Deep Generative Models

Chapter 6: Generation by Diffusion Process

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

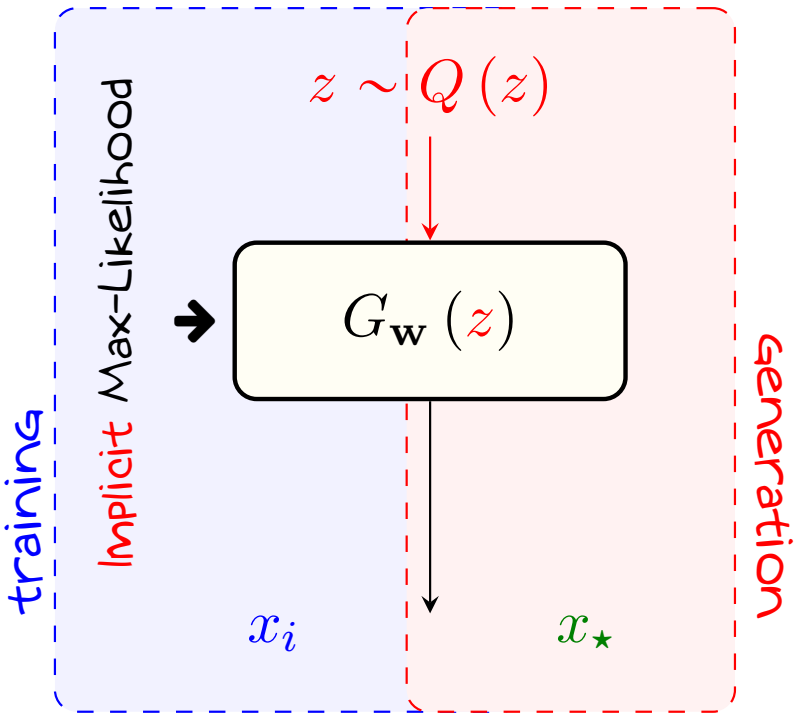
University of Toronto

Summer 2025

Modern Data Generation

Recall that we are still working with modern generative models

modern generative models mainly learn how to sample!



Learning via *Diffusion Process*

Our main goal in this chapter is to

build models that learn by mimicking a diffusion process

+ *What is a diffusion process?!*
- We understand it after a bit of preliminaries

Langevin Equation

Langevin equation describes a diffusion process in which particle moves under friction and random collisions: the location of the particle in time t satisfies

$$\mathrm{d}x\left(t\right)=\nabla_{x}\log P\left(x\right)\mathrm{d}t+\sqrt{2}\mathrm{d}B\left(t\right)$$

where $B\left(t\right)$ is the Brownian motion process

Brownian Motion

+ *What is Brownian motion process?*
- It's a stochastic process that evolves i.i.d. Gaussian

Brownian Motion

The Brownian motion process starts at zero, i.e., $B(0) = 0$ *and independently progresses through time by Gaussian distribution*

↳ *The variance increases linearly with time*

$$B(t_2) - B(t_1) \sim \mathcal{N}(0, t_2 - t_1)$$

↳ *Any difference* $B(t_2) - B(t_1)$ *is independent of past*

Langevin Equation: *Learning Viewpoint*

+ *What is important about this equation?*

$$\mathrm{d}x\left(t\right) = \nabla_x \log P\left(x\right) \mathrm{d}t + \sqrt{2}\mathrm{d}B\left(t\right)$$

– It describes the physical diffusion

Diffusion Process by Langevin Equation

As time evolves, the location of the particle at time t, i.e., $x\left(t\right)$, converges to a random variable whose distribution is $P\left(x\right)$

+ *But how can we use such a fact?!*
– Well! Let's be a bit naive and *approximate* solution of this equation

Langevin Dynamics: *Markov Chain*

Say we move tiny steps in time: *we shift step by step from* $t_i = t$ *to* $t_{i+1} = t + \delta$ *and approximate* $\mathrm{d}t = \delta$ *as well as*

$$\mathrm{d}x(t) \approx x(t+\delta) - x(t) \equiv x^{(i+1)} - x^{(i)}$$
$$\mathrm{d}B(t) \approx B(t+\delta) - B(t) \equiv \Delta B^{(i)} \sim \mathcal{N}(0, \delta)$$

Now, we could say

$$x^{(i+1)} - x^{(i)} = \nabla_x \log P\left(x^{(i)}\right)\delta + \sqrt{2}\Delta B^{(i)}$$

We can say $\Delta B^{(i)} = \sqrt{\delta}\varepsilon^{(i)}$ *with* $\varepsilon^{(i)} \sim \mathcal{N}(0,1)$ *and write*

$$x^{(i+1)} = x^{(i)} + \delta \nabla_x \log P\left(x^{(i)}\right) + \sqrt{2\delta}\varepsilon^{(i)}$$

The Markov process $x_1 \to x_2 \to \cdots$ *is what we call Langevin dynamics*

Key Property of Langevin Dynamics

+ *Well! Then what?!*
– We can use this process to sample $P(x)$

Sampling via Langevin Dynamics

Recall that in Langevin equation $x(t)$ converges in distribution to $P(x)$

1. *Start by an arbitrary point $x^{(0)}$*
2. *Evolve through time by*

$$x^{(i+1)} = x^{(i)} + \delta \nabla_x \log P\left(x^{(i)}\right) + \sqrt{2\delta}\varepsilon^{(i)}$$

3. *Get a sample of $P(x)$*

Recall: *Sampling EBMs*

+ *Wait a moment! This sounds familiar!*
– Yep! We used it to sample EBMs

`Langevin_Sampling():`

1: *Initiate sample* $x^{(0)}$ *and choose a converging series* $\{\epsilon_t\}$
2: **for** $t = 1, \ldots, T$ **do**
3: Sample $\eta_t \sim \mathcal{N}(0, 1)$
4: Update $x^{(t)} \leftarrow x^{(t-1)} + \frac{\epsilon_t}{2} \nabla_x \log P_{\mathbf{w}}\left(x^{(t-1)}\right) + \sqrt{\epsilon_t}\eta_t$
5: **end for**
6: **return** $x^{(T)}$ *for* T *larger than burn-in period*

Alternative Objective for Generation

This brings up this idea:

> *Why don't we model and learn directly the term $\nabla_x \log P(x)$?!*

If we learn $\nabla_x \log P(x)$ $\rightsquigarrow$ we can sample $P(x)$ by mimicking diffusion

---

+ *Why should this be better than learning $P(x)$ itself?!*
- *Well! There are two key reasons*
  1. *As we have seen in Chapter 4, this can be computationally easier*

$$\nabla_x \log P(x) = \nabla_x \log \frac{\exp\{-\mathcal{E}(x)\}}{\mathcal{Z}} = -\nabla_x \mathcal{E}(x)$$

  2. *It opens a way to learn the distribution implicitly*
     ↳ *We will see this in this chapter*

Score Function

Score Function

The score function is defined as the gradient of the log-likelihood w.r.t. argument

$$s(x) = \nabla_x \log P(x)$$

! Attention: *Score Function vs Informant*

The score function is different from the informant which is also called score sometimes in statistics: if we have a model $P_{\mathbf{w}}(x)$; then, the informant is

$$\iota_{\mathbf{w}}(x) = \nabla_{\mathbf{w}} \log P_{\mathbf{w}}(x)$$

We just keep calling this informant to avoid confusion

Score Function: *Gaussian Example*

Example: *Say we have a Gaussian distribution, i.e.,*

$$P(x) = \frac{\exp\left\{-x^2/2\right\}}{\sqrt{2\pi}}$$

The score function in this case is

$$s(x) = \frac{\mathrm{d}}{\mathrm{d}x} \log P(x) = -\frac{1}{2}\frac{\mathrm{d}x^2}{\mathrm{d}x} = -x$$

We note that in this example

- *The score function is a random variable*
- *The mean and variance are*

$$\mathbb{E}_{x\sim P}\left\{s(x)\right\} = 0 \qquad \mathbb{E}_{x\sim P}\left\{s^2(x)\right\} = 1$$

Properties of Score Function: *Normalization Independence*

Normalization Independence

The score function does not change by any distribution scaling

+ *How can such property help?*
– When we know the distribution up to a constant

Say we know joint distribution $P(x, y)$ and look for score function of $P(x|y)$

- *Computing $P(x|y)$ itself needs marginalization over x*
- *The score function is though given by*

$$\nabla_x \log P(x|y) = \nabla_x \log \frac{P(x, y)}{P(y)} = \nabla_x \log P(x, y)$$

which does not need any marginalization

Properties of Score Function: *Zero-Mean*

Score Function is Zero-Mean

For well-defined distributions, the expected score function is zero, i.e.,

$$\mathbb{E}_{x\sim P}\left\{s\left(x\right)\right\}=0$$

Let's expand the expression

$$\mathbb{E}_{x\sim P}\left\{s\left(x\right)\right\}=\int s\left(x\right)P\left(x\right)\mathrm{d}x=\int\nabla_{x}\log P\left(x\right)P\left(x\right)\mathrm{d}x$$
$$=\int\frac{\nabla_{x}P\left(x\right)}{P\left(x\right)}P\left(x\right)\mathrm{d}x=\int\nabla_{x}P\left(x\right)\mathrm{d}x$$

Say we want to compute entry i

$$\int\frac{\partial}{\partial x_{i}}P\left(x\right)\mathrm{d}x=P\left(x_{i}\right)\Big|_{-\infty}^{+\infty}=0\leftarrow\text{if }P\text{ is decaying}$$

Properties of Score Function

Score Function is Zero-Mean

For well-defined distributions, the covariance matrix is the negative expected Jacobian of the score function, i.e.,

$$\operatorname{Cov}\left[s\left(x\right)\right] = -\mathbb{E}_{x\sim P}\left\{\nabla_x s\left(x\right)\right\}$$

We can again show this by definition

$$\begin{aligned}\operatorname{Cov}\left[s\left(x\right)\right] &= \mathbb{E}_{x\sim P}\left\{\left(s\left(x\right) - \mathbb{E}_{x\sim P}\left\{s\left(x\right)\right\}\right)\left(s\left(x\right) - \mathbb{E}_{x\sim P}\left\{s\left(x\right)\right\}\right)^{\mathsf{T}}\right\} \\ &= \mathbb{E}_{x\sim P}\left\{s\left(x\right)s\left(x\right)^{\mathsf{T}}\right\} = \cdots = -\mathbb{E}_{x\sim P}\left\{\nabla_x s\left(x\right)\right\}\end{aligned}$$

Score Models

Back to our problem: *our idea is to learn the score function of data*

Score-based Model

Score-based model $s_{\mathbf{w}}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^d$ *is a computational model approximating the score funciton* $s(x) = \nabla_x \log P(x)$ *of the data from its sample*

The key point about the score-based models is that

- *They do not need to fit into a specific definition*
  - ↳ *Models for distributions should add up to* 1
- *They could be directly used for sampling*
  - ↳ *We just need to replace them in Langevin dynamics*

Training Score Models

+ *Say we choose a NN as score model, how can we train it?!*
– We try to do score matching!

Score Matching

We train the model, such that its difference to the true score is minimized. This means that we train the model for the risk function

$$R(\mathbf{w}) = \mathbb{E}_{x\sim P}\left\{\|s_{\mathbf{w}}(x) - s(x)\|^2\right\}$$

+ *Excellent! We have neither the true score nor data distribution!*
– So was it the case when *we wanted to minimize KL divergence*
  ↳ *We ended up with maximum likelihood!*

Training Score: *Score Risk*

Let's try to look at this risk function: *given the definition we have*

$$R(\mathbf{w}) = \mathbb{E}_{x\sim P}\left\{\|s_{\mathbf{w}}(x) - s(x)\|^2\right\}$$
$$= \mathbb{E}_{x\sim P}\left\{\|s_{\mathbf{w}}(x)\|^2\right\} + \mathbb{E}_{x\sim P}\left\{\|s(x)\|^2\right\} - 2\mathbb{E}_{x\sim P}\left\{s_{\mathbf{w}}(x)^{\mathsf{T}} s(x)\right\}$$

Since we want to minimize w.r.t. model parameters $\mathbf{w}$*, we can say*

$$\underset{\mathbf{w}}{\operatorname{argmin}} R(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\mathbb{E}_{x\sim P}\left\{\|s_{\mathbf{w}}(x)\|^2\right\} - 2\mathbb{E}_{x\sim P}\left\{s_{\mathbf{w}}(x)^{\mathsf{T}} s(x)\right\}\right]$$

Looking at the above term, we could say

- ✔ *The term* $\mathbb{E}_{x\sim P}\left\{\|s_{\mathbf{w}}(x)\|^2\right\}$ *can be estimated by sampling*
- ✖ *The term* $\mathbb{E}_{x\sim P}\left\{s_{\mathbf{w}}(x)^{\mathsf{T}} s(x)\right\}$ *still needs data distribution*

Training Score: *Score Divergence*

Let's try to further expand the latter term

$$\begin{aligned}\mathbb{E}_{x\sim P}\left\{s_{\mathbf{w}}(x)^{\mathsf{T}} s(x)\right\} &= \int s_{\mathbf{w}}(x)^{\mathsf{T}} s(x) P(x)\,\mathrm{d}x \\ &= \int s_{\mathbf{w}}(x)^{\mathsf{T}} \underbrace{\nabla_x \log P(x)\, P(x)}_{\nabla_x P(x)}\,\mathrm{d}x \\ &= \int s_{\mathbf{w}}(x)^{\mathsf{T}} \nabla_x P(x)\mathrm{d}x\end{aligned}$$

Using the definition of the gradient, we can write

$$\begin{aligned}\int s_{\mathbf{w}}(x)^{\mathsf{T}} \nabla_x P(x)\mathrm{d}x &= \int \sum_i s_{\mathbf{w},i}(x) \frac{\partial}{\partial x_i} P(x)\mathrm{d}x \\ &= \sum_i \int s_{\mathbf{w},i}(x) \frac{\partial}{\partial x_i} P(x)\mathrm{d}x\end{aligned}$$

Recall: *Integration by Part*

We want to compute

$$\int_a^b f(x) \frac{\mathrm{d}}{\mathrm{d}x} g(x) \, \mathrm{d}x$$

But, we would like to work with g itself instead of its derivative

We use the fact that

$$\frac{\mathrm{d}}{\mathrm{d}x} [f(x) g(x)] = \frac{\mathrm{d}}{\mathrm{d}x} f(x) \, g(x) + f(x) \frac{\mathrm{d}}{\mathrm{d}x} g(x)$$

and write the original integral as

$$\int_a^b f(x) \frac{\mathrm{d}}{\mathrm{d}x} g(x) \, \mathrm{d}x = f(x) g(x) \Big|_a^b - \int_a^b \frac{\mathrm{d}}{\mathrm{d}x} f(x) g(x) \, \mathrm{d}x$$

Training Score: *Score Divergence*

Using integration by part, we could say

$$\int s_{\mathbf{w},i}(x)\frac{\partial}{\partial x_i}P(x)\mathrm{d}x = s_{\mathbf{w},i}(x)\,P(x)\Big|_{-\infty}^{\infty} - \int \frac{\partial}{\partial x_i}s_{\mathbf{w},i}(x)P(x)\mathrm{d}x$$
$$= 0 - \mathbb{E}_{x\sim P}\left\{\frac{\partial}{\partial x_i}s_{\mathbf{w},i}(x)\right\}$$

This concludes that the cross-product term is given by

$$\mathbb{E}_{x\sim P}\left\{s_{\mathbf{w}}(x)^{\mathsf{T}}\,s(x)\right\} = \sum_i \int s_{\mathbf{w},i}(x)\frac{\partial}{\partial x_i}P(x)\mathrm{d}x$$
$$= -\sum_i \mathbb{E}_{x\sim P}\left\{\frac{\partial}{\partial x_i}s_{\mathbf{w},i}(x)\right\}$$
$$= -\mathbb{E}_{x\sim P}\left\{\sum_i \frac{\partial}{\partial x_i}s_{\mathbf{w},i}(x)\right\}$$

Training Score: *Score Divergence*

Score Divergence

The divergence of the score-model $s_{\mathbf{w}}(\cdot)$ is defined as

$$\operatorname{div}\left(s_{\mathbf{w}}(x)\right) = \sum_i \frac{\partial}{\partial x_i} s_{\mathbf{w},i}(x) = \operatorname{tr}\{\nabla_x s_{\mathbf{w}}(x)\}$$

Using this definition, we could say

$$\mathbb{E}_{x\sim P}\left\{s_{\mathbf{w}}(x)^{\mathsf{T}}\, s(x)\right\} = -\mathbb{E}_{x\sim P}\left\{\operatorname{div}\left(s_{\mathbf{w}}(x)\right)\right\}$$

which can be estimated from data samples!

Training Score: *Hyvärinen Result*

Aapo Hyvärinen was the one who showed this result

Score Risk

The score matching is equivalent to minimizing

$$R_{\text{score}}(\mathbf{w}) = \mathbb{E}_{x\sim P}\left\{\|s_{\mathbf{w}}(x)\|^2\right\} - 2\mathbb{E}_{x\sim P}\left\{\text{div}\left(s_{\mathbf{w}}(x)\right)\right\}$$

This means that the risk for score matching can be estimated as

$$\hat{R}_{\text{score}}(\mathbf{w}) = \hat{\mathbb{E}}_{x\sim P}\left\{\|s_{\mathbf{w}}(x)\|^2\right\} - 2\hat{\mathbb{E}}_{x\sim P}\left\{\text{div}\left(s_{\mathbf{w}}(x)\right)\right\}$$

using the data samples

Training Computational Score Models

`Score_Train(`$\mathbb{D}$`:dataset):`

1: *Initiate the score model* $s_{\mathbf{w}}$ *with some* $\mathbf{w}$
2: **for** *multiple epochs* **do**
3: *Sample a batch of data samples* $\left\{x^j : j = 1, \ldots, n\right\}$ *from* $\mathbb{D}$
4: **for** *sample* $j = 1, \ldots, n$ **do**
5: *Compute the model Jacobian* $\mathbf{J}^j \leftarrow \nabla_x s_{\mathbf{w}}\left(x^j\right)$
6: *Compute sample risk as* $R^j \leftarrow \left\|s_{\mathbf{w}}\left(x^j\right)\right\|^2 + \operatorname{tr}\{\mathbf{J}^j\}$
7: *Backpropagate to compute sample gradient* ∇R^j
8: **end for**
9: *Update* $\mathbf{w}$ *using* `Opt_avg` $\left\{\nabla_{\mathbf{w}} \hat{R}^j\right\}$
10: **end for**
11: **return** *trained score model*

This is though computationally expensive

↳ *We need to backpropagate on* $\nabla_x s_{\mathbf{w}}\left(x\right)$

Sampling Computational Score Models

`Score_Sampling():`

1: *Initiate sample* $x^{(0)}$ *and choose a converging series* $\{\epsilon_t\}$
2: **for** $t = 1, \ldots, T$ **do**
3: Sample $\eta_t \sim \mathcal{N}(0, 1)$
4: Update $x^{(t)} \leftarrow x^{(t-1)} + \frac{\epsilon_t}{2} s_{\mathbf{w}}\left(x^{(t-1)}\right) + \sqrt{\epsilon_t}\eta_t$
5: **end for**
6: **return** $x^{(T)}$ *for* T *larger than burn-in period*