Score-Based Generative Models

Pramook Khungurn

March 16, 2022

In 2021, I read a paper on denoising diffusion models, which proposes a new type of generative models [HJA20]. Now, it turns out that there are parallel works by Yang Song and Stefano Ermon on the so-called "score-based models," which is later discovered to be deeply connected (in other words, pretty much equivalent) to the former approach [SE19].[1]The approaches share the advantanges of being very stable to train and being capable of generating high quality samples, and they also share the disadvantage of being slow when sampling. Researchers have claimed that these models beat GANs in image generation [DN21], and so my interested is piqued.

I read the introductory blog post by Yang Song [Son21], but it seems that I lack the background to understand this body of work. This note aims to fill this understanding gap by summarizing relavant research papers.

1 Preliminary

- We are given n data items $\mathbf{x}_1$, $\mathbf{x}_2$, $\dots$, $\mathbf{x}_N$ that are sampled i.i.d. from a probability distribution $p_{\mathrm{data}}(\mathbf{x})$, which is unknown to us.
- We are interested in modeling $p_{\mathrm{data}}(\mathbf{x})$ by finding a model $p_{\boldsymbol{\theta}}(\mathbf{x})$ with parameters $\boldsymbol{\theta}$ that best approximates it.
  - To reduce levels of subscription, we will sometimes write $p_{\boldsymbol{\theta}}(\mathbf{x})$ as $p(\mathbf{x}; \boldsymbol{\theta})$.
- For the models in this note, we cannot compute the probability $p_{\boldsymbol{\theta}}(\mathbf{x})$ directly.
- However, we would still be able to sample from it, which is something that is of practical use.
- Hence, the focus would be on (1) how to estimate the parameters $\boldsymbol{\theta}$, and (2) how to sample from the model given the parameters.

2 Score Matching [Hyvärinen 2005]

- First, however, we need to take a detour from generative modeling and study a related problem: parameter estimation of unnormalized models.
- For some probabilistic models, the form of the probability distribution is known up to the normalization constant:

$$p(\mathbf{x}) \propto q(\mathbf{x})$$

So, we have that

$$p(\mathbf{x}) = \frac{q(\mathbf{x})}{Z}.$$

[1]FYI, my time in grad school overlapped with that of Stefano's, but we never interacted.

where

$$Z = \int p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

is the normalization constant.

- A common class of such model is the **energy-based model**, where

$$p(\mathbf{x}) \propto e^{-E(\mathbf{x})}.$$

Here, $E(\mathbf{x})$ is called the **energy function**, and the normalization constant

$$Z = \int e^{-E(\mathbf{x})}\,\mathrm{d}\mathbf{x}$$

is called the **partition function**. Energy-based models show up a lot in statistical physics.

- Another class of such models is the **graphical model** where

$$p(\mathbf{x}) \propto \prod_{\mathbf{a}\in\mathcal{F}} p_{\mathbf{a}}(\mathbf{x}).$$

Here, we assume that $\mathbf{x} = (x_1, x_2, \dots, x_n)$, $\mathbf{F}$ is a set of subsets of $\{1, 2, \dots, n)$, and $p_{\mathbf{a}}(\mathbf{x})$ is a function of components of $\mathbf{x}$ whose set of indices are exactly $\mathbf{a}$. You can read more about such models in another note of mine [Khu21].

- We are interested in such probability distributions with paremeters. In other words,

$$p(\mathbf{x};\boldsymbol{\theta}) = \frac{q(\mathbf{x};\boldsymbol{\theta})}{Z(\boldsymbol{\theta})}$$

where

$$Z(\boldsymbol{\theta}) = \int q(\mathbf{x};\boldsymbol{\theta})\,\mathrm{d}\mathbf{x}.$$

- We are particularly interested in estimating $\boldsymbol{\theta}$ from sampled data $\mathbf{x}_1$, $\mathbf{x}_2$, $\dots$, $\mathbf{x}_N$.
- The standard approch would be to perform maximum likelihood estimation (MLE):

$$\arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\mathbf{x}_i;\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \left\{ \sum_{i=1}^{N} \log q(\mathbf{x}_i;\boldsymbol{\theta}) - N \log Z(\boldsymbol{\theta}) \right\}$$

- The general problem is that computing $Z(\boldsymbol{\theta})$ is often infeasible.
- The common way to deal with this is to estimate $Z(\boldsymbol{\theta})$ with Monte Carlo integration.
  - Markov chain Monte Carlo (MCMC) methods are often employed to generate samples that yield low variances.
  - Nevertheless, MCMC methods are slow because they need to generate many samples before convergence.
- In 2005, Aapo Hyvärinen proposed **score matching** as a way to estimate $\boldsymbol{\theta}$ without explicitly dealing with $Z(\boldsymbol{\theta})$ [Hyv05].
- The idea is to "match" the "score function" instead of doing the optimization on the probabilities directly.

- The (Stein) **score function** of a probability distribution $p(\mathbf{x};\boldsymbol{\theta})$ is the gradient with respect to $\mathbf{x}$ of its logarithm:

$$\boldsymbol{\Psi}(\mathbf{x};\boldsymbol{\theta}) = \begin{bmatrix} \Psi_1(\mathbf{x};\boldsymbol{\theta}) \\ \Psi_2(\mathbf{x};\boldsymbol{\theta}) \\ \vdots \\ \Psi_n(\mathbf{x};\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \partial(\log p(\mathbf{x};\boldsymbol{\theta}))/\partial x_1 \\ \partial(\log p(\mathbf{x};\boldsymbol{\theta}))/\partial x_2 \\ \vdots \\ \partial(\log p(\mathbf{x};\boldsymbol{\theta}))/\partial x_n \end{bmatrix} = \nabla_{\mathbf{x}} \log p(\mathbf{x};\boldsymbol{\theta}).$$

  Viewing the distribution p as a function with signature $\mathbb{R}^d \to \mathbb{R}$, we have that $\boldsymbol{\Psi}$ has signature $\mathbb{R}^d \to \mathbb{R}^d$.

- Note that the good thing about the score function is that it allows us to bypass the partition function:

$$\begin{aligned}\nabla_{\mathbf{x}} \log p(\mathbf{x};\boldsymbol{\theta}) &= \nabla_{\mathbf{x}} \log \frac{q(\mathbf{x};\boldsymbol{\theta})}{Z(\boldsymbol{\theta})} = \nabla_{\mathbf{x}}\big(\log q(\mathbf{x};\boldsymbol{\theta}) - \log Z(\boldsymbol{\theta})\big) = \nabla_{\mathbf{x}} \log q(\mathbf{x};\boldsymbol{\theta}) - \nabla_{\mathbf{x}} \log Z(\boldsymbol{\theta}) \\ &= \nabla_{\mathbf{x}} \log q(\mathbf{x};\boldsymbol{\theta}).\end{aligned}$$

  This is because $Z(\boldsymbol{\theta})$ is a constant with respect to $\mathbf{x}$.

- Recall that we are given $\mathbf{x}_1$, $\mathbf{x}_2$, $\dots$, $\mathbf{x}_N$ sampled from an unknown distribution p_{data}. We can define the score function of the data distribution:

$$\boldsymbol{\Psi}_{\mathrm{data}}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\mathrm{data}}(\mathbf{x}).$$

- The "matching" in score matching is trying to find $\boldsymbol{\theta}$ that makes $\boldsymbol{\Psi}(\mathbf{x},\boldsymbol{\theta})$ as close as possible to $\boldsymbol{\Psi}_{\mathrm{data}}(\mathbf{x})$. Operationally, Hyvärinen proposes minimizing the expected squared Euclidean distance between the scores:

$$J(\boldsymbol{\theta}) = \frac{1}{2} E_{\mathbf{x}\sim p_{\mathrm{data}}}\Big[\big\|\boldsymbol{\Psi}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\Psi}_{\mathrm{data}}(\mathbf{x})\big\|^2\Big] = \frac{1}{2}\int p_{\mathrm{data}}(\mathbf{x})\big\|\boldsymbol{\Psi}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\Psi}_{\mathrm{data}}(\mathbf{x})\big\|^2\,\mathrm{d}\mathbf{x}.$$

  The estimator is thus given by by:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}}\ J(\boldsymbol{\theta}).$$

- The function $J(\boldsymbol{\theta})$ is commonly known as the **Fisher divergence** between two distributions. Given two distributions p_0 and p_1, the Fisher divergence is defined as:

$$F(p_0\|p_1) = E_{\mathbf{x}\sim p_0}\Big[\big\|\nabla_{\mathbf{x}} \log p_0(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_1(\mathbf{x})\big\|^2\Big] = \int p_0(\mathbf{x})\big\|\nabla_{\mathbf{x}} \log p_0(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_1(\mathbf{x})\big\|^2\,\mathrm{d}\mathbf{x}.$$

  In other words, we are minimizing $F(p_{\mathrm{data}}(\mathbf{x})\|p(\mathbf{x};\boldsymbol{\theta}))$.

- Recall again that we are only given $\mathbf{x}_1$, $\mathbf{x}_2$, $\dots$, $\mathbf{x}_N$. We don't know what p_{data} is. So, how do we do the optimization then? The good news is that we can rewrite the Fisher divergence in a form that does not involve p_{data} instance the expectation operator.

- **Theorem 1.** *We have that*

$$\begin{aligned} J(\boldsymbol{\theta}) &= \int p_{\mathrm{data}}(\mathbf{x}) \sum_{i=1}^{d}\bigg[\frac{\partial \Psi_i(\mathbf{x};\boldsymbol{\theta})}{\partial x_i} + \frac{1}{2}\big(\Psi_i(\mathbf{x};\boldsymbol{\theta})\big)^2\bigg]\,\mathrm{d}\mathbf{x} + C \\ &= \int p_{\mathrm{data}}(\mathbf{x}) \sum_{i=1}^{d}\bigg[\frac{\partial^2(\log p(\mathbf{x};\boldsymbol{\theta}))}{\partial x_i^2} + \frac{1}{2}\bigg(\frac{\partial(\log p(\mathbf{x};\boldsymbol{\theta}))}{\partial x_i}\bigg)^2\bigg]\,\mathrm{d}\mathbf{x} + C \end{aligned} \tag{1}$$

  where C is a constant that does not depend on $\boldsymbol{\theta}$. The theorem holds under the following conditions:

  - $\mathbf{\Psi}$ *is differentiable.*
  - $p_{\text{data}}(\mathbf{x})$ *is differentiable.*
  - $E_{\mathbf{x}\sim p_{\text{data}}}[\|\mathbf{\Psi}(\mathbf{x};\boldsymbol{\theta})\|^2]$ *and* $E_{\mathbf{x}\sim p_{\text{data}}}[\|\mathbf{\Psi}_{\text{data}}(\mathbf{x})\|^2]$ *are finite for every* $\boldsymbol{\theta}$.
  - $p_{\text{data}}(\mathbf{x})\mathbf{\Psi}(\mathbf{x};\boldsymbol{\theta})$ *has bounded support.*

  The proof can be found in Hyvärinen's paper [Hyv05], and it is based on applying integration by parts.

- Note that the RHS of (1) can be rewritten as:

$$J(\boldsymbol{\theta}) = E_{\mathbf{x}\sim p_{\text{data}}}\left[\nabla_{\mathbf{x}}\cdot\mathbf{\Psi}(\mathbf{x};\boldsymbol{\theta}) + \frac{1}{2}\|\mathbf{\Psi}(\mathbf{x};\boldsymbol{\theta})\|^2\right] = E_{\mathbf{x}\sim p_{\text{data}}}\left[\Delta_{\mathbf{x}}\log p(\mathbf{x};\boldsymbol{\theta}) + \frac{1}{2}\|\nabla_{\mathbf{x}}\log p(\mathbf{x};\boldsymbol{\theta})\|^2\right].$$

  Here, $\nabla_{\mathbf{x}}\cdot$ is the **divergence operator**

$$\nabla_{\mathbf{x}}\cdot\mathbf{f} = \sum_{i=1}^{d}\frac{\partial f_i}{\partial x_i},$$

  and $\Delta_{\mathbf{x}} = \nabla_{\mathbf{x}}\cdot\nabla_{\mathbf{x}}$ is the **Laplace operator**

$$\Delta_{\mathbf{x}} f = \nabla_{\mathbf{x}}\cdot\nabla_{\mathbf{x}} f = \sum_{i=1}^{d}\frac{\partial^2 f}{\partial x_i^2}.$$

- Theorem 1 allows us to approximate $J(\theta)$ by Monte Carlo integrations using the samples we have:

$$J(\boldsymbol{\theta}) \approx \widehat{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{j=1}^{N}\left[\nabla_{\mathbf{x}}\cdot\mathbf{\Psi}(\mathbf{x}_j;\boldsymbol{\theta}) + \frac{1}{2}\|\mathbf{\Psi}(\mathbf{x}_j;\boldsymbol{\theta})\|^2\right]. \tag{2}$$

  This means that we can now optimize for $\boldsymbol{\theta}$.

- Hyvärinen also shows that optimizing for $J(\boldsymbol{\theta})$ would yield the right distribution.

  Theorem 2. *Assume that there is a unique* $\boldsymbol{\theta}^*$ *such that* $p_{\text{data}}(\mathbf{x}) = p(\mathbf{x};\boldsymbol{\theta}^*)$ *and that* $q(\mathbf{x};\boldsymbol{\theta}) > 0$ *for all* $\mathbf{x}$ *and* $\boldsymbol{\theta}$*. Then,* $J(\boldsymbol{\theta}) = 0$ *if and only if* $\boldsymbol{\theta} = \boldsymbol{\theta}^*$.

  Corollary 3. *Under the assumption of Theorem 2, the estimator* $\widehat{J}(\boldsymbol{\theta})$ *is consistent. In other words, it converges in probability towards the true value of* $J(\boldsymbol{\theta})$ *when the sample size approaches infinity.*

3 Denoising Score Matching [Vincent 2011]

- Minimizing the Fisher divergence using the estimate in Equation (2), however, is still not practical. This is because we need to compute

$$\nabla_{\mathbf{x}}\cdot\mathbf{\Psi}(\mathbf{x}_j;\boldsymbol{\theta}) = \sum_{i=1}^{d}\frac{\partial^2(\log p(\mathbf{x}_j;\boldsymbol{\theta}))}{\partial x_i^2},$$

  which requires computing second-order derivatives. While this can certainly be arranged using modern deep learning framework, it would entail computing the Hessian of large neural network $q(\cdot;\boldsymbol{\theta})$, which is not practical.

- In 2011, Pascal Vincent discovered that, by changing the target distribution p_{data} a little, we can rewrite $J(\boldsymbol{\theta})$ into a function that does not involve second-order derivatives [Vin11].

- We change p_{data} by convolving it with an isotropic Gaussian noise. Let us denote this corrupted distribution by p^σ_{data} where σ is the standard deviation of the Gaussian. The sampling process of this distribution is as follows.

  1. Sample $\mathbf{x} \sim p_{\text{data}}$.
  2. Sample $\widetilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \sigma^2 I)$ and return $\widetilde{\mathbf{x}}$ as the output of the sampling process.

  In this way, we have that

$$p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) = \int k(\widetilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

  where

$$k(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{1}{(\sqrt{2\pi}\sigma)^d} \exp\left(\frac{-\|\widetilde{\mathbf{x}} - \mathbf{x}\|^2}{2\sigma^2} \right)$$

  denotes the Gaussian kernel function.

- We note that p^σ_{data} would not be very different from p_{data} if σ is small enough.

- Let us perform score matching on p^σ_{data}. We have that the Fisher divergence is given by

$$\begin{aligned}
J^\sigma(\boldsymbol{\theta}) &= \frac{1}{2} E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\left\| \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}) - \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\|^2 \right] \\
&= \frac{1}{2} E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}) - \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}), \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}) - \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\rangle \right] \\
&= \frac{1}{2} E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2 - 2\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\rangle + \|\boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}})\|^2 \right] \\
&= E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2} \right] - E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\rangle \right] + E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\frac{\|\boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}})\|^2}{2} \right]
\end{aligned}$$

  Because $\|\boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}})\|^2$ is an expression that does not involve (θ), we can treat as a constant that is irrelevent to the optimization process. As a result, we may write

$$J^\sigma(\boldsymbol{\theta}) = E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2} \right] - E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\rangle \right] + C_1$$

  Looking at the middle term, we have that

$$\begin{aligned}
E_{\widetilde{\mathbf{x}} \sim p^\sigma_{\text{data}}} \left[\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\rangle \right] &= \int_{\widetilde{\mathbf{x}}} p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \right\rangle \mathrm{d}\widetilde{\mathbf{x}} \\
&= \int_{\widetilde{\mathbf{x}}} p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}))}{\partial \widetilde{\mathbf{x}}} \right\rangle \mathrm{d}\widetilde{\mathbf{x}} \\
&= \int_{\widetilde{\mathbf{x}}} \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \frac{\partial(\log p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}))}{\partial \widetilde{\mathbf{x}}} \right\rangle \mathrm{d}\widetilde{\mathbf{x}}.
\end{aligned}$$

  Using the fact that

$$f(\mathbf{x}) \frac{\partial(\log f(\mathbf{x}))}{\partial \mathbf{x}} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$$

  for any differentiable function f, we have that

$$\begin{aligned}
p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}) \frac{\partial(\log p^\sigma_{\text{data}}(\widetilde{\mathbf{x}}))}{\partial \widetilde{\mathbf{x}}} = \frac{\partial p^\sigma_{\text{data}}(\widetilde{\mathbf{x}})}{\partial \widetilde{\mathbf{x}}} = \frac{\partial}{\partial \widetilde{\mathbf{x}}} \int_{\mathbf{x}} k(\widetilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x} &= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \frac{\partial k(\widetilde{\mathbf{x}}|\mathbf{x})}{\partial \widetilde{\mathbf{x}}}\, \mathrm{d}\mathbf{x} \\
&= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) k(\widetilde{\mathbf{x}}|\mathbf{x}) \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}}\, \mathrm{d}\mathbf{x}
\end{aligned}$$

Plugging the integral back, we have that

$$\begin{aligned}
E_{\widetilde{\mathbf{x}}\sim p^{\sigma}_{\text{data}}}\Big[\big\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \boldsymbol{\Psi}^{\sigma}_{\text{data}}(\widetilde{\mathbf{x}})\big\rangle\Big] &= \int_{\widetilde{\mathbf{x}}} \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \int p_{\text{data}}(\mathbf{x}) k(\widetilde{\mathbf{x}}|\mathbf{x}) \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}}\,\mathrm{d}\mathbf{x} \right\rangle \mathrm{d}\widetilde{\mathbf{x}} \\
&= \int_{\widetilde{\mathbf{x}}}\int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) k(\widetilde{\mathbf{x}}|\mathbf{x}) \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} \right\rangle \mathrm{d}\mathbf{x}\,\mathrm{d}\widetilde{\mathbf{x}} \\
&= \int_{\mathbf{x}}\int_{\widetilde{\mathbf{x}}} p_{\text{data}}(\mathbf{x}) k(\widetilde{\mathbf{x}}|\mathbf{x}) \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} \right\rangle \mathrm{d}\widetilde{\mathbf{x}}\,\mathrm{d}\mathbf{x} \\
&= E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} \right\rangle\right].
\end{aligned}$$

As a result,

$$J^{\sigma}(\boldsymbol{\theta}) = E_{\widetilde{\mathbf{x}}\sim p^{\sigma}_{\text{data}}}\left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2}\right] - E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} \right\rangle\right] + C_1.$$

Looking at the first term, we have that

$$\begin{aligned}
E_{\widetilde{\mathbf{x}}\sim p^{\sigma}_{\text{data}}}\left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2}\right] &= \int_{\widetilde{\mathbf{x}}} p^{\sigma}_{\text{data}}(\widetilde{\mathbf{x}}) \frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2}\,\mathrm{d}\widetilde{\mathbf{x}} \\
&= \int_{\widetilde{\mathbf{x}}}\left(\int_{\mathbf{x}} k(\widetilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x})\,\mathrm{d}\mathbf{x}\right)\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2}\,\mathrm{d}\widetilde{\mathbf{x}} \\
&= \int_{\mathbf{x}}\int_{\widetilde{\mathbf{x}}} p_{\text{data}}(\mathbf{x}) k(\widetilde{\mathbf{x}}|\mathbf{x}) \frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2}\,\mathrm{d}\widetilde{\mathbf{x}}\,\mathrm{d}\mathbf{x} \\
&= E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2}\right].
\end{aligned}$$

So,

$$J^{\sigma}(\boldsymbol{\theta}) = E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2} - \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} \right\rangle\right] + C_1.$$

Now, let

$$C_2 = E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\frac{1}{2}\left\|\frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}}\right\|^2\right].$$

We have that

$$\begin{aligned}
J^{\sigma}(\boldsymbol{\theta}) &= E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\frac{\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta})\|^2}{2} - \left\langle \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}), \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} \right\rangle + \frac{1}{2}\left\|\frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}}\right\|^2\right] - C_2 + C_1 \\
&= \frac{1}{2}E_{\mathbf{x}\sim p_{\text{data}},\,\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})}\left[\left\|\boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}) - \frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}}\right\|^2\right] - C_2 + C_1.
\end{aligned}$$

Letting $\widetilde{J}^{\sigma}((\theta))$ denote the expectation term on the RHS, the above equation becomes:

$$J^{\sigma}(\boldsymbol{\theta}) = \widetilde{J}^{\sigma}(\boldsymbol{\theta}) - C_2 + C_1.$$

Because C_1 and C_2 are constants with respect to $\boldsymbol{\theta}$, we have that

$$\arg\min_{\boldsymbol{\theta}} J^{\sigma}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \widetilde{J}^{\sigma}(\boldsymbol{\theta}).$$

- The above lengthy derivation tells us that, instead of minimizing $J^\sigma(\boldsymbol{\theta})$, we may minimize $J^\sigma(\boldsymbol{\theta})$ instead. We can see that the new objective is much nicer because

$$\frac{\partial(\log k(\widetilde{\mathbf{x}}|\mathbf{x}))}{\partial \widetilde{\mathbf{x}}} = \frac{\partial}{\partial \widetilde{\mathbf{x}}} \log\left(\frac{1}{(\sqrt{2\pi}\sigma)^2} \exp\left(\frac{-\|\widetilde{\mathbf{x}} - \mathbf{x}\|^2}{2\sigma^2} \right) \right) = \frac{\partial}{\partial \widetilde{\mathbf{x}}} \left(\frac{-\|\widetilde{\mathbf{x}} - \mathbf{x}\|^2}{2\sigma^2} \right) = -\frac{\widetilde{\mathbf{x}} - \mathbf{x}}{\sigma^2}.$$

  So,

$$\widetilde{J}^\sigma(\boldsymbol{\theta}) = \frac{1}{2\sigma^4} E_{\mathbf{x}\sim p_{\text{data}},\, \widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})} \Big[\|\sigma^2 \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}) + (\widetilde{\mathbf{x}} - \mathbf{x})\|^2 \Big].$$

  Dropping the $1/\sigma^4$ factor does not change the optimization problem, so we typically write the loss function as

$$\begin{aligned}
\widetilde{J}^\sigma(\boldsymbol{\theta}) &= \frac{1}{2} E_{\mathbf{x}\sim p_{\text{data}},\, \widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})} \Big[\big\|\sigma^2 \boldsymbol{\Psi}(\widetilde{\mathbf{x}};\boldsymbol{\theta}) + (\widetilde{\mathbf{x}} - \mathbf{x})\big\|^2 \Big] \\
&= \frac{1}{2} E_{\mathbf{x}\sim p_{\text{data}},\, \widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})} \Big[\big\|\sigma^2 \nabla \log q(\widetilde{\mathbf{x}};\boldsymbol{\theta}) + (\widetilde{\mathbf{x}} - \mathbf{x})\big\|^2 \Big] \\
&\approx \frac{1}{2N} \sum_{j=1}^{N} E_{\widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x}_j)} \Big[\big\|\sigma^2 \nabla \log q(\widetilde{\mathbf{x}};\boldsymbol{\theta}) + (\widetilde{\mathbf{x}} - \mathbf{x}_j)\big\|^2 \Big] \\
&\approx \frac{1}{2N} \sum_{j=1}^{N} E_{\boldsymbol{\xi}\sim \mathcal{N}(\mathbf{0},\sigma^2 I)} \Big[\big\|\sigma^2 \nabla \log q(\mathbf{x}_j + \boldsymbol{\xi};\boldsymbol{\theta}) + \boldsymbol{\xi}\big\|^2 \Big],
\end{aligned}$$

  which does not have any second-order derivatives.

- Let us summarize the above discussion into a theorem.

  Theorem 4. *The score matching object can be recasted as a denoising objective. In other words, we have that*

$$\arg\min_{\boldsymbol{\theta}} J^\sigma(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \widetilde{J}^\sigma(\boldsymbol{\theta}).$$

  where

$$\begin{aligned}
J^\sigma(\boldsymbol{\theta}) &= \frac{1}{2} E_{\widetilde{\mathbf{x}}\sim p^\sigma_{\text{data}}} \Big[\big\|\nabla \log q(\mathbf{x}_j + \boldsymbol{\xi};\boldsymbol{\theta}) - \boldsymbol{\Psi}^\sigma_{\text{data}}(\widetilde{\mathbf{x}})\big\|^2 \Big], \\
\widetilde{J}^\sigma(\boldsymbol{\theta}) &= \frac{1}{2} E_{\mathbf{x}\sim p_{\text{data}},\, \widetilde{\mathbf{x}}\sim k(\cdot|\mathbf{x})} \Big[\big\|\sigma^2 \nabla \log q(\mathbf{x}_j + \boldsymbol{\xi};\boldsymbol{\theta}) + (\widetilde{\mathbf{x}} - \mathbf{x})\big\|^2 \Big].
\end{aligned}$$

- What the new objective suggests us to do is as follows.
  - For each data point $\mathbf{x}_j$, we corrupt it by adding a Gaussian noise $\boldsymbol{\xi}$ sampled according to $\mathcal{N}(\mathbf{0}, \sigma^2 I)$ to get $\widetilde{\mathbf{x}} = \mathbf{x}_j + \boldsymbol{\xi}$.
  - We feed the corrupted data $\widetilde{\mathbf{x}}$ to our network, and let it produce the gradient $\nabla \log q(\widetilde{\mathbf{x}};\boldsymbol{\theta})$.
  - We want $\sigma^2 \nabla \log q(\widetilde{\mathbf{x}};\boldsymbol{\theta})$ to be as close as possible to $-\boldsymbol{\xi}$. In other words, if we add it to $\widetilde{x}$, we should get the original signal $\mathbf{x}_j$ back. Phrasing this another way, **we want to denoise $\widetilde{\mathbf{x}}$ back to $\mathbf{x}_j$.**

  As a result, we match the score of p^σ_{data} (and therefore p_{data} if σ is small enough) by training a denoiser!

- Note that the result in this section bears a similarity to the following result in statistics.

  Theorem 5 (Tweedie's formula). *Suppose* $\mathbf{x}$ *and* $\mathbf{y}$ *are random variables that satisfy the distribution* $\mathbf{y} \sim \mathcal{N}(\mathbf{x}, \sigma^2 I)$*. Then,*

$$E[\mathbf{x}|\mathbf{y}] = \mathbf{y} + \sigma^2 \nabla \log p(\mathbf{y})$$

  where $p(\mathbf{y})$ *is the probability density of* $\mathbf{y}$*.*

4 Sampling from a Score-Based Model

- Suppose we have optimized $\boldsymbol{\theta}$ to match the score of p_{data} (or $p^{\sigma}_{\mathrm{data}}$ if we use denoising score matching). How do we sample from data points from $p_{\boldsymbol{\theta}}$ then?
- We actually do not have access to $p_{\boldsymbol{\theta}}$ because we do not know the normalizing constant $Z(\boldsymbol{\theta})$. We only know $q_{\boldsymbol{\theta}}$, which is unnormalized. This forces us to use techniques such as Markov Chain Monte Carlo (MCMC), which can sample from unnormalized distributions.
- There are many variants of MCMC algorithms [Khu22], but we should use those that exploit the score $\nabla \log q_{\boldsymbol{\theta}}(\cdot)$, which is readily availble (because we have just spent a lot of time matching it). Two candidates naturally emerge:
  - **Hamiltonian Monte Carlo** (HMC).
  - **Langevin algorithm**, for which there are two easy variants.
    - **Unadjusted Langevin algorithm** (ULA).
    - **Metropolis-adjusted Langevin algorithm** (MALA).
- ULA is the simplest of the bunch. It requires picking a small constant $\varepsilon > 0$ as a parameter, and it goes as follows:

```
– Start from a random point $\mathbf{x}_0$.
– for $m = 1, 2, 3, \ldots$
    ∗ Sample $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I)$.
    ∗ Compute $\mathbf{x}_m \leftarrow \mathbf{x}_{m-1} + \frac{\varepsilon}{2}\nabla \log q_{\boldsymbol{\theta}}(\mathbf{x}_{m-1}) + \sqrt{\varepsilon}\boldsymbol{\xi}$.
  end for
```

  If our target distribution is not too nasty, then the Markov chain should approach a stationary distribution after being simulated for M iterations for some big value of M. We can then take $\mathbf{x}_M$, $\mathbf{x}_{M+1}$, and so on as the output samples.
- ULA has the following advantages:
  - It only requires the ability to compute the score $\nabla \log q_{\boldsymbol{\theta}}(\mathbf{x}_{m-1})$. Hence, we can make our model take in the input sample $\mathbf{x}$ and output the score directly without having to worry about what the (unnormalized) probability of $\mathbf{x}$ would be.
  - This also greatly simplify training because, in denoising score matching, we do not have to compute any gradients with respect to $\mathbf{x}$. The only gradient we have to compute is that of $\widetilde{J}^{\sigma}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, which would be used in parameter optimization.
- However, ULA has several problems:
  - It may not converge to a stationary distribution.
  - If it does converge, the stationary distribution might not be $p_{\boldsymbol{\theta}}$.

  Nevertheless, according to Song and Ermon, people ignore these problems and use it anyway [SE19].
- MALA and HMC, while guaranteeing the correct stationary distribution, are more complicated.
  - They require the ability to evaluate $q_{\boldsymbol{\theta}}(\mathbf{x})$ when computing the acceptance probability.
  - During sampling, we have to evaluate

$$\nabla \log q_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{\partial(\log q_{\boldsymbol{\theta}}(\mathbf{x}))}{\partial \mathbf{x}}.$$

    - Moreover, after evalutating the gradient above, we have to use it in the calculation of of the loss $\widetilde{J}^\sigma(\boldsymbol{\theta})$, which we must compute the gradient

$$\frac{\partial}{\partial \boldsymbol{\theta}} \widetilde{J}^\sigma(\boldsymbol{\theta})$$

      with respect to $\boldsymbol{\theta}$ for optimization. This can be done with modern deep learning libraries, but the implementation would be slower than that of ULA.

- We have learned how to optimize a score-based model and also learned how to sample from it. This gives us a form of generative models.

5 Noise-Conditional Score-Based Model [Song and Ermon 2019]

- Vanilla score-based models are not practical generative models because of several reasons. The Song–Ermon paper has toy examples the illustrate these points.
    - **The manifold hypothesis.**
        * Real-world datasets have some sort of constraints on their members, and this phenomenon is often thought of as each dataset residing in a low-dimensional manifold inside the ambient space.
        * This means that the score is zero almost everywhere in the ambient space. Vanilla score matching thus cannot work on real-world datasets.
        * Convolving p_{data} with a Gaussian of small variance extends the support to the whole ambient space and allows for denoising score matching.
        * However, we are not in the clear yet.
    - **Low data density regions.**
        * In practice, we only have a finite number of samples of the data. In areas where the p_{data} is low, we might not enough samples to accurately estimate the scores in that area.
        * When two modes of the data distribution are separated by low density regions, Langevin dynamics will not be able to correctly recover the relative weights of these two modes in reasonable time and so might not converge to the true distribution.
- Song and Ermon observed that convolving the target distribution with a Guassian (i.e., working with p^σ_{data} instead of p_{data}) solves both problems simultaneously.
    - The support of any Gaussian distribution is the whole ambient space, so the data no longer resides in a low-dimensional manifold.
    - If σ is high enough of the Gaussian is high enough, it can fill areas where the data distribution is low.
- We see that, as σ becomes larger, the perturbed distribution p^σ_{data} becomes more tractable but different from p_{data}. So, if we have a sequence of standard deviations

$$\sigma_1 > \sigma_2 > \cdots > \sigma_L,$$

  then we have that

$$p^{\sigma_1}_{\mathrm{data}},\ p^{\sigma_1}_{\mathrm{data}},\ \dots,\ p^{\sigma_L}_{\mathrm{data}}$$

  is a sequence of less and less tractable distributions that converge to p_{data}.
- The idea, then, is to do Langevin dynamics on the above sequence of probability distributions rather than on p_{data} or on a single p^σ_{data}.

  - The algorithm would have L phases, and the ith phase would use the score of $p^{\sigma_i}_{\mathrm{data}}$.
  - The hope is that the steps near the beginning would allow us to explore all significant regions of p_{data}, and the steps near the end would home us in on one of the modes.

- The practice of starting with a tractable distribution and successively refine it to a target distribution is not new. It can be found in the method of **simulated annealing** in the context of optimization [KGV83], and **annealed importance sampling** in statistics [Nea98].

5.1 Training and Sampling

- Song and Ermon propose to implement the above idea with the **noise-conditional score networks** (NCSN). This is a single network $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma)$ to estimate the score $\nabla \log p^{\sigma}_{\mathrm{data}}(\mathbf{x})$ of the data distribution after being convolved with an isotropic Gaussian with standard deviation σ.

- For training and sampling, the paper fixes a sequence of standard deviations $\{\sigma_i\}_{i=1}^{L}$. It chooses one such that
$$\frac{\sigma_1}{\sigma_2} = \frac{\sigma_2}{\sigma_3} = \cdots = \frac{\sigma_{L-1}}{\sigma_L} > 1.$$
σ_1 should be learge enough to fill low density areas, and σ_L should be small enough to minimize the effect on data.

- The NCSN is trained with the following objective
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{L}\sum_{i=1}^{L} \lambda(\sigma_i)\ell(\boldsymbol{\theta}, \sigma_i)$$
where $\ell(\cdot, \cdot)$ is the denoising score matching objective
$$\ell(\boldsymbol{\theta}, \sigma) = \frac{1}{2} E_{\mathbf{x} \sim p_{\mathrm{data}}, \widetilde{x} \sim \mathcal{N}(\mathbf{x}, \sigma^2 I)} \bigg[\mathbf{s}_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}, \sigma) + \frac{\widetilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \bigg].$$

- The paper chooses the weight $\lambda(\sigma) = \sigma^2$ in order to make sure that the magnitude of $\lambda(\sigma)\ell(\boldsymbol{\theta}, \sigma)$ does not depend on σ. This is based on the observation that, when $\mathbf{s}_{\boldsymbol{\theta}}$ is trained to optimality, we would have that $\|\mathbf{s}_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}, \sigma)\|$ would be about $1/\sigma$. (See the paper for more details.)

- The sampling algorithm, called **annealed Langevin dynamics**, is as follows.

```
Initialize $\widetilde{\mathbf{x}}_0$.
**for** $i \leftarrow 1$ to $L$ **do**
    $\alpha_i \leftarrow \varepsilon \cdot \sigma_i^2 / \sigma_L^2$
    **for** $t \leftarrow 1$ to $T$ **do**
        Draw $\boldsymbol{\xi}_t \sim \mathcal{N}(\mathbf{0}, I)$.
        $\widetilde{\mathbf{x}}_t \leftarrow \widetilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\xi_t$
    **end for**
    $\widetilde{\mathbf{x}}_0 \leftarrow \widetilde{\mathbf{x}}_T$
**end for**
**return** $\widetilde{\mathbf{x}}_0$.
```

Here ε and T are hyperparameters of the algorithm where ε is the step size, and T is the number of time steps Langevin dynamics is simulated for each standard deviation.

- The paper chooses $\alpha_i \propto \sigma_i^2$ because it wants to make the signal to noise ratio $\frac{\|\alpha_i \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma_i)\|}{2\|\sqrt{\alpha_i}\xi\|}$ constant.

5.2 Image Inpainting

- It is very easy to modify the annealed Langevin dynamics algorithm to do inpainting instead of image generation.
- The modified algorithm takes an image $\mathbf{x}$ and a mask $\mathbf{m}$ of the part of $\mathbf{x}$ that should be preserved.
- The inpainting algorithm is as follows:

```
Initialize $\widetilde{\mathbf{x}}_0$.
for $i \leftarrow 1$ to $L$ do
    $\alpha_i \leftarrow \varepsilon \cdot \sigma_i^2 / \sigma_L^2$
    Draw $\widetilde{\boldsymbol{\xi}} \sim \mathcal{N}(\mathbf{0}, \sigma_i^2 I)$.
    $\mathbf{y} \leftarrow \mathbf{x} + \widetilde{\boldsymbol{\xi}}$.
    for $t \leftarrow 1$ to $T$ do
        Draw $\boldsymbol{\xi}_t \sim \mathcal{N}(\mathbf{0}, I)$.
        $\widetilde{\mathbf{x}}_t \leftarrow \widetilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i} \xi_t$
        $\widetilde{\mathbf{x}}_t \leftarrow \widetilde{\mathbf{x}}_t \otimes (\mathbf{1} - \mathbf{m}) + \mathbf{y} \otimes \mathbf{m}$.
    end for
    $\widetilde{\mathbf{x}}_0 \leftarrow \widetilde{\mathbf{x}}_T$
end for
return $\widetilde{\mathbf{x}}_0$.
```

5.3 Network Architecture

- The network architecture used in the paper relies on three importance components.
  - Conditional instance normalization.
  - Dilated convolutions.
  - U-Net architecture.
- Conditional instance normalization.
  - Let $\mathbf{x}$ denote a feature tensor with C channels, and let $\mathbf{x}_k$ denote its kth channel. Let μ_k and s_k denote the mean and the standard deviation of elements of $\mathbf{x}_k$. In standard instance normalization, the kth channel of the output tensor $\mathbf{z}$ is given by

    $$\mathbf{z}_k = \gamma[k] \frac{\mathbf{x}_k - \mu_k}{s_k} + \beta[k]$$

    where $\boldsymbol{\gamma} = (\gamma[1], \gamma[2], \dots, \gamma[C])$ and $\boldsymbol{\beta} = (\beta[1], \beta[2], \dots, \beta[C])$ are learnable parameters.
  - The paper observed that instance normalization removes all information about μ_k, and so may lead to color shifts when generating images. To alleviate this problem, it tries to introduce μ_k back as follows:

    $$\mathbf{z}_k = \gamma[k] \frac{\mathbf{x}_k - \mu_k}{s_k} + \beta[k] + \alpha[k] \frac{\mu_k - m}{v}$$

    where m is the mean of the μ_k's among the channels, v is the standard deviation of the μ_k's, and $\boldsymbol{\alpha} = (\alpha[1], \dots, \alpha[C])$ is another learnable parameter.

- The "conditional" part of the conditional instance normalization refers to the fact that the paper uses different $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ for each of the σ_i. Hence, the equation becomes:

$$\mathbf{z}_k = \gamma[i,k]\frac{\mathbf{x}_k - \mu_k}{s_k} + \beta[i,k] + \alpha[i,k]\frac{\mu_k - m}{v}.$$

- Dilate convolution.
  - The paper replaces all subsampling layers in its networks with dilated convolution, except the first subsampling layer.
- U-Net architecture.
  - The overall architecture is that of RefineNet [LMSR16], which is a U-Net that incorporates ResNet's design.
  - Customizations:
    - 4-cascaded RefineNet.
    - Pre-activation residual blocks.
    - Replace all batch normalization with conditional instance normalziation.
    - Replace max pooling layer in Refine Blocks with average pooling in order to get smoother images.
    - Add a conditional instance normalization before each convolution and average pooling in the Refine Blocks.
    - Activation function is ELU.
    - Use dilated convolutions instead of subsampling layers in residual blocks, except the first one.
    - Dilation is increased by a factor of 2 when going to the next cascade.

5.4 Experimental and Training Setup

- The paper used NCSNs to generate images from MNIST, CelebA, and CIFAR-10. All images are of size 32×32. Each pixel has value in range $[0, 1]$.
- The paper chooses $L = 10$, $\sigma_1 = 1$, and $\sigma_1 0 = 0.01$.
- For sampling the paper chooses $T = 10$ and $\varepsilon = 2 \times 10^{-5}$. The initial value of $\mathbf{x}_0$ is sampled from a uniform distribution.
- All models were trained with Adama with learning rate of 0.001 for 200000 iterations. The batch size was 128.

6 Improved Training [Song and Ermon 2020]

- Song and Ermon published a followed up paper [SE20] to the one in the last section [SE19]. It does not present a new algorithm but rather introduces various improvements to the model.
- It turns out that there are several problems with the model in [SE19].
  - The hyperparameter settings such as $L = 10$, $\sigma_1 = 1$, and $\sigma_{10} = 0.01$ were all ad hoc.
  - The authors couldn't train models to general images with resolution higher than 32×32.
- To address the above problems, the authors propose several techniques to improve training. These include:

  – A heuristic to choose the noise scale.
  – A simple improvement to the model architecture.
  – A recommendation to sue exponential moving average (EMA) of model parameters while training.

- With all the techniques in the last item, the authors were able to train score-based models to on FFHQ and LSUN datasets to generate images with resolutions as large as 256×256.
- We will now talk about each of the techniques in more details.

6.1 Technique 1: How to choose the initial noise scale

- In [SE19], σ_1 is set to 1 perhaps because the pixel values are in the $[0, 1]$ range.
- However, in real world images, it is typical to have two data points $\mathbf{x}_1$ and $\mathbf{x}_2$ where $\|\mathbf{x}_1 - \mathbf{x}_2\| \gg 1$. Hence, even with the perturbed data distribution $p^{\sigma_1}_{\mathrm{data}}$, it would be hard for Langevin dynamics to reach $\mathbf{x}_2$ when starting from $\mathbf{x}_1$, and this can lead to mode collapse.
- Hence, the paper propose the following technique:

  Technique 1. Choose σ_1 to be as large as the maximum Euclidena distance between all pairs of training data points.

6.2 Technique 2: How to choose other noise scales

- After choosing σ_1, we next have to choose $\sigma_2, \dots, \sigma_L$.
- In [SE19], the paper choose σ_L to be a low value (0.01). The other noise scales are chosen so that the ratio σ_{i-1}/σ_i is a constant.
- What we really want from the noise scales, however, is that the probability density $p^{\sigma_{i-1}}_{\mathrm{data}}$ and $p^{\sigma_i}_{\mathrm{data}}$ should be similar enough that we can transition from the former to the latter with few problems.
- The authors performed a simplified analysis where $p^{\sigma}_{\mathrm{data}}$ is assumed to be $\mathcal{N}(\mathbf{0}, \sigma^2 I)$. They showed that, if we let $r = \|\mathbf{x}\|$, then it follows that, when the data is D-dimensional,

$$p^{\sigma}_{\mathrm{data}}(r) = \frac{1}{2^{D/2-1}\Gamma(D/2)} \frac{r^{D-1}}{\sigma^D} \exp\left(\frac{-r^2}{2\sigma^2}\right)$$

  where $\Gamma(\cdot)$ denotes the gamma function.

- Moreover, they showed that

$$r - \sqrt{D}\sigma \to \mathcal{N}(0, \sigma^2/2)$$

  as $D \to \infty$. Hence, for high resolution images, it is OK to assume that

$$p^{\sigma}_{\mathrm{data}}(r) \approx \mathcal{N}(\sqrt{D}\sigma, \sigma^2/2).$$

- To simplify the notation, let m_i denots $\sqrt{D}\sigma_i$ and s_i denotes $\sqrt{\sigma_i^2/2}$. We have that

$$p^{\sigma_i}_{\mathrm{data}}(r) \approx \mathcal{N}(m_i, s_i^2).$$

- Now, we make sure that $p^{\sigma_{i-1}}_{\mathrm{data}}$ and $p^{\sigma_i}_{\mathrm{data}}$ are similar. So, we want to make sure that they overlap as much as possible. In other words, $p^{\sigma_i}_{\mathrm{data}}$ should have high mass in areas where $p^{\sigma_{i-1}}_{\mathrm{data}}$ has high mass.

- $p_{\text{data}}^{\sigma_{i-1}}(r)$ has high mass in the interval $\mathcal{I}_{i-1} = [m_{i-1} - 3s_{i-1}, m_{i-1} + 3s_{i-1}]$. The mass of $p_{\text{data}}^{\sigma_i}$ on the above interval is given by

$$p_{\text{data}}^{\sigma_i}(r \in \mathcal{I}_{i-1}) = \operatorname{erf}(\sqrt{2D}(\gamma_i - 1) + 3\gamma_i) - \operatorname{erf}(\sqrt{2D}(\gamma_i - 1) - 3\gamma_i)$$

  where $\gamma_i = \sigma_{i-1}/\sigma_i$. We want the above quantity to be reasonably large.

- **Technique 2**. Chooose $\sigma_2, \dots, \sigma_L$ to be a geometric progression with common ratio γ such that

$$\operatorname{erf}(\sqrt{2D}(\gamma - 1) + 3\gamma) - \operatorname{erf}(\sqrt{2D}(\gamma - 1) - 3\gamma) \approx 0.5.$$

- I think how to choose σ_L has not change: we fix it to 0.01 or something lower.

6.3 Technique 3: How to condition with the noise scale

- In [SE19], the noise scale is used to calculate the biases and scales of the instance normalization units.
- In [SE20], they significantly simplify the architecture based on the observation that $\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{s}, \sigma)\| \approx 1/\sigma$.
- **Technique 3**. Parameterize the NCSN with $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma) = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})/\sigma$ where $\mathbf{s}_{\boldsymbol{\theta}}$ is an unconditonal network.
- The authors found that this new architecture achived similar training losses compared to the architecture employed in [SE19].

6.4 Technique 4: How to configure Langevin dynamics

- In [SE19], for each noise scale, we run Langevin dynamics for T timesteps with step size $\alpha = \varepsilon \cdot \sigma_i^2/\sigma_L^2$.
- The [SE19] paper uses $\varepsilon = 2 \times 10^{-5}$ and $T = 100$.
- The authors performed another simplified analysis to understand the behavior of annealed Langevin dynamics.
  - They assumed that $p_{\text{data}}^{\sigma_i} = \mathcal{N}(\mathbf{0}, \sigma_i^2 I)$.
  - They would like to understand the behavior of running Langevin dynamics for T iterations under a single noise level σ_i. In this setting, we start with $\mathbf{x}_0 \sim p_{\text{data}}^{\sigma_{i-1}} = \mathcal{N}(\mathbf{0}, \sigma_{i-1}^2 I)$. Then, for T times, we then run the update step

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \alpha \nabla_{\mathbf{x}} \log p_{\text{data}}^{\sigma_i}(\mathbf{x}_t) + \sqrt{2\alpha}\boldsymbol{\xi} \tag{3}$$

    where $\xi \sim \mathcal{N}(\mathbf{0}, I)$.[2]
  - It follows that $\mathbf{x}_T \sim N(\mathbf{0}, s_T^2 I)$ where

$$\frac{s_T^2}{\sigma_i^2} = \left(1 - \frac{\varepsilon}{\sigma_L^2}\right)^{2T} \left(\gamma^2 - \frac{2\varepsilon}{\sigma_L^2 - \sigma_L^2(1 - \varepsilon/\sigma_L^2)^2}\right) + \frac{2\varepsilon}{\sigma_L^2 - \sigma_L^2(1 - \varepsilon/\sigma_L^2)^2}$$

    where $\gamma = \sigma_{i-1}/\sigma_i$.

- The ideal behavior should be that $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \sigma_i^2 I)$. Hence, we should strive to make $s_T^2/\sigma_i^2 = 1$.
- **Technique 4**. Chooose T as large as allowed by a computing budget and then select ε that makes s_T^2/σ_i^2 as close as possible to 1.

[2]Note that (3) is slightly different from $\widetilde{\mathbf{x}}_t \leftarrow \widetilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\boldsymbol{\xi}_t$, which was used in the annealed Langevin dynamics algorithm in Section 5.1. However, one can see that they are essentially the same as we simply replaced α_i with 2α.

6.5 Technique 5: How to improve stability

- The authors observed that the images generated by the model of [SE19] have unstable visual quality: the FID score fluctuated wildly during training. Moreover, the generated images show color shifts which varies wildly based on the number of training iterations.
- To improve the stability of the generated images, the authors recommended using exponential moving average (EMA) of the network weights when generating images at test time.
- **Technique 5**. Apply exponential moving average to network parameters when sampling.

7 Generalization to Stochastic Differental Equations [Song et al. 2021]

- In this new paper, the authors called the approach in the [SE19] paper **score matching with Langevin dynamics** (SMLD).
- The **denoising diffusion probability model** (DDPM) [SWMG15, HJA20] is another class of generative models whose generative process involves inverting corruption (in our case, denoising) done to a data item.
  - I wrote a note on DDPMs at [Khu20].
- By denoising, the DDPM implicitly computes the score, and so both DDPM and SMLD can be collectively called **score-gased generative models**.
- In [SSDK+21], the authors propose a framework that unify SMLD and DDPM through the lense of stochastic differential equations. This enables new sampling methods and extensions to the existing models.
- The framework.
  - There is a continuous-time stochastic process called the **forward process** that corrupts the data distrbution p_{data} into a distribution that has a well-known form (most of the time, a multi-variate Gaussian distribution).
  - The forward process is described by a stochastic differential equation (SDE) that has it as a solution.
  - Given a forward process, there is a **reverse process** that runs the process backwards in time. It is the solution of a reverse-time SDE [And82].
  - Generation can then be casted as simulating the reverse-time SDE, starting from the well-known distribution. This is, of course, done by a time-dependent neural network.
- The above framework yielded the following benefits.
  - The framework allows employment of any general-purpose SDE solver to do sampling. The paper proposes two algorithms.
    * The **Predictor–Corrector** (PC) samplers which has a Metropolis–Hastings-like correction step. Examples include the Langevin algorithm and HMC.
    * The **deterministic** samplers that are based on the probabiliy flow ordinary differential equation (ODE). This approach allows:
      · Fast sampling via black-box ODE solvers.
      · Flexible dasta manipulation via latent codes.
      · Exact likelihood computation.

    * The generation process can be manipulated by conditioning on information not available during training. This allows:
        · class-conditional generation,
        · image inpainting,
        · colorization, and
        · solving other inverse problems.

- The paper also give a better architecture for SMLD that achieved new state-of-the-art Inception scores and FID scores.

References

[And82] Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[DN21] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[Hyv05] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Khu20] Pramook Khungurn. Generative diffusion models. `https://pkhungurn.github.io/notes/notes/ml/generative-diffusion-models/index.html`, 2020. Accessed: 2022-03-15.

[Khu21] Pramook Khungurn. Introduction to conditional random fields. `https://pkhungurn.github.io/notes/notes/ml/conditional-random-fields-intro/index.html`, 2021. Accessed: 2022-02-07.

[Khu22] Pramook Khungurn. A primer on markov chain monte carlo algorithms. `https://pkhungurn.github.io/notes/notes/scicomp/mcmc-primer/mcmc-primer.pdf`, 2022. Accessed: 2022-02-08.

[LMSR16] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. *CoRR*, abs/1611.06612, 2016.

[Nea98] Radford M. Neal. Annealed importance sampling, 1998.

[SE19] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *CoRR*, abs/1907.05600, 2019.

[SE20] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *CoRR*, abs/2006.09011, 2020.

[Son21] Yang Song. Generative modeling by estimating gradients of the data distribution. `https://yang-song.github.io/blog/2021/score/`, 2021. Accessed: 2022-02-07.

[SSDK$^+$21] 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.

[SWMG15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015.

[Vin11] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.