Diffusion and Score-based Generative Modeling

Simone Petruzzi-1811872 Domenico Tersigni-

August 2023

1 Introduction

Given a dataset $\{x_1, x_2, ..., x_n\}$ where each point is drawn independently from an underlying distribution $p(x)$, the goal of generative modelling is to fit a model to the data distribution, such that we can synthetize new data points at will by sampling from the model of the distribution.
Let's say p_{data} is our model and $x_i \sim p_{data}$. We want to find a single probability distribution, minimizing the distance from p_{data} to p_θ. Afterwards we can generate samples from p_θ. In order to estimate the model we can use as starting point a Gaussian distribution (basically a computational graph composed by two layers). However a single Gaussian is too simple and we want to leverage on a bigger and deeper computational graph. Thus we want to use deep neural networks to represent complex data distributions. This kind of architectures typically convert high dimensional input into one dimensional output $f_\theta(x)$, this output should be converted into a probability density and for this reason we take the exponential of the output in order to make it always positive. Then we normalize by means of a constant Z_θ obtaining:

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

Z_θ by definition should be computed by evaluating the high dimensional integral of the exponential function of out θ, over all possible values of x in the space:

$$Z_\theta = \int e^{f_\theta(x)}\, dx \tag{2}$$

In case of deep neural networks the computation of this integral becomes intractable (NP-complete problem). In order to tackle with the intractability of the normalizing constant, this proposal aims to work with **score functions**.

2 Score-based generative modeling

The score funcion is a vector field that gives the direction where the density fucntion grows most quickly. We define the score of a probability density $p(x)$ to be:

$$p(x) = \nabla_x \; log \; p(x) \tag{3}$$

Score function is good since in anytime given the density function we can compute the score function very easily by simply texting the dervative. Conversely given the score function we can recover the density function in principle by computing the integral. Thus, we argue that score function mantains preseved all mathematical information of the density function, being computationally more easy to deal with. Indeed with score functions we don't have any normalization restriction:

$$\nabla_x \; log \; p_\theta(x) = \nabla_x f_\theta(x) \; - \nabla_x \log Z_\theta \tag{4}$$

the term $\nabla_x \log Z_\theta$ is always 0 because it is the gradient of a constant. Thus the score function is the gradient of the deep neural network $f_\theta(x)$. We define the score model $s_\theta(x)$ as:

$$\nabla_x f_\theta(x) = s_\theta(x) \tag{5}$$

and we want to develop a technique that allows us to train $s_\theta(x)$, in order to estimate the underlying score function from a limited set of training data points

2.1 Score matching

We must train our score model to be close to our ground-truth data score function. Mathematically we can capture the distance between the two vector fields of score by the **Fisher divergence objective**:

$$\frac{1}{2} E_{p_{data}(x)}[||\nabla_x \; log \; p_{data}(x) - s_\theta(x)||_2^2] \tag{6}$$

However Fisher divergence can not be computed since we don't know the ground trith value of the data score function $\nabla_x \; log \; p_{data}(x)$. There is a way for addressing this challenge and the method is called **score matching**. Score matching uses integration by parts of Gauss' theorem to convert Fisher divergence into the following equivalent objective:

$$E_{p_{data}(x)}[\frac{1}{2}||s_\theta(x)||_2^2 + trace(\nabla_x \; s_\theta(x))] \tag{7}$$

where $\nabla_x \; s_\theta(x)$ denotes the Jacobian of $s_\theta(x)$. We call it score matching objective, and it is equivalent to the fisher divergence up to a costant. Since constants do not affect optimization, their score matching objective defines the same optimum as the Fisher divergence. In score matching there is no dependency on the score function of the data distribution anymore. Moreover the expectation in score matching can be efficiently be approximated by using empirical mean:

$$E_{p_{data}(x)}[\frac{1}{2}||s_\theta(x)||_2^2 + trace(\nabla_x \; s_\theta(x))] \approx \frac{1}{N}\sum_{i=1}^{N}[\frac{1}{2}||s_\theta(x)||_2^2 + trace(\nabla_x \; s_\theta(x_i))] \tag{8}$$

again this is not scalable to compute since we need one forward propagation to compute the first element of the score function output and we need a backpropagation to compute the first element on the diagonal of its Jacobian. This procedure has to be repeated multiple times until we recovered all diagonal elements on the Jacobian (trace is given by the sum over the diagonal elements). Thus, number of backpropagations needed is proportional to the dimenstionality of our data points. It follows that the whole procedure when modelling high dimensional data like images, requires a lot of backpropagations making this naive score matching approach not scalable.

2.2 Sliced score matching

This approach aims to solve the problem of naive score matching: the basic intuition is that we want to convert high dimensional problem to a one-dimensional one (since it is much easier to solve).
The idea is to leverage on random projection, this is done in order to approximate $trace(\nabla_x\ s_\theta(x))$ and get one-dimensional scalar fields. Again to capture this intuition we use sliced Fisher divergence:

$$\frac{1}{2}E_{p_v}E_{p_{data}(x)}[(\mathbf{v}^T\nabla_x\ log\ p_{data}(x) - \mathbf{v}^T s_\theta(x))^2] \tag{9}$$

also there we apply integration by parts and obtain sliced score matching:

$$E_{p_v}E_{p_{data}(x)}[\mathbf{v}^T\nabla_x s_\theta(x)\mathbf{v}\ + \frac{1}{2}||s_\theta(x)||_2^2] \tag{10}$$

$\mathbf{v}^T\ \nabla_x s_\theta(x)\mathbf{v}$ is much more scalable to compute. It is not hard to find that we can rewrite:

$$\mathbf{v}^T\nabla_x s_\theta(x)\mathbf{v} = \mathbf{v}^T\nabla_x(\mathbf{v}^T s_\theta(x)) \tag{11}$$

we just need one forward propagation to get the output $s_\theta(x)$ and then we can directly compute the inner product between $\mathbf{v}^T$ and s_θ.
Sliced score matching provides score estimation for the original *unperturbed* data distribution.

2.3 Denoising score matching

Another approach is denoising score matching which estimates the scores of *perturbed* data distribution.
Indeed, the idea here is to perturb the original data distribution employing a perturbation kernel q_σ, then estimate the score function of the noisy data density instead of the original one.

$$p_{data}(x)\ \longrightarrow q_\sigma(\tilde{x}|x)\ \longrightarrow q_\sigma(\tilde{x}) \tag{12}$$

We employ score matching to estimate the score of the perturbed data distribution $q_\sigma(\tilde{x})$, and when the perturbation is very small, we can approximately consider the score function of the noisy data density as equivalent to the score function of the noise-free data density. The denoising score matching objective function has been proved equivalent to the following

$$\frac{1}{2}\mathbb{E}_{p_{data}(x)}\mathbb{E}_{q_\sigma(\tilde{x}|x)}[||s_\theta(\tilde{x}) - \nabla_{\tilde{x}} log q_\sigma(\tilde{x}|x)||_2^2] \tag{13}$$

in this approach we must take care of the trade-off between the quantity of the noise injected. If you want to work well in estimating score functions of noise-free data densities, you need a very small σ. However when σ is too small, the variance of the objective function becomes bigger and eventually explodes.

2.4 Langevin dynamics

Now that we have trained our score model to estimate the underlying score function $s_\theta \approx \nabla_x logp(x)$ we want a method for generating new data points from the given vector field of score functions.
Langevin dynamics is an iterative procedure that allows to draw samples from our score model. The procedure is as follows:

- Initialize the sample from an arbitrary prior distribution: $x^o \sim \pi(x)$, then repeat this for $z^t \leftarrow 1, 2, ..., T$ $z^t \sim \mathcal{N}(0, I)$
- In each sampling step, first generate a random Gaussian vector from the standard Gaussian distribution, then modify x according to the following recurrence:

$$x_t \leftarrow x^{t-1} + \frac{\epsilon}{2}\nabla_x log\ p(x^{t-1}) + \sqrt{\epsilon}\ z^t \tag{14}$$

We basically update the previous sample using our score function, plus a scaled version of the Gaussian noise vector. If $\epsilon \to 0$ and $T \to \infty$, we are guaranteed to have $x^t \sim p(x)$.
It becomes very natural to replace the score function in Langevin dynamics with our score model ($s_\theta(x) \approx \nabla_x log\ p(x)$), then we can generate new data samples, defining a new generative model.

2.5 Naive score-based generative modeling and its pitfalls

This procedure has drawbacks: we have at first problem in estimating correctly the score function in low data density regions. In those regions we usually don't have enough samples to estimate the score function accurately. Remember that score matching minimizes the expected squared error of the score estimates, *i.e.*, $\frac{1}{2}\mathbb{E}_{p_{data}(x)}[||\nabla_x\ log\ p_{data}(x) - s_\theta(x)||_2^2]$. In practice the expectation is always estimated using i.i.d samples $\{x_i\}_{i=0}^N \sim p_{data}(x)$ and when there is a lack of them, the estimation is inaccurate.

Also, Langevin dynamics, when two modes of the data distribution are separated by low density regions, will be not able to recover the relative weights of these two modes in reasonable time. Langevin dynamics can produce correct samples in theory, but it may require a very small step size and a very large number of step to mix, increasing drastically the computation time.

3 Score-based generative modeling with multiple noise perturbations

In the previous section we talked about score-based generative modeling and it's problems. The main problem of that approach is bad estimation of the score function in low data density regions. We can address this challenge by injecting Gaussian noise to perturb our data data points and then doing the exact same procedure on the perturbed data. After having added enough Gaussian noise we provide useful directional information for Langevin dynamics, also score functions of noisy data densities are much easier to be estimated accurately.

However by simply injecting Gaussian noise we don't solve the problem, because after perturbing data points, the noisy data densities distances, are no longer good approximation for the original data density. How to choose the right level of noise ? To solve this, we can use multiple noise levels simultaneously and consider a noise conditional score model. This model takes noise level σ as additional input dimension to the model and the output corresponds to the score function of the perturbed data density by σ. We perturb the data distribution $p(x)$ with a Gaussian noise $\mathcal{N}(0, \sigma_i^2 I), i = 1, 2, ..., L$, obtaining the noise-perturbed distribution:

$$p_{\sigma_i}(x) = \int p(y)\ \mathcal{N}(x; y, \sigma_i^2 I) dy \tag{15}$$

We choose σ_1 large enough such that we don't have problems in low density regions, and σ_L small enough to minimize the effect on data.
Built upon this intuition we improve score-based generative modeling by first perturbing data as written above, then simultaneously estimate scores corresponding to all noise levels by training a single noise conditional core network.

3.1 Noise conditional score network

We aim to train this network to jointly estimate the scores of all perturbed data distributions, $i.e., \forall \sigma \in \{\sigma_i\}_{i=1}^{L} : s_\theta(x, \sigma) \approx \nabla_x log q_\sigma(x)$. We call $s\theta(x, \sigma)$ a $Noise\ Conditional\ Score\ Network$ (NCSN).

3.2 NCSN learning with score matching

Again for training this kind of network we can leverage on score matching techniques (both sliced and denoising score matching). Here we adopt denoising score matching loss function:

$$\frac{1}{2N} \sum_{i=1}^{N} \lambda(\sigma_i)\ \mathbb{E}_{p_{\sigma_i}}(x)[||\nabla_x\ log p_{\sigma_i}(x) - s_\theta(x, \sigma_i)||_2^2] \tag{16}$$

and in the special case of Gaussian perturbations, we can simplify the denoising objective as:

$$\frac{1}{2N} \sum_{i=1}^{N} \lambda(\sigma_i)\ \mathbb{E}_{p_{\sigma_i}}(x)[||s_\theta(\tilde{x}, \sigma) + \frac{\tilde{x} - x}{\sigma^2}||_2^2] \tag{17}$$

$\lambda(\sigma_i)$ is a positive weighting function (often chosen to be $\lambda(\sigma_i) = \sigma_i^2$), which balances the scales of score matching across all noise levels, making helpful for optimization.

3.3 NCSN sampling with annealed Langevin dynamics

Given the trained noise conditional score model we can use annealed Langevin dynamics to produce samples. We start this process by initializing the samples from some fixed prior distribution, then we first apply Langevin dynamics to sample from the score model with the biggest perturbation noise, let's say $q_{\sigma_1}(x)$. Next we run Langevin dynamics to sample from $q_{\sigma_2}(x)$, starting from the final samples of the previous simulation. This procedure continues until eventually we reach the score function with the smallest noise level, running Langevin dynamics to sample from $q_{\sigma_L}(x)$, which is close to $p_{data}(x)$ when $\sigma_L \approx 0$.

When σ_1 is sufficiently large, the low density regions of $q_{\sigma_1}(x)$ become small and the modes become less isolated. Also as discussed previously the score estimation is more accurate. We can argue that Langevin dynamics produces good samples for $q_{\sigma_1}(x)$, and also that those samples are likely to come from high density regions of $q_{\sigma_1}(x)$, meaning that this is valid also for $q_{\sigma_2}(x)$ (given that $q_{\sigma_1}(x)$ and $q_{\sigma_2}(x)$ slightly differ each other), and so on. Thus as score estimation and Langevin dynamics perform better in high density regions, samples from $q_{\sigma_{i-1}}(x)$ provide good initial point for $q_{\sigma_i}(x)$. Finally we obtain good samples of good quality from $q_{\sigma_L}(x)$ that is our last step.

4 Score-based generative modeling with stochastic differential equations

As already discussed, adding multiple noise scale is critical for the success of score-based generative models. In this section the aim is to generalize the previous framework (that is using a finite number of noise levels), to use an infinite number of noise levels. By generalizing the number of noise scales to infinity, we obtain not only high quality samples, but also, among others, exact log-likelihood computation, and controllable generation. Specifically we consider a continuum of distribution that evolve over time according to a diffusion process that progressively diffuses a data point itno random noise, and is given by a SDE (Stochastic Differential Equation) that does not depend on the data and has no trainable parameters. After the training procedure, by reversing the previous process, we can mold random noise into data for sample generation. This reverse process relys on a reverse-time SDE, which can be derived from the forward SDE given the score of the marginal probbility densities as function of time. Therefore we can approximate the reverse-time SDE with a time-dependent neural network which estimates the scores, and then produce samples by simply using numerical SDE solvers.

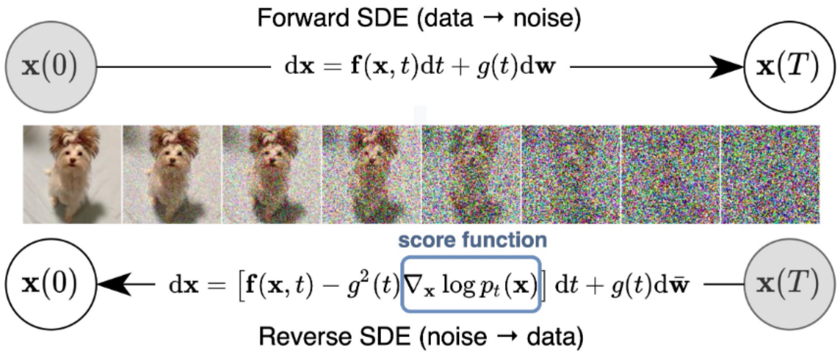
Forward SDE (data → noise)
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\boxed{\nabla_{\mathbf{x}} \log p_t(\mathbf{x})}\right] \mathrm{d}t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

Figure 1: SDE model

4.1 Perturbing data with SDEs

When the number of noise scales approaches infinity, we essentially perturb the data distribution with continuously growing levels of noise. The perturbation procedure is a continuous time stochastic process. Our goal is to construct a diffusion process that is solution of a stochastic differential equation. This diffusion process $\{x(t)\}_{t=0}^{T}$ can be modeled as the solution of the following SDE:

$$dx = f(x, t)dt + g(t)dw \tag{18}$$

where $f(., t) : \mathbb{R}^d \to \mathbb{R}^d$ is a vector-valued function called drift coefficient of $x(t)$ (controls deterministic properties of the stochastic process), $g(.) : \mathbb{R} \to \mathbb{R}$ is a scalar function known as the diffusion coefficient of $x(t)$, and w is the standard Wiener process. The dw_t can be seen as the infinitesimal Gaussian noise the each time is injected.
The solution of a stochastic differential is continuous collection of random variables that

trace stochastic trajectories as the time grows. Let $p_t(x)$ denote the marginal probability density function of $x(t)$ with $t \in [0, T]$, clearly $p_0(x)$ is the data distribution with no perturbation. After perturbing the data distribution with enough noise , $p_T(x)$ becomes close to a tractable noise distribution $\pi(x)$ that is called prior distribution.

A specific differential equation that we can use for this purpose (choice of SDE isn't unique) could be:

$$dx = e^t \ dw \tag{19}$$

it perturbs data with a Gaussian noise of mean zero and exponentially growing variance

4.2 Reversing the SDE for sample generation

For a finite number of noise scales, we can generate samples by reversing the perturbation process using annealed Langevin dynamics, i.e, sequentially sampling from each noise perturbed distribution using Langevin dynamics. For infinite noise scales we can do the same as the Langevin dynamics by reversing the forward SDE. Thus by starting form samples of $x(T) \sim p_T$ and reversing the process, we can obtain samples $x(0) \sim p_0$. It was proved that the reverse of a diffusion process is also a diffusion process, and any SDE has corresponding reverse SDE. The reverse SDE in our case is:

$$dx = [f(x,t) - g(t)^2 \nabla_x logp_t(x)]dt + g(t)d\bar{w} \tag{20}$$

where $\bar{w}$ is a standard Wiener process when the time flows backwards from T to 0, and dt is the infinitesimal reverse (negative) time step. In order to estimate the reverse SDE we must compute $\nabla_x logp_t(x)$, which is exactly the score function of $p_t(x)$

4.3 Estimating the reverse SDE with score-based models and score matching

In order to solve the reverse SDE is required the terminal distribution $p_T(x)$ and the score function $\nabla_x logp_t(x)$. To estimate the score function we can train a time-dependent score-based model $s_\theta(x,t)$. The objective function is a continuous weighted combination of Fisher divergence:

$$\theta^* = \underset{\theta}{argmin}\, \mathbb{E}_t\{\lambda(t)\mathbb{E}_{x(0)}\mathbb{E}_{x(t)|x(0)}[||s_\theta(x(t),t) - \nabla_x logp_{0t}(x(t)|x(0))||_2^2]\} \tag{21}$$

Here $\lambda : [0,T] \to \mathbb{R}_{>0}$ is a positive weighting function and $x(0) \sim p_0(x)$ and $x(t) \sim p_{0t}(x(t)|x(0))$. As before, this combination of Fisher divergence can be optimized efficiently with score metching methods and once the model is trained we can plug it into the reverse SDE.
When $\lambda(t) = g^2(t)$, we have an important connection between the combination of Fisher divergence and the Kullback-Leibler divergence form p_0 to p_θ under some regularity conditions. Since the KL divergence is directly related to maximal likelihood training, by minimizing the score function with this particular likelihood of weighting functions, we are implicitly maximizing likelihoods.

$$KL(p_0(x)||p_\theta \leq \frac{1}{2}\mathbb{E}_{t\sim Uniform[0,T]}[\lambda||\nabla_x logp_t(x) - s_\theta(x,t)||_2^2] + KL(p_t||\pi) \tag{22}$$

$KL(p_t||\pi) \approx 0$ if T is large enough and this does not affect the optimization procedure, since it does not depend on model parameter θ.

4.4 Solving the reverse SDE

After having trained a time-depedent score model s_θ, we can use it to construct the reverse-time SDE and generate samples. We can basically use any numerical stohastic differential equation solver: a simple approach is Euler-Maruyama, which is a stochastic generalization to the classical Euler solver fo ordinary differential equations. When applied to our estimated reverse SDE, it discretizes the SDE using finite time steps and small Gaussian noise: specifically, it chooses a small negative step $\Delta t \approx 0$, initializes $t \leftarrow T$, then iterates the following procedure until $t \approx 0$:

$$\begin{aligned} &\Delta x \leftarrow [f(x,t) - g^2(t)s_\theta(x,t)]\Delta t + g(t)\sqrt{|\Delta t|}z_t \\ &x \leftarrow x + \Delta x \\ &t \leftarrow t + \Delta t \end{aligned} \tag{23}$$

where $z \sim \mathcal{N}(0, |\Delta t|I)$. Aside the Euler-Maruyama method, other numerical SDE solver can be used.

Since we have additional information that can be used, we can improve simple generic SDE solvers. Since we have a score-based model $s_\theta(x,t) \approx \nabla_x log\ p_t(x)$, and we only care about sampling for each marginal distribution $p_t(x)$, we can employ score-based MCMC (Markov Chain Monte Carlo) approaches such Langevin MCMC (Parisi, 1981) to fine-tune the trajectories obtained by the solver. Specifically we can use a predictor-corrector method. At each time step, the numerical SDE solver first gives an estimate of the sample at next time step (playing role of "predictor"). Then, the score based MCMC approach corrects the marginal distribution of estimated sample (playing the role of "corrector")

4.5 Probability flow and connection to neural ODEs

With this kind of continuous SDE approach, we do not only improve the empirical performace (sample generation), but also we can accurately compute the exact log-likelihood. This requires to convert the stochastic differential equation into an ordinary one without changing its marginal distributions $\{p_t(x)\}_{t\in[0,T]}$. Thus by solving this ODE, we can sample from the same distributions as the reverse SDE. The corresponding ODE of an SDE is named probability flow ODE, given by:

$$dx = [f(x,t) - \frac{1}{2}g^2(t\nabla_x log\ p_t(x))]dt \tag{24}$$

Again this relys only on the score function and we can plug our conditional score model into the ODE in order to solve it. When we plug out conditional score model into the probability flow ODE, it becomes a special case of neural ODE. As such, the probability flow ODE inherits all properties of neural ODEs or continuous normalizing flows, including exact log-likelihood computation. Specifically, we can leverage the instantaneous change-of-variable formula o compute the unknown data density p_0 from the known prior density p_T with numerical ODE solvers.

5 Experiments

in this section we will describe the implementation in pytorch and the obtained results.

5.1 Time-dependent Score-Based Model

There are no special constraint in defining the architecture of a time-dependent score model, except that their output should have the same dimensionality as the input, and they should be conditioned on time.
The architecture used in this case is the U-net architecture as the backbone for the score network $s_\theta(x,t)$ since it works very well and it provides the the same dimensionality as the input. Here the time information is incorporated via Gaussian random features. More specifically we first sample $\omega \sim \mathcal{N}(0, s^2 I)$ which is subsequently fixed for the model (i.e. not learnable). For a time step t, the corresponding Gaussian random feature is defined as

$$[\sin(2\pi\omega t); \cos(2\pi\omega t)], \tag{25}$$

feature can be used as an encoding for time step t so that the score network can condition on t by incorporating this encoding.
We then rescale the output of the U-net by $1/\sqrt{\mathbb{E}[\|\nabla_{\mathbf{x}} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2]}$, the optimal $s_\theta(\mathbf{x}(t),t)$ has an ℓ_2-norm close to $\mathbb{E}[\|\nabla_{\mathbf{x}} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))]\|_2$, due to which the rescaling helps capture the norm of the true score. It's worth recalling that the training objective consists of sums of the following form:

$$\mathbf{E}_{\mathbf{x}(t)\sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))}[\|s_\theta(\mathbf{x}(t),t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2].$$

Hence, it's reasonable to anticipate that the optimal score model $s_\theta(\mathbf{x},t) \approx \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))$.
Also we use exponential moving average (EMA) of weights when sampling, this improves sample quality at the expense of more training effort.

5.2 Training

For training procedure we entirely leveraged on Google Colab.

5.2.1 Training with Weighted Sum of Denoising Score Matching Objectives

First of all we need to specify an SDE that perturbs the data distribution p_0 to p_t:

$$d\mathbf{x} = \sigma^t d\mathbf{w}, \quad t \in [0,1] \tag{26}$$

in which we can see σ^t as continuous time generalization of the noise table $\sigma_1, ..., \sigma_t$ and where $d\mathbf{w}$ is the infinitesimal Gaussian noise. The training objective function, contains sums of the form:

$$\lambda(t)\ \mathbf{E}_{\mathbf{x}(t)\sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))}[\|s_\theta(\mathbf{x}(t),t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2].$$

We can choose the weighting function $\lambda(t) = \frac{1}{2\log\sigma}(\sigma^{2t} - 1)$. When σ is large, the prior distribution, $p_{t=1}$ is

$$\int p_0(\mathbf{y}) \mathcal{N}\left(\mathbf{x}; \mathbf{y}, \frac{1}{2\log\sigma}(\sigma^2 - 1)\mathbf{I}\right) d\mathbf{y} \approx \mathbf{N}\left(\mathbf{x}; \mathbf{0}, \frac{1}{2\log\sigma}(\sigma^2 - 1)\mathbf{I}\right),$$

which is approximately independent of the data distribution and is easy to sample from.

5.2.2 Epochs

We made several experiments on the number of epochs needed to reach a good score-based model. Number of epochs we trained our model varies from 50 to 200. After 50 epochs our score-based model already works well obtaining a solid base for sampling. However we can improve training for 200 epochs. After 200 epochs the improvement of the loss is almost null (as we can see in the image below), thus we decided to stop after 200 epochs.

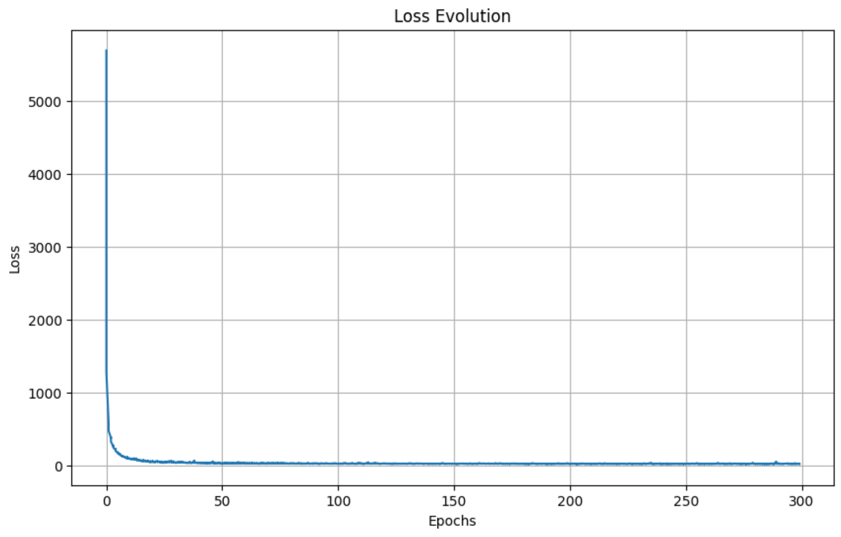

Figure 2: Train loss

5.3 Sampling

For sampling we adopted three different methods

5.3.1 Numerical SDE solver

For any SDE of the form

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

the reverse-time SDE is given by

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

We have chosen the forward SDE to be

$$d\mathbf{x} = \sigma^t d\mathbf{w}, \quad t \in [0, 1],$$

and the reverse-time SDE is given by

$$d\mathbf{x} = -\sigma^{2t} \nabla_{\mathbf{x}} \log p_t(\mathbf{x})dt + \sigma^t d\bar{\mathbf{w}},$$

To sample from our time-dependent score-based model $s_\theta(\mathbf{x}, t)$, we first draw a sample from the prior distribution $p_1 \approx \mathbf{N}\left(\mathbf{x}; \mathbf{0}, \frac{1}{2}(\sigma^2 - 1)\mathbf{I}\right)$, and then solve the reverse-time SDE with numerical methods.

In particular, using our time-dependent score-based model, the reverse-time SDE can be approximated by

$$d\mathbf{x} = -\sigma^{2t} s_\theta(\mathbf{x}, t)dt + \sigma^t d\bar{\mathbf{w}}$$

Numerical methods can be used to solve the reverse-time SDE, such as the [Euler-Maruyama] approach. It is based on a simple discretization to the SDE, replacing dt with Δt and $d\mathbf{w}$ with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, g^2(t)\Delta t\mathbf{I})$. When applied to our reverse-time SDE, we can obtain the following iteration rule

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t + \sigma^{2t} s_\theta(\mathbf{x}_t, t)\Delta t + \sigma^t \sqrt{\Delta t}\mathbf{z}_t, \tag{27}$$

where $\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

5.3.2 Predictor-Corrector method

We can make use of the specific properties of our reverse-time SDE to improve the solutions, apart from using general numerical SDE solvers. With an estimate of the score of $p_t(\mathbf{x}(t))$ from the score-based model, i.e., $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$, we can combine numerical SDE solvers with score-based MCMC techniques, like Langevin MCMC, to enhance the accuracy of the solutions.

Score-based MCMC methods generate samples from a distribution $p(\mathbf{x})$ when its score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is available. For instance, Langevin MCMC performs iterations according to the rule for $i = 1, 2, \cdots, N$:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_i) + \sqrt{2\epsilon}\mathbf{z}_i,$$

where $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\epsilon > 0$ is the step size, and $\mathbf{x}_1$ is initialized from any prior distribution $\pi(\mathbf{x}_1)$. As $N \to \infty$ and $\epsilon \to 0$, the final value $\mathbf{x}_{N+1}$ converges to a sample from $p(\mathbf{x})$ under certain conditions. Thus, when we have $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, we can obtain an approximate sample from $p_t(\mathbf{x})$ by executing multiple steps of Langevin MCMC, using $s_\theta(\mathbf{x}, t)$ in place of $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ in the iteration rule.

Predictor-Corrector samplers combine numerical SDE solvers for the reverse-time SDE with the Langevin MCMC technique. Specifically, we initiate with a single step of the numerical SDE solver, generating $\mathbf{x}_{t-\Delta t}$ from $\mathbf{x}_t$, known as the "predictor" step. Following this, we apply several steps of Langevin MCMC to refine $\mathbf{x}_t$, making $\mathbf{x}_t$ a more accurate sample from $p_{t-\Delta t}(\mathbf{x})$. This procedure is termed the "corrector" step, as MCMC aids in reducing the error introduced by the numerical SDE solver.

5.3.3 Numerical ODE solver

The associated ordinary differential equation (ODE) for any SDE of the form

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

is given by

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

where both trajectories share the same marginal probability density $p_t(\mathbf{x})$. Consequently, by solving this ODE in reverse time, we can effectively sample from the distribution that solving the reverse-time SDE yields. We refer to this ODE as the *probability flow ODE*.

Here's a schematic figure illustrating the distinction between trajectories produced by the probability flow ODE and SDE trajectories. Despite the differences, both methods sample from the same distribution. Thus, starting from a sample from p_T, we can integrate the ODE in reverse time to acquire a sample from p_0. Specifically, for the SDE in our ongoing example, the following ODE can be integrated from $t = T$ to 0 to generate samples

$$d\mathbf{x} = -\frac{1}{2}\sigma^{2t}s_\theta(\mathbf{x}, t)dt.$$

This integration can be performed using various black-box ODE solvers available in packages like 'scipy'.

5.4 Results

The training procedure has been carried out by using the MNIST dataset only. Unfortunately for training on datasets with more detailed images was required a lot of computational effort we could not afford with Google Colab. On the MNIST dataset instead, we reached good results in the generation process. The loss score was not amazing (around 15 %), however we could generate with different samplers (described above) good quality images. We report some of the results differentiating on samplers that have been used.

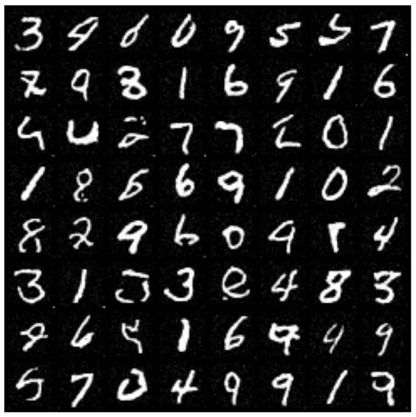

Figure 3: Euler-Maruyama sampler

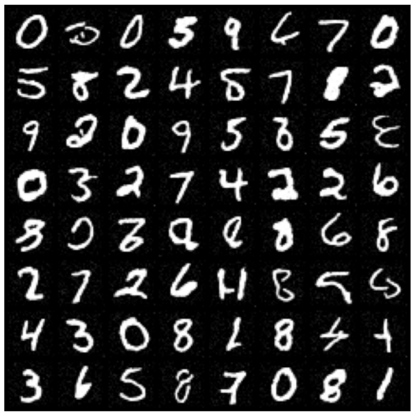

Figure 4: Predictor-Corrector sampler

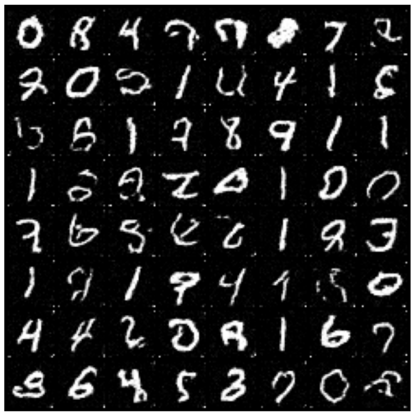

Figure 5: ODE sampler

As we can clearly see from the figures, we plotted a grid (8x8) of images, containing brand new samples. We also tried to train our model on specific classes of the MNIST dataset obtaining less loss accuracy (around 20 %), due to a fewer presence of training images. We present an example of sampling given by the training on the 5 class.

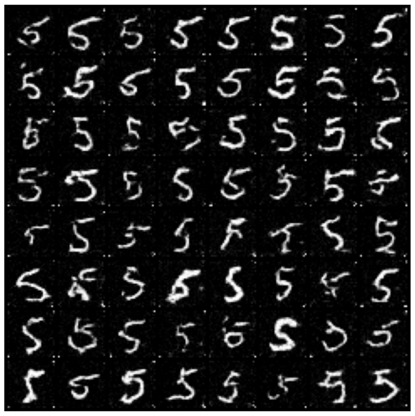

Figure 6: ODE sampler

5.5 Coclusion

As we can see from the previous images, we reached good results in the generation process. However the reader must consider that our model is a toy one, in the sense that it can be improved a lot. The authors of the papers from which we built the theory, have built a neural network and a predictor-corrector sampler that achieve state-of-the-art sample quality on CIFAR-10 (measured in FID and Inception scores), outperforming the best GAN model to date (StyleGAN2 + ADA). They used a better architecture with respect to our model and also they could train it on better machines.