio. The signal-to-noise ratio impacts the correction step: it controls the amount of signal inserted in the image correction. Intuitively, the more signal we insert at the correction step, the more sharp the image will be, with strong borders, but the more artifacts will be present. In contrast, a low signal-to-noise ratio will roughly give the same output than the Euler-Maruyama method, but with a higher computational cost (2 model evaluations per step instead of 1). The number of steps and signal-to-noise ratio seems not to be correlated, this is why we chose to show the results for a fixed number of steps (2000) and varying signal-to-noise ratios, respectively 0.2, 0.5 and 1.0, in figure 3.

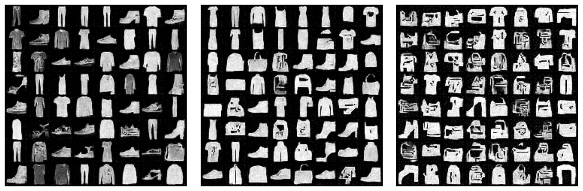
Figure 3. Images generated by the predictor-corrector method with 2000 steps and signal-to-noise ratios of 0.2, 0.5 and 1.0.

As expected, the highest signal-to-noise ratio gives the sharpest images, but with a lot of artifacts, the best value being around 0.2, which gives the best images in terms of sharpness and fidelity.

3.2.3 Probability flow ODE

This method was the slowest to generate images, since it relies on a black-box ODE solver, which is not optimized for the problem at hand. While the two other methods gave roughly similar images, this method really stood out, with a very different, more original output. The unique hyperparameter of this method is the error tolerance in the ODE solver, which controls the number of function evaluations required by the solver. The lower the error tolerance, the more evaluations are required, and the more time it takes to generate an image, but the better the image quality. It is interesting to notice, that even at very high error tolerance (and hence very low number of steps), the convergence seems to take a completely different path than the two previous method, with much more variety in the noise patterns. We chose to show the results for a fixed error tolerance of 0.1, 0.001 and 1e-6, which respectively required 32, 92 and 542 evaluation steps to converge, in figure 4.

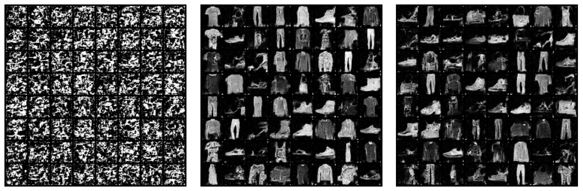
Figure 4. Images generated by the probability flow ODE method with an error tolerance of 0.1, 0.001 and 1e-6.

4. Conclusions

In this work, we have presented a score based generative model trained on the FashionMNIST [20] dataset, using various SDE [1] solvers. We have assessed that the model is able to generate images that are visually similar to the ones in the dataset, and that the SDE solver has a significant impact on the quality of the generated images. Moreover, tuning hyperparameters of the generation part (in SDE solvers method) gives the ability to find a trade-off between generation speed and image quality. As expected, training this kind of model is much simpler (and stabler) than training a GAN [5], with still good performances. Alas, the speed of generation can be an issue, especially running from CPU since it requires as many calls to the score model as denoising steps in the process. More importantly, score-based generative modeling is a promising field, and we believe that it will be extensively applied to other tasks than image generation, such as audio generation [3] [10], since its goal is to iteratively move from a distribution to another one.

References

[1] Brian Anderson et al. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982. 2, 4

[2] Florian Bordes, Sina Honari, and Pascal Vincent. Learning to Generate Samples from Noise through Infusion Training, Mar. 2017. arXiv:1703.06975 [cs, stat]. 1

[3] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. WaveGrad: Estimating Gradients for Waveform Generation, Oct. 2020. arXiv:2009.00713 [cs, eess, stat]. 1, 4

[4] Yilun Du and Igor Mordatch. Implicit Generation and Generalization in Energy-Based Models, June 2020. arXiv:1903.08689 [cs, stat]. 1

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. arXiv:1406.2661 [cs, stat]. 1, 4

[6] Anirudh Goyal, Nan Rosemary Ke, Surya Ganguli, and Yoshua Bengio. Variational Walkback: Learning a Transition Operator as a Stochastic Recurrent Net, Nov. 2017. arXiv:1711.02282 [cs, stat]. 1

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, Dec. 2015. arXiv:1512.03385 [cs]. 1

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, Dec. 2020. arXiv:2006.11239 [cs, stat]. 1

[9] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017. 3

[10] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. DiffWave: A Versatile Diffusion Model for Audio Synthesis, Mar. 2021. arXiv:2009.09761 [cs, eess, stat]. 1, 4

[11] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 3

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 1, 3

[13] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, Apr. 2022. arXiv:2204.06125 [cs]. 1

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. arXiv:1505.04597 [cs]. 3

[15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, Apr. 2015. arXiv:1409.1556 [cs]. 1

[16] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution, Oct. 2020. arXiv:1907.05600 [cs, stat]. 2

[17] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. 1, 2

[18] Joshua S. Speagle. A Conceptual Introduction to Markov Chain Monte Carlo Methods, Mar. 2020. arXiv:1909.12313 [astro-ph, physics:physics, stat]. 2

[19] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011. 2

[20] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, Sept. 2017. arXiv:1708.07747 [cs, stat]. 3, 4