Notes on CS236: Deep Generative Models

Hengyi Wang

April 2023

1. Introduction

This note is for my study of *CS236: Deep Generative Models*, which mainly focuses on generative models that view the world through probability. In such a worldview, we can think of any kind of observed data, say $\mathcal{D}$, as a finite set of samples from an underlying distribution, say p_{data}. The goal of any generative model is then to approximate the data distribution p_{data} given access to the dataset $\mathcal{D}$. The hope is that if we are able to *learn* a good generative model, we can use the learned model for downstream *inference*.

Learning

We will be primarily interested in parametric approximations to the data distribution, which summarizes all the information about the dataset $\mathcal{D}$ in a finite set of parameters. In the parametric setting, we can think of the task of learning a generative model as picking the parameters within a family of model distributions that minimizes some notion of distance between the model distribution $p_{\boldsymbol{\theta}}$ and the data distribution p_{data}.

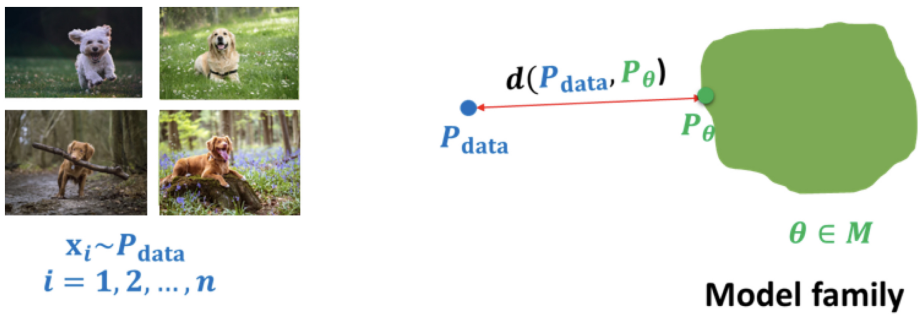


Figure 1: Illustration of learning a model distribution $p_{\boldsymbol{\theta}}$ from data distribution p_{data}.

We can specify our goal of learning the parameters of a generative model $\boldsymbol{\theta}$ within a model family $\mathcal{M}$ such that the model distribution $p_{\boldsymbol{\theta}}$ is close to the data distribution p_{data} as:

$$\min_{\theta \in \mathcal{M}} d(p_{\text{data}}, p_\theta), \tag{1}$$

where $d(\cdot)$ is a notion of distance between probability distributions. To address the optimization problem in Eq. 1, we need to answer following questions:

- What is the representation for the model family $\mathcal{M}$?
- What is the objective function $d(\cdot)$?
- What is the optimization procedure for minimizing $d(\cdot)$?

Inference

For a discriminative model such as logistic regression, the fundamental inference task is to predict a label for any given datapoint. Generative models, on the other hand, learn a joint distribution over the entire data. There are three fundamental inference queries for evaluating a generative model:

- *Density estimation*: Given a datapoint $\mathbf{x}$, what is the probability assigned by the model, i.e., $p_{\boldsymbol{\theta}}(\mathbf{x})$?
- *Sampling*: How can we generate novel data from the model distribution, i.e., $\mathbf{x}_{\text{new}} \sim p_{\boldsymbol{\theta}}(\mathbf{x})$?
- *Unsupervised representation learning*: How can we learn meaningful feature representations for a datapoint $\mathbf{x}$?

Considering an example of learning a generative model over dog images, we can intuitively expect a good generative model to work as follows: For density estimation, we expect $p_{\boldsymbol{\theta}}(\mathbf{x})$ to be high for dog images and low otherwise. Alluding to the name generative model, sampling involves generating novel images of dogs beyond the ones we observe in our dataset. Finally, representation learning can help discover high-level structure in the data such as the breed of dogs.

2. Autoregressive Models

Considering a joint disctribution $p(\mathbf{x})$, we can factorize it over the n-dimensions by the chain rule of probability as

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i | x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i | \mathbf{x}_{<i}), \tag{2}$$

where $\mathbf{x}_{<i} = [x_1, x_2, \ldots, x_{i-1}]$ denotes the vector of random variables with index less than i. For simplicity, we assume the datapoints are binary, i.e., $\mathbf{x} \in \{0, 1\}^n$. The chain rule factorization can be expressed graphically as a Bayesian network.

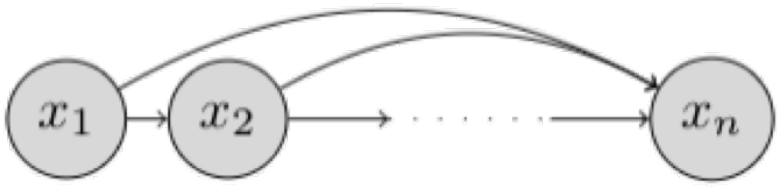
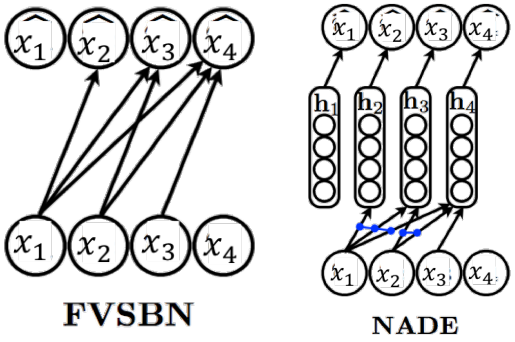


Figure 2: Graphical model for an autoregressive Bayesian network with no conditional independence assumptions.

Such a Bayesian network that makes no conditional independence assumptions is said to obey the *autoregressive* property. The term *autoregressive* originates from the literature on time-series models where observations from the previous time steps are used to predict the value at the current time step.

One problem is that if we allow for every conditional $p(x_i | \mathbf{x}_{<i})$ to be specified in a tabular form, then such a representation is fully general and can represent any possible distribution over n random variables. However, the space complexity for such a representation grows exponentially with n.

In an *autoregressive generative model*, the conditionals are specified as parameterized functions with a fixed number of parameters. That is, we assume the conditional distributions $p(x_i | \mathbf{x}_{<i})$ to



a) FVSBN b) NADE

Figure 3: Illustration of a) fully visible sigmoid belief network over four variables, and b) A neural autoregressive density estimator over four variables. The blue connections denote the tied weights $W[., i]$ used for computing the hidden layer activations.

correspond to a Bernoulli random variable and learn a function that maps the preceding random variables $x_1, x_2, \ldots, x_{i1}$ to the mean of this distribution. Hence, we have

$$p_{\theta_i}(x_i|\mathbf{x}_{<i}) = \text{Bern}(f_i(x_1, x_2, \ldots, x_{i-1})) \tag{3}$$

where θ_i denotes the set of parameters used to specify the mean function $f_i : \{0, 1\}^{i-1} \to [0, 1]$. The number of parameters of an autoregressive generative model is given by $\sum_{i=1}^{n} |\theta_i|$.

In the simplest case, we can specify the function as a linear combination of the input elements followed by a sigmoid non-linearity (to restrict the output to lie between 0 and 1). This gives us the formulation of a *Fully-visible sigmoid belief network* (FVSBN).

$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma(\alpha_0^{(i)} + \alpha_1^{(i)} x_1 + \ldots + \alpha_{i-1}^{(i)} x_{i-1}), \tag{4}$$

where σ denotes the sigmoid function and $\theta_i = \{\alpha_0^{(i)}, \alpha_1^{(i)}, \ldots, \alpha_{i-1}^{(i)}\}$ denote the parameters of the mean function. The conditional for variable i requires i parameters, and hence the total number of parameters in the model is given by $\sum_{i=1}^{n} i = O(n^2)$.

A natural way to increase the expressiveness of an autoregressive generative model is to use more flexible parameterizations for the mean function e.g., multi-layer perceptrons (MLP). For example, consider the case of a neural network with 1 hidden layer. The mean function for variable i can be expressed as

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<\mathbf{i}} + \mathbf{c}_i) \tag{5}$$

$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma(\boldsymbol{\alpha}^{(i)} \mathbf{h}_i + b_i), \tag{6}$$

where $\mathbf{h}_i \in \mathbb{R}^d$ denotes the hidden layer activations for the MLP and $\theta_i = \{A_i \in \mathbb{R}^{d \times (i-1)}, \mathbf{c}_i \in \mathbb{R}^d, \boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d, b_i \in \mathbb{R}\}$ are the set of parameters for the mean function $\mu_i(\cdot)$. The total number of parameters in this model is dominated by the matrices A_i and given by $O(n^2 d)$.

The *Neural Autoregressive Density Estimator* (NADE) provides an alternate MLP-based parameterization. Parameters are shared across the functions used for evaluating the conditionals:

$$\mathbf{h}_i = \sigma(W_{\cdot,<i} \mathbf{x}_{<\mathbf{i}} + \mathbf{c}) \tag{7}$$

$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma(\boldsymbol{\alpha}^{(i)} \mathbf{h}_i + b_i), \tag{8}$$

where $\theta = \{W \in \mathbb{R}^{d\times n}, \mathbf{c} \in \mathbb{R}^d, \{\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d\}_{i=1}^n, \{b_i \in \mathbb{R}\}_{i=1}^n\}\theta = \{W \in \mathbb{R}^{d\times n}, \mathbf{c} \in \mathbb{R}^d, \{\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d\}_{i=1}^n, \{b_i \in \mathbb{R}\}_{i=1}^n\}$ is the full set of parameters for the mean functions $f_1(\cdot), f_2(\cdot), \ldots, f_n(\cdot)$. The weight matrix W and the bias vector c are shared across the conditionals. Therefore, the total number of parameters gets reduced from $O(n^2 d)$ to $O(nd)$.

Learning and inference

Recall that learning a generative model involves optimizing the closeness between the data and model distributions. One common measurement of distance between two distributions is KL divergence:

$$\min_{\theta\in\mathcal{M}} d_{KL}(p_{\text{data}}, p_\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \left[\log p_{\text{data}}(\mathbf{x}) - \log p_\theta(\mathbf{x})\right]. \tag{9}$$

Since p_{data} does not depend on θ, we can equivalently recover the optimal parameters via maximizing likelihood estimation:

$$\max_{\theta\in\mathcal{M}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \left[\log p_\theta(\mathbf{x})\right]. \tag{10}$$

Here $\log p_\theta(\mathbf{x})$ is referred to as the log-likelihood of the datapoint x with respect to the model distribution $p_{\boldsymbol{\theta}}$. To approximate the expectation over the unknown p_{data}, we make an assumption: points in the dataset $\mathcal{D}$ are sampled independently and identically distributed (IID) from p_{data}. This allows us to obtain an unbiased Monte Carlo estimate of the objective as

$$\max_{\theta\in\mathcal{M}} \frac{1}{|D|} \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}) = \mathcal{L}(\theta|\mathcal{D}). \tag{11}$$

The maximum likelihood estimation (MLE) objective has an intuitive interpretation: pick the model parameters $\theta \in \mathcal{M}$ that maximize the log-probability of the observed datapoints in $\mathcal{D}$.

In practice, we optimize the MLE objective using mini-batch gradient ascent. At every iteration t, we sample a mini-batch $\mathcal{B}_t$ of datapoints sampled randomly from the dataset and update the parameters based on the computed gradient:

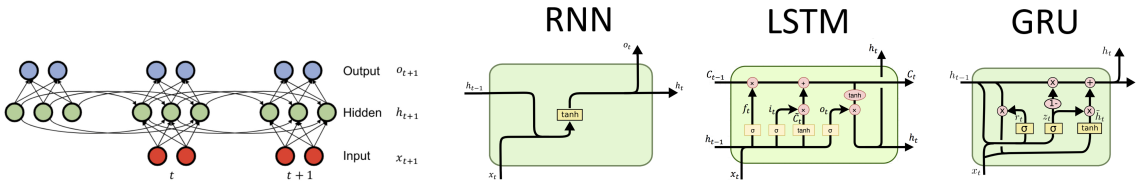
$$\theta^{(t+1)} = \theta^{(t)} + r_t \nabla_\theta \mathcal{L}(\theta^{(t)}|\mathcal{B}_t), \tag{12}$$

where r_t is the learning rate at iteration t. Now that we have a well-defined objective and optimization procedure, the only remaining task is to evaluate the objective in the context of an autoregressive generative model. To this end, we substitute the factorized joint distribution of an autoregressive model in the MLE objective to get

$$\max_{\theta\in\mathcal{M}} \frac{1}{|D|} \sum_{\mathbf{x}\in\mathcal{D}} \sum_{i=1}^{n} \log p_{\theta_i}(x_i|\mathbf{x}_{<i}), \tag{13}$$

where $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ now denotes the collective set of parameters for the conditionals.

Inference in an autoregressive model is straightforward. For density estimation of an arbitrary point $\mathbf{x}$, we simply evaluate the log-conditionals $\log p_{\theta_i}(x_i|\mathbf{x}_{<i})$ for each i and add these up to obtain the log-likelihood assigned by the model to $\mathbf{x}$. Sampling from an autoregressive model is a simple sequential procedure. Finally, an autoregressive model does not directly learn unsupervised representations of the data.



a) Details of RNN b) Comparison of RNN, LSTM, and GRU

Figure 4: Illustration of a) Details of RNN architecture and b) Comparison of vanilla RNN, LSTM, and GRU.

Autoregressive model family

Recurrent Neural Nets (RNN). RNN aims at addressing the challenge of modeling $p(x_t|x_{1:t-1};\alpha^t)$ as "History" $x_{1:t-1}$ keeps getting longer. Thus, RNN proposes to keep a summary and recursively update it as follows:

$$\begin{aligned} \text{Summary update rule: } h_{t+1} &= \tanh\left(W_{hh}h_t + W_{xh}x_{t+1}\right) \\ \text{Prediction: } o_{t+1} &= W_{hy}h_{t+1} \\ \text{Summary initalization: } h_0 &= \boldsymbol{b}_0 \end{aligned} \tag{14}$$

Here h_t is a summary of inputs seen till time t, output layer specifies parameters for conditional $p(x_t|x_{1:t-1})$. One problem of vanilla RNN is that through backpropagation, the gradient might get smaller/larger through time and lead to a vanishing/exploding gradient problem at the earlier time step. To overcome this problem two variants of RNN: Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) are proposed.

Long Short Term Memory (LSTM). LSTM aims to address the problem of the vanishing/exploding gradient in RNN. The cell state and three gates (input gate, forget gate, output gate) are proposed. The feed-forward process can be written as:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ h_t &= o_t \odot \tanh(C_t), \end{aligned} \tag{15}$$

where we can find out that LSTM uses f_t as the weight for cell states from the previous time step i.e. C_{t-1}. This is referred to as forget gate. i_t is then used as the weight for merging information into the cell state C_t. This is referred to as the input gate. Finally, the output gate would give the final output based on the cell state C_t and o_t.

Gated Recurrent Unit (GRU). In comparison to LSTM, GRU is much more simple as it does not use cell states. Instead, GRU only uses hidden states with two gates (Update gate, and reset gate) to control the hidden states. The feed-forward process can be written as:

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \\ \tilde{h}_t &= \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_{t-1}, \end{aligned} \tag{16}$$

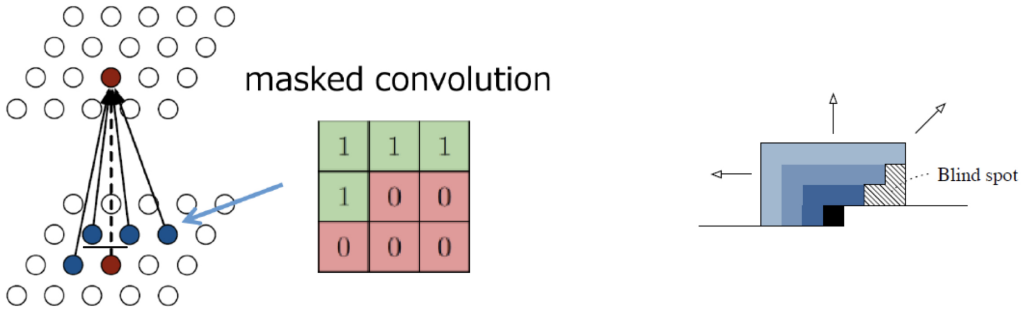


Figure 5: Illustration of the masked convolution and the corresponding blind spot in PixelCNN.

where we can find out that the reset gate uses r_t as the weight for hidden states from the previous time step h_{t-1}, and the update gate uses z_t to update the hidden state.

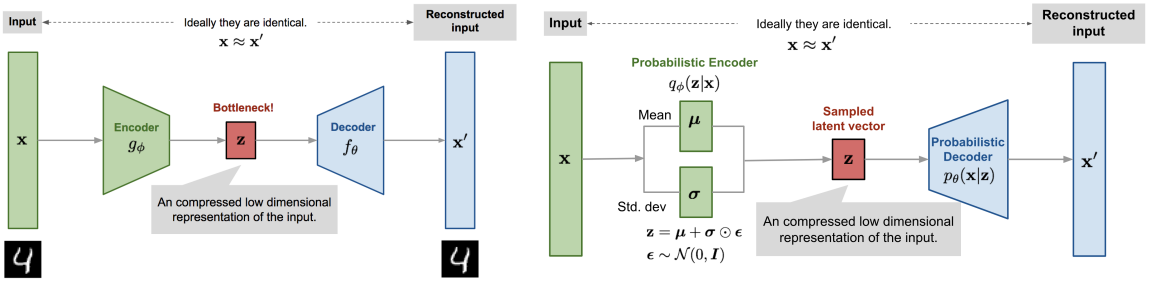
PixelCNN. The idea of Pixel CNN is to predict the next pixel given the context (a neighborhood of pixels) based on convolutional architecture. To make it an autoregressive model, the convolution kernel needs to be masked as in Fig. 5. Note as the image usually contain three color channels, we can formulate the conditional probability in the sequential order of red, green, and blue:

$$p(x_i|\mathbf{x}_{<i}\}) = p(x_i^{red}|\mathbf{x}_{<i}\})p(x_i^{green}|\mathbf{x}_{<i}, x_i^{red}\})p(x_i^{blue}|\mathbf{x}_{<i}, x_i^{red}, x_i^{green}\}) \tag{17}$$

One problem of the PixelCNN is that masked convolution will cause the blind spot as in Fig. 5. GatedPixelCNN solves such an issue by stacking the horizontal and vertical masked convolution.

3. Variational Autoencoder

Autoencoder (AE) aims to learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process so as to discover a more efficient and compressed representation. The model contains an encoder $g(\cdot)$ parameterized by ϕ and a decoder $f(\cdot)$ parameterized by θ. The low-dimensional code learned for input $\mathbf{x}$ in the bottleneck layer is $\mathbf{z} = g_\phi(\mathbf{x})$ and the reconstructed input is $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$.



a) Autoencoder b) Variational Autoencoder

Figure 6: Comparison of a) Autoencoder b) variational Autoencoder.

The parameters (θ, ϕ) are learned together to output a reconstructed data sample the same as the original input, $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$, or in other words, to learn an identity function. The training objective can be a simple MSE loss:

$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2. \tag{18}$$

The idea of the Variational Autoencoder (VAE) is deeply rooted in the methods of the variational Bayesian and graphical models. Instead of mapping the input into a fixed vector, we want to map it into a distribution. Let's label this distribution as p_θ, parameterized by θ. The relationship between the data input $\mathbf{x}$ and the latent encoding vector $\mathbf{z}$ can be fully defined by:

- Prior $p_\theta(\mathbf{z})$
- Likelihood $p_\theta(\mathbf{x}|\mathbf{z})$
- Posterior $p_\theta(\mathbf{z}|\mathbf{x})$

Assuming that we know the optimal parameter $\theta^\star$, which maximize the probability of generating real data samples (we use the log probability here):

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}), \tag{19}$$

where

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}. \tag{20}$$

It is not easy to compute $p_\theta(\mathbf{x}^{(i)}$ as it is very expensive to check all the possible values of $\mathbf{z}$. Thus, we need a new approximation function to output what is a likely code given an input $\mathbf{x}$, $q_\phi(\mathbf{z}|\mathbf{x})$ parameterized by ϕ.

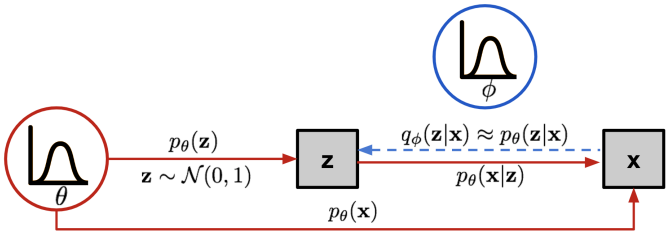


Figure 7: Illustration of Variational Autoencoder.

Fig. 7 illustrates the pipeline of the variational autoencoder. The conditional probability $p_\theta(\mathbf{x}|\mathbf{z})$ defines a generative model, which is also known as a probabilistic decoder. The approximation function $q_\phi(\mathbf{z}|\mathbf{x})$ is the probabilistic encoder.

Loss Function: Evidence Lower Bound (ELBO)

The estimated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ should be very close to $p_\theta(\mathbf{x}|\mathbf{z})$. We can use Kullback-Leibler (KL) divergence to quantify the distance between these two distributions. In our case we want to minimize $D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})|p_\theta(\mathbf{z}|\mathbf{x}))$ with respect to ϕ. We can expand the equation:

$$
\begin{aligned}
&D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} && \text{; Because } p(z|x)=p(z,x)/p(x) \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \big(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} \big) d\mathbf{z} \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} && \text{; Because } \textstyle\int q(z|x)dz=1 \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} && \text{; Because } p(z,x)=p(x|z)p(z) \\
&= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} [\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \\
&= \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})
\end{aligned}
$$

Thus, we will have

$$
D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}). \tag{21}
$$

Once rearrange the left and RHS of the equation:

$$
\log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})). \tag{22}
$$

We can find that the LHS of the Eq. 22 is exactly what we want to maximize: 1) we want to maximize the log-likelihood of generating real data $\log p_\theta(\mathbf{x})$ and also minimize the difference between the real and estimated posterior distributions $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$. Note that $p_\theta(\mathbf{x})$ is fixed with respect to q_ϕ. The negation of the Eq. 22 defines our loss function:

$$
\begin{aligned}
L_{\mathrm{VAE}}(\theta,\phi) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) && (23) \\
&= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) && (24)
\end{aligned}
$$

In Variational Bayesian methods, this loss function is known as the variational lower bound, or evidence lower bound. The "lower bound" part in the name comes from the fact that KL divergence is always non-negative and thus $-L_{\mathrm{VAE}}$ is the lower bound of $\log p_\theta(\mathbf{x})$:

$$
-L_{\mathrm{VAE}} = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x}). \tag{25}
$$

Therefore by minimizing the loss, we are maximizing the lower bound of the probability of generating real data samples.

Reparameterization Trick

The expectation term in Eq. 24 invokes generating samples from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$, such process cannot backpropagate the gradient. Thus, to make it trainable, we need the reparameterization trick: express the random variable $\mathbf{z}$ as a deterministic variable $\mathbf{z} = \mathcal{T}_\phi(\mathbf{x}, \boldsymbol{\epsilon})$, where $\boldsymbol{\epsilon}$ is an auxiliary independent random variable, and the transformation function $\mathcal{T}_\phi$ parameterized by ϕ converts $\boldsymbol{\epsilon}$ to $\mathbf{z}$. For example, a common choice of the form of $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian with a diagonal covariance structure:

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \quad \text{; Reparameterization trick.}$$

where $\odot$ refers to element-wise product. The reparameterization trick works for other types of distributions as well. In the multivariate Gaussian case, we make the model trainable by learning the mean μ and variance σ of the distribution explicitly using the reparameterization trick, while the stochasticity remains in the random variable $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$.

4. Normalizing Flow Models

Normalizing flow is another type of likelihood-based generative model. Let us recall the previous two types of likelihood-based generative models:

- Autoregressive Models: $p_\theta(\mathbf{x}) = \prod_{i=1}^{N} p_\theta(x_i|\mathbf{x}_{<i})$
- Variational Autoencoders: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z})\mathrm{d}\mathbf{z}$

Autoregressive models provide tractable likelihoods but no direct mechanism for learning features, whereas variational autoencoders can learn feature representations but have intractable marginal likelihoods. Normalizing flows, instead, combines the best of both worlds, allowing both feature learning and tractable marginal likelihood estimation.

Change of Variables Formula

In normalizing flows, we wish to map simple distributions (easy to sample and evaluate densities) to complex ones (learned via data). The change of variables formula describes how to evaluate the densities of a random variable that is a deterministic transformation from another variable. Given two random variables Z and X, the mapping between Z and X, given by $f : \mathbb{R}^n \to \mathbb{R}^n$, is invertible such that $X = f(Z)$ and $Z = f^{-1}(X)$. Then,

$$p_X(\mathbf{x}) = p_Z(f^{-1}(\mathbf{x})) \left| \det \left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|, \tag{26}$$

where $\mathbf{x}$ and $\mathbf{z}$ need to be continuous and have the same dimension. Since for any invertible matrix A, $\det(A^{-1}) = \det(A)^{-1}$, we have

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1} \tag{27}$$

Formulation of Normalizing Flow

Let us consider a directed, latent-variable model over observed variables X and latent variables Z. In a normalizing flow model, the mapping between Z and X, given by $f_\theta : \mathbb{R}^n \to \mathbb{R}^n$, is deterministic and invertible such that $X = f_\theta(Z)$ and $Z = f_\theta^{-1}(X)$. Using change of variables, the marginal likelihood $p(\mathbf{x})$ is given by

$$p_X(\mathbf{x}; \theta) = p_Z(f_\theta^{-1}(\mathbf{x})) \left| \det \left(\frac{\partial f_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|. \tag{28}$$

Different from autoregressive models and variational autoencoders, deep normalizing flow models require specific architectural structures:

- The input and output dimensions must be the same
- The transformation must be invertible
- Computing the determinant of the Jacobian needs to be efficient (and differentiable)

Normalizing Flow Model Family

The Planar Flow introduces the following invertible transformation

$$\mathbf{x} = f_\theta(z) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)\mathbf{x} = f_\theta(z) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b) \tag{29}$$

parametrized by $\theta = (\mathbf{w}, \mathbf{u}, b)$, where $h(\cdot)$ is a non-linearity. The absolute value of the determinant of the Jacobian is given by

$$\left|\det\left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\right)\right| = \left|1 + h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{u}^\top \mathbf{w}\right|. \tag{30}$$

However, here $\mathbf{u}, \mathbf{w}, b, h(\cdot)$ needed to be restricted in order to be invertible. Note that while $f_\theta(\mathbf{z})$ is invertible, computing $f_\theta^{-1}(\mathbf{z})$ could be difficult analytically. The following models address this problem, where both f_θ and f_θ^{-1} have simple analytical forms.

The Nonlinear Independent Components Estimation (NICE) model and Real Non-Volume Preserving (RealNVP) model composes two kinds of invertible transformations: additive coupling layers and rescaling layers. The coupling layer in NICE partitions a variable $\mathbf{z}$ into two disjoints subsets say $\mathbf{z}_1$ and $\mathbf{z}_2$. Then it applies the following transformation:
Forward mapping $\mathbf{z} \to \mathbf{x}$:

1. $\mathbf{x}_1 = \mathbf{z}_1$, which is an identity mapping.
2. $\mathbf{x}_2 = \mathbf{z}_2 + m_\theta(\mathbf{z_1})$, where m_θ is a neural network.

Inverse mapping $\mathbf{x} \to \mathbf{z}$:

1. $\mathbf{z}_1 = \mathbf{x}_1$, which is an identity mapping.
2. $\mathbf{z}_2 = \mathbf{z}_2 - m_\theta(\mathbf{x_1})$, which is the inverse of the forward transformation.

Therefore, the Jacobian of forward mapping is

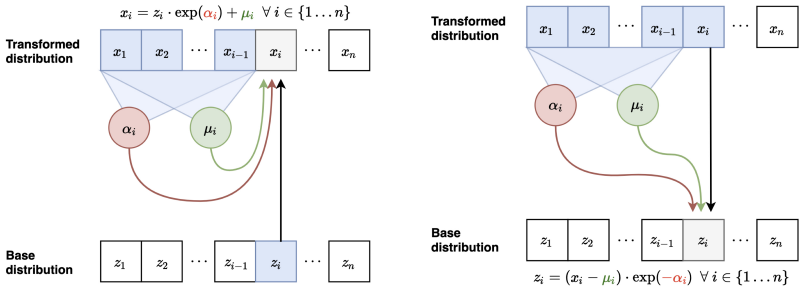
$$\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial f_1}{z_1} & \frac{\partial f_1}{z_2} \\ \frac{\partial f_2}{z_1} & \frac{\partial f_2}{z_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m_\theta^{'}(\mathbf{z_1} & 1 \end{bmatrix}, \tag{31}$$

which is a lower triangular matrix and its determinant is simply the product of the elements on the diagonal, which is 1. Therefore, this defines a volume-preserving transformation. RealNVP adds scaling factors to the transformation:

$$\mathbf{x}_2 = \exp(s_\theta(\mathbf{z}_1)) \odot \mathbf{z}_2 + m_\theta(\mathbf{z_1}), \tag{32}$$

where $\odot$ denotes the element-wise product. This results in a non-volume preserving transformation.

Some autoregressive models can also be interpreted as flow models. Let us consider a Gaussian autoregressive model



a) Forward mapping b) Backward mapping

Figure 8: Illustration of a) forward mapping and b) backward mapping of a MAF model.

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i}) \tag{33}$$

such that $p(x_i|\mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \dots, x_{i-1}), \exp(\alpha_i(x_1, \dots, x_{i-1}))^2)$. Here $\mu_i(\cdot)$ and $\alpha_i(\cdot)$ are neural networks for $i > 1$ and constant for $i = 1$. The sampling of this model is:

- Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \dots, n$
- Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$
- Let $x_2 = \exp(\alpha_1)z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
- Let $x_3 = \exp(\alpha_3)z_3 + \mu_3$ ...

We can consider this Gaussian autoregressive model as a flow model by transforming samples from standard Gaussian $(z_1, z_2, \dots, z_n)$ to those generated from the model $(x_1, x_2, \dots, x_n)$ via invertible transformations (parameterized by $\mu_i(\cdot), \alpha_i(\cdot)$).

Masked Autoregressive Flow (MAF) uses this interpretation. As shown in Fig. 8, the forward and backward mapping of MAF are:
Forward mapping $\mathbf{z} \to \mathbf{x}$:

1. Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$
2. Let $x_2 = \exp(\alpha_1)z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
3. Let $x_3 = \exp(\alpha_3)z_3 + \mu_3$ ...

Inverse mapping $\mathbf{x} \to \mathbf{z}$:

1. Compute all μ_i, α_i given $\mathbf{x}$
2. Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$ (scale and shift)
3. Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$
4. Let $z_3 = (x_3 - \mu_3)/\exp(\alpha_3)$ ...

One problem of MAF is that the computational complexity of sampling is $\mathcal{O}(n)$, where n is the dimension of the samples. To address the sampling problem, the Inverse Autoregressive Flow (IAF) simply inverts the generating process of MAF to achieve fast sampling. However, computing the likelihood of new data points would be slow.

Parallel WaveNet combines the best of both worlds for IAF and MAF where it uses an IAF student model to retrieve sample and a MAF teacher model to compute likelihood. The teacher model can be efficiently trained via maximum likelihood, and the student model is trained by minimizing the KL divergence between itself and the teacher model. Since computing the IAF likelihood for an IAF sample is efficient, this process is efficient.

5. Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN) is a likelihood-free learning method that considers objectives that do not depend directly on a likelihood function. Considering two-sample tests: given a data distribution $S_1 = \mathcal{D} = \{\mathbf{x} \sim p_{\text{data}}\}$ and a model distribution $S_2 = \{\mathbf{x} \sim p_\theta\}$, we aim to train the generative model to minimize a two-sample test objective between S_1 and S_2. However, finding a two-sample test objective in high dimensions is hard. Thus, the key idea of GAN is to learn a statistic that maximizes a suitable notion of distance between the two sets of samples S_1 and S_2, i.e. learn any functions (e.g. neural networks) that try to distinguish "real" samples from the data and the "fake" samples generated from the model. The training objective for the discriminator D is:

$$\max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [\, \log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_G} [\, \log(1 - D(\mathbf{x}))] \,. \tag{34}$$

For a fixed generator G, the discriminator is performing the binary classification with the cross entropy objective: assign probability 1 to true data points $\mathbf{x} \sim p_{\text{data}}$, and probability 0 to fake samples $\mathbf{x} \sim p_G$. The optimal discriminator should be:

$$D_G^\star(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \in [0, 1] \,. \tag{35}$$

Once the generator is trained to its optimal, p_G gets very close to p_{data}. When $p_G = p_{\text{data}}$, $D_G^\star(\mathbf{x})$ becomes $1/2$.

In addition to the discriminator, GAN also has a generator that plays a two-player minimax game between a generator and a discriminator. The generator is a directed latent variable model with a deterministic mapping between $\mathbf{z}$ and $\mathbf{x}$ given G_θ, that minimizes a two-sample test objective. The training objective for the generator G is:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [\, \log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_G} [\, \log(1 - D(\mathbf{x}))] \,. \tag{36}$$

For the optimal discriminator $D_G^\star(\cdot)$, we have

$$\begin{aligned}
&V\left(G, D_G^*(\mathbf{x})\right) \\
&= E_{\mathbf{x}\sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}\,(\mathbf{x})}{p_{\text{data}}\,(\mathbf{x}) + p_G(\mathbf{x})}\right] + E_{\mathbf{x}\sim p_G} \left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}\,(\mathbf{x}) + p_G(\mathbf{x})}\right] \\
&= E_{\mathbf{x}\sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}\,(\mathbf{x})}{\frac{p_{\text{data}}\,(\mathbf{x}) + p_G(x)}{2}}\right] + E_{\mathbf{x}\sim p_G} \left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}\,(\mathbf{x}) + p_G(x)}{2}}\right] - \log 4 \\
&= \underbrace{D_{KL}\left[p_{\text{data}}\,, \frac{p_{\text{data}}\,+ p_G}{2}\right] + D_{KL}\left[p_G, \frac{p_{\text{data}}\,+ p_G}{2}\right]}_{2\times \text{ Jenson-Shannon Divergence (JSD)}} - \log 4 \\
&= 2D_{JS}\left[p_{\text{data}}\,, p_G\right] - \log 4,
\end{aligned} \tag{37}$$

where

$$D_{JS}\left[p,q\right]=\frac{1}{2}\left(D_{KL}\left[p,\frac{p+q}{2}\right]+D_{KL}\left[q,\frac{p+q}{2}\right]\right) \tag{38}$$

is also called the symmetric KL divergence. For the optimal discriminator $D^{\star}_{G^{\star}}(\cdot)$ and the generator $G^{\star}(\cdot)$, we have

$$V(G^{\star},D^{\star}_{G^{\star}}(\mathbf{x}))=-\log 4. \tag{39}$$

The GAN training algorithm is as follows:

Algorithm 1 GAN training algorithm

Training objective:

$$\min_{\theta}\max_{\phi}V(G_{\theta},D_{\phi})=\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log D_{\phi}(\mathbf{x})\right]+\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1-D_{\phi}(G_{\theta}(\mathbf{z})))\right].$$

1. Sample minibatch of m training points $\mathbf{x}^{(1)},\mathbf{x}^{(2)},\ldots,\mathbf{x}^{(m)}$ from $\mathcal{D}$
2. Sample minibatch of m training points $\mathbf{z}^{(1)},\mathbf{z}^{(2)},\ldots,\mathbf{z}^{(m)}$ from p_z
3. Update the discriminator parameter ϕ by stochastic gradient ascent:

$$\nabla_{\phi}V(G_{\theta},D_{\phi})=\frac{1}{m}\nabla_{\phi}\sum_{i=1}^{m}\left[\log D_{\phi}(\mathbf{x}^{(i)})+\log(1-D_{\phi}(G_{\theta}(\mathbf{z}^{(i)})))\right].$$

3. Update the generator parameter θ by stochastic gradient ascent:

$$\nabla_{\theta}V(G_{\theta},D_{\phi})=\frac{1}{m}\nabla_{\theta}\sum_{i=1}^{m}\log(1-D_{\phi}(G_{\theta}(\mathbf{z}^{(i)}))).$$

4. Repeat for a fixed number of epochs

Problems of GAN

Mode collapse. During training, the generator may collapse to a setting where it always produces same outputs, this is commonly referred as *mode collapse*, i.e. the model fails to learn to represent the real-world distribution and stucks in a small space with extremely low variety.
Vanishing gradient. The supports of p_{data} and p_G lie on low dimensional manifolds. They are almost certainly gonna to be disjoint. In that case, we are always capable of finding a perfect discriminator that separates real and fake samples 100% correctly. In that case, we will have $D(\mathbf{x})=1,\forall\mathbf{x}\in p_{\text{data}}$ and $D(\mathbf{x})=0,\forall\mathbf{x}\in p_G$ and the gradient for generator falls to zero. Thus, the training of a GAN faces an dilemma: a perfect discriminator results in zero gradient while a bad discriminator provides inaccurate gradient. This makes the training of GAN become very instable.

Wasserstein GAN (WGAN)

Given two densities p and q, the f-divergence is given by

$$D_f(p,q)=\mathbb{E}_{\mathbf{x}\sim q}\left[f(\frac{p(\mathbf{x})}{q(\mathbf{x})})\right]. \tag{40}$$

For f-divergence, the support of q has to cover the support of p. Otherwise discontinuity arises in f-divergences. Here is an example:

$$\text{Let } p(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} = 0 \\ 0, & \mathbf{x} \neq 0 \end{cases}, \text{ and } q_\theta(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} = \theta \\ 0, & \mathbf{x} \neq \theta \end{cases}$$

$$D_{KL}\left(p, q_\theta\right) = \begin{cases} 0, & \theta = 0 \\ \infty, & \theta \neq 0 \end{cases} \tag{41}$$

$$D_{JS}\left(p, q_\theta\right) = \begin{cases} 0, & \theta = 0 \\ \log 2, & \theta \neq 0 \end{cases}$$

We can find out that no matter what value θ is, the KL divergence is always ∞ and JS divergence is always $\log 2$. Thus, a smoother distance $D(p, q)$ that is defined when p and q have disjoint supports is needed.

Wasserstein Distance is a measure of the distance between two probability distributions. It is also called Earth Mover's distance, short for EM distance because informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution. The distance can be written as

$$D_w(p,q) = \inf_{\gamma \sim \Pi(p,q)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}[\|\mathbf{x} - \mathbf{y}\|], \tag{42}$$

where $\Pi(p, q)$ is the set of all possible joint probability distributions between p and q. One joint distribution $\gamma \in \Pi(p, q)$ describes one dirt transport plan. Precisely $\gamma(\mathbf{x}, \mathbf{y})$ states the percentage of dirt should be transported from point x to y so as to make x follow the same probability distribution of y. In this case, we can get

$$D_w\left(p, q_\theta\right) = \|\theta\|. \tag{43}$$

However, it is intractable to exhaust all the possible joint distributions in $\Pi(p, q)$ to compute $\inf_{\gamma\sim\Pi(p,q)}$. Thus, based on Kantorovich-Rubinstein duality, WGAN transforms the Eq. 42 into

$$D_w(p, q) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim q}[f(\mathbf{x})], \tag{44}$$

where sup (supremum) is the opposite of inf (infimum); we want to measure the least upper bound or, in even simpler words, the maximum value. Thus, the training objective of WGAN with discriminator $D_\phi(\mathbf{x})$ and generator $G_\theta(\mathbf{z})$ is

$$\min_\theta \max_\phi \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_(\mathbf{z})}[D_\phi(G_\theta(\mathbf{z}))]. \tag{45}$$

The Lipschitzness of $D_\phi(\mathbf{x})$ is enforced through weight clipping or gradient penalty. In comparison to the original GAN, the changes of WGAN can be summarized as: 1) Remove the Sigmoid in the output layer of the discriminator as WGAN needs to regress the Wasserstein distance. 2) No log in the loss 3) Weight clipping or gradient penalty 4) Use optimizer without momentum, e.g. RMSProp, SGD. (Empirical conclusion).

6. Energy-based Models (EBMs)

Let us consider a probability distribution $p(\mathbf{x})$. To make a valid probability distribution, it needs to satisfy two criteria: 1) Non-negative, and 2) sum-to-one. Thus, given any $g_{\boldsymbol{\theta}}(\mathbf{x})$ that is non-negative, we can parameterize $p(\mathbf{x})$ as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{Volume(g_{\boldsymbol{\theta}})} g_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{\int g_{\boldsymbol{\theta}}(\mathbf{x}) dx} g_{\boldsymbol{\theta}}(\mathbf{x}). \tag{46}$$

For EBMs, we usually use a function with exponential form $g_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(E_{\boldsymbol{\theta}}(\mathbf{x}))$ instead of a non-smooth function to capture the large variations in the probability. Thus, the density given by an EBM can be written as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{\exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))}{Z_{\boldsymbol{\theta}}}, \tag{47}$$

where $E_{\boldsymbol{\theta}}(\mathbf{x})$ is the energy, which is a nonlinear regression function with parameters $\boldsymbol{\theta}$, and $Z_{\boldsymbol{\theta}}$ denotes the normalizing constant:

$$Z_{\boldsymbol{\theta}} = \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) \, \mathrm{d}\mathbf{x} \tag{48}$$

which is constant w.r.t $\mathbf{x}$ but is a function of $\boldsymbol{\theta}$. Since $Z_{\boldsymbol{\theta}}$ is a function of $\boldsymbol{\theta}$, evaluation and differentiation of $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ w.r.t. its parameters involve a typically intractable integral.

Maximum likelihood Training of EBMs

Let $p_{\boldsymbol{\theta}}(\mathbf{x})$ be a probabilistic model parameterized by $\boldsymbol{\theta}$, and $p_{\text{data}}(\mathbf{x})$ be the underlying data distribution of a dataset, we can fit $p_{\boldsymbol{\theta}}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by maximizing the expected log-likelihood function over the data distribution, defined by

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x})]$$

as a function of $\boldsymbol{\theta}$. Here the expectation can be easily estimated with samples from the dataset. Maximizing likelihood (ML) is equivalent to minimizing the KL divergence between $p_{\text{data}}(\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x})$, because

$$\begin{aligned} -\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x})] &= D_{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_{\text{data}}(\mathbf{x})] \\ &= D_{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) - \text{constant}, \end{aligned}$$

where the second equality holds because $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log p_{\text{data}}(\mathbf{x})]$ does not depend on $\boldsymbol{\theta}$.

Although we cannot directly compute the likelihood of an EBM due to the intractable normalizing constant $Z_{\boldsymbol{\theta}}$, we can still estimate the gradient of the log-likelihood and perform ML with gradient ascent. In particular, the gradient of log-probability in Eq. 47 decomposes into two terms:

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = -\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}) - \nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}}. \tag{49}$$

The gradient of $-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})$ is straightforward to estimate with automatic differentiation. However, it is challenging to approximate the $\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}}$ as it is intractable. This gradient term can be rewritten as the following expection:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}} &= \nabla_{\boldsymbol{\theta}} \log \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&\overset{(i)}{=} \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x}\right)^{-1} \nabla_{\boldsymbol{\theta}} \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&= \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x}\right)^{-1} \int \nabla_{\boldsymbol{\theta}} \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&\overset{(ii)}{=} \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x}\right)^{-1} \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&= \int \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x}\right)^{-1} \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&\overset{(iii)}{=} \int \frac{\exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))}{Z_{\boldsymbol{\theta}}}(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&\overset{(iv)}{=} \int p_{\boldsymbol{\theta}}(\mathbf{x})(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}))d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}(\mathbf{x})}\left[-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})\right],
\end{aligned}
$$

where steps (i) and (ii) are due to the chain rule of gradients, and (iii) and (iv) are from definitions in Eqs. (47) and (48). Thus, we can obtain an unbiased one-sample Monte Carlo estimate of the log-likelihood gradient by

$$
\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}} \simeq -\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}), \tag{50}
$$

where $\tilde{\mathbf{x}} \sim p_{\boldsymbol{\theta}}(\mathbf{x})$, *i.e.*, a random sample from the distribution over $\mathbf{x}$ given by the EBM. Thus, we can approximate the gradient of log-probability in Eq. 49 as:

$$
\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = -\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}) - \nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}} \simeq -\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}). \tag{51}
$$

Therefore, as long as we can draw random samples from the model, we have access to an unbiased Monte Carlo estimate of the log-likelihood gradient, allowing us to optimize the parameters with stochastic gradient ascent.

MCMC Sampling

Some efficient MCMC sampling, such as Langevin MCMC, make use of the fact that the gradient of the log-probability w.r.t. $\mathbf{x}$ (a.k.a., *score*) is equal to the (negative) gradient of the energy, therefore it is easy to calculate:

$$
\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\boldsymbol{\theta}}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_{\boldsymbol{\theta}}}_{=0} = -\nabla_{\mathbf{x}} E_{\boldsymbol{\theta}}(\mathbf{x}). \tag{52}
$$

When using Langevin MCMC to sample from $p_{\boldsymbol{\theta}}(\mathbf{x})$, we first draw an initial sample $\mathbf{x}^0$ from a simple prior distribution, and then simulate an (overdamped) Langevin diffusion process for K steps with step size $\epsilon > 0$:

$$
\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \frac{\epsilon^2}{2} \underbrace{\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^k)}_{=-\nabla_{\mathbf{x}} E_{\boldsymbol{\theta}}(\mathbf{x})} + \epsilon \mathbf{z}^k, \quad k = 0, 1, \cdots, K-1. \tag{53}
$$

When $\epsilon \to 0$ and $K \to \infty$, $\mathbf{x}^K$ is guaranteed to distribute as $p_{\boldsymbol{\theta}}(\mathbf{x})$ under some regularity conditions. In practice, we have to use a small finite ϵ, but the discretization error is typically negligible.

7. Score-based Models

Score Matching

When $f(\mathbf{x})$ and $g(\mathbf{x})$ are log probability density functions (PDFs) with equal first derivatives, the normalization requirement (Eq. (47)) implies that $\int \exp(f(\mathbf{x}))\mathrm{d}\mathbf{x} = \int \exp(g(\mathbf{x}))\mathrm{d}\mathbf{x} = 1$, and therefore $f(\mathbf{x}) \equiv g(\mathbf{x})$. Thus, one can learn an EBM by matching the first derivatives of its log-PDF to the first derivatives of the log-PDF of the data distribution. The first-order gradient function of a log-PDF is also called the *score* of that distribution.

Let $p_{\text{data}}(\mathbf{x})$ be the underlying data distribution. The basic Score Matching (SM) objective minimizes a discrepancy between two distributions called the Fisher divergence:

$$D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x})\right\|^2\right]. \tag{54}$$

However, it is generally impractical to calculate $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ in Eq. (54). It is shown that under certain regularity conditions[1], the Fisher divergence can be rewritten using integration by parts, with second derivatives of $E_{\boldsymbol{\theta}}(\mathbf{x})$ replacing the unknown first derivatives of $p_{\text{data}}(\mathbf{x})$:

$$D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i}\right)^2 + \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{(\partial x_i)^2}\right] + \text{constant} \tag{55}$$

where d is the dimensionality of $\mathbf{x}$. The constant does not affect optimization and thus can be dropped for training. An important downside of the objective Eq. (55) is that, in general, computation of full second derivatives is quadratic in the dimensionality d, thus does not scale to high dimensionality. Therefore, the implicit SM formulation of Eq. (55) has only been applied to relatively simple energy functions where computation of the second derivatives is tractable.

Denoising Score Matching (DSM)

The Score Matching objective in Eq. (55) requires several regularity conditions for $\log p_{\text{data}}(\mathbf{x})$, *e.g.*, it should be continuously differentiable and finite everywhere. However, these conditions may not always hold in practice. For example, a distribution of digital images is typically discrete and bounded.

To alleviate this difficulty, one can add a bit of noise to each datapoint: $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$. As long as the noise distribution $p(\boldsymbol{\epsilon})$ is smooth, the resulting noisy data distribution $q(\tilde{\mathbf{x}}) = \int q(\tilde{\mathbf{x}} \mid \mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}$ is also smooth, and thus the Fisher divergence $D_F(q(\tilde{\mathbf{x}}) \parallel p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}))$ is a proper objective. In order to completely avoid both the unknown term $p_{\text{data}}(\mathbf{x})$ and computationally expensive second-order derivatives, we can use an elegant and scalable objective function (Vincent 2011):

$$D_F(q(\tilde{\mathbf{x}}) \parallel p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})) = \mathbb{E}_{q(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})\right\|_2^2\right] \tag{56}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x}), \tilde{\mathbf{x}}\sim q(\tilde{\mathbf{x}}|\mathbf{x})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}|\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})\right\|_2^2\right] + \text{constant}, \tag{57}$$

where the expectation is again approximated by the empirical average of samples, thus completely avoiding both the unknown term $p_{\text{data}}(\mathbf{x})$ and computationally expensive second-order derivatives. However, $D_F(q(\tilde{\mathbf{x}}) \parallel p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})) \neq D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x}))$ leads to an inconsistent optimization. One way to attenuate the inconsistency of DSM is to choose $q \approx p_{\text{data}}$, *i.e.*, use a small noise perturbation.

1. Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research, 6(Apr):695–709, 2005.

However, this often significantly increases the variance of objective values and hinders optimization. As an example, suppose $q(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 I)$ and $\sigma \approx 0$. The corresponding DSM objective is

$$\begin{aligned} D_F(q(\tilde{\mathbf{x}}) \parallel p_{\boldsymbol{\theta}}(\tilde{\mathbf{x}})) &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\left[\frac{1}{2}\left\|\frac{\mathbf{z}}{\sigma} + \nabla_{\mathbf{x}}\log p_{\boldsymbol{\theta}}(\mathbf{x}+\sigma\mathbf{z})\right\|_2^2\right] \\ &\simeq \frac{1}{2N}\sum_{i=1}^{N}\left\|\frac{\mathbf{z}^{(i)}}{\sigma} + \nabla_{\mathbf{x}}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}+\sigma\mathbf{z}^{(i)})\right\|_2^2, \end{aligned} \tag{58}$$

When $\sigma \to 0$, we can leverage Taylor series expansion to rewrite the Monte Carlo estimator in Eq. (58) to

$$\frac{1}{2N}\sum_{i=1}^{N}\left[\frac{2}{\sigma}(\mathbf{z}^{(i)})^{\mathsf{T}}\nabla_{\mathbf{x}}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + \frac{\left\|\mathbf{z}^{(i)}\right\|_2^2}{\sigma^2}\right] + \text{constant}. \tag{59}$$

When estimating the above expectation with samples, the variances of $(\mathbf{z}^{(i)})^{\mathsf{T}}\nabla_{\mathbf{x}}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})/\sigma$ and $\left\|\mathbf{z}^{(i)}\right\|_2^2/\sigma^2$ will both grow unbounded as $\sigma \to 0$ due to division by σ and σ^2. This enlarges the variance of DSM and makes optimization challenging.

Sliced Score Matching (SSM)

Sliced Score Matching is one alternative to DSM that is both consistent and computationally efficient. Instead of minimizing the Fisher divergence between two vector-valued scores, SSM randomly samples a projection vector $\mathbf{v}$, takes the inner product between $\mathbf{v}$ and the two scores, and then compares the resulting two scalars. More specifically, Sliced Score Matching minimizes the following divergence called the sliced Fisher divergence

$$D_{SF}(p_{\text{data}}(\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{p(\mathbf{v})}\left[\frac{1}{2}(\mathbf{v}^{\mathsf{T}}\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}) - \mathbf{v}^{\mathsf{T}}\nabla_{\mathbf{x}}\log p_{\boldsymbol{\theta}}(\mathbf{x}))^2\right],$$

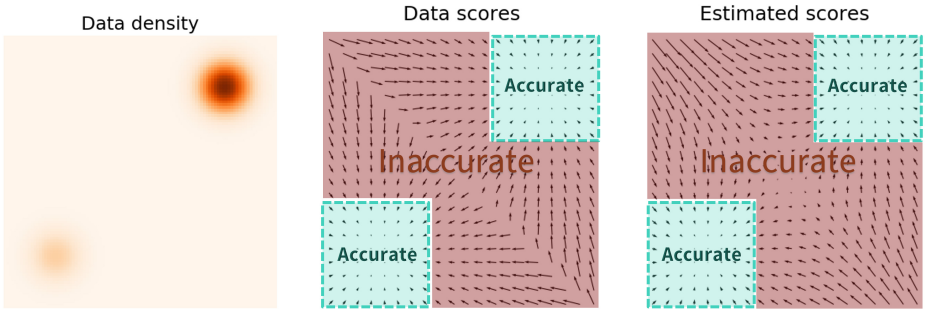
where $p(\mathbf{v})$ denotes a projection distribution such that $\mathbb{E}_{p(\mathbf{v})}[\mathbf{v}\mathbf{v}^{\mathsf{T}}]$ is positive definite. Similar to Fisher divergence, sliced Fisher divergence has an implicit form that does not involve the unknown $\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x})$, which is given by

$$\begin{aligned} &D_{SF}(p_{\text{data}}(\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{x})) \\ &\quad= \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{p(\mathbf{v})}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i}v_i\right)^2 + \sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j}v_i v_j\right] + \text{constant}. \end{aligned} \tag{60}$$

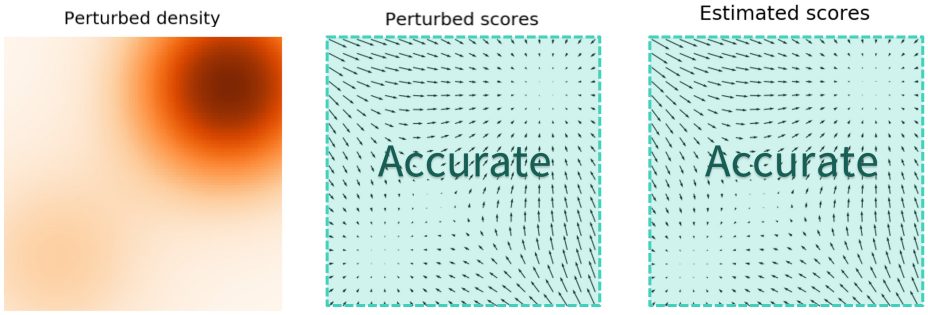
All expectations in the above objective can be estimated with empirical means, and again the constant term can be removed without affecting training. The second term involves second-order derivatives of $E_{\boldsymbol{\theta}}(\mathbf{x})$, but contrary to SM, it can be computed efficiently with a cost linear in the dimensionality d. This is because

$$\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j}v_i v_j = \sum_{i=1}^{d}\frac{\partial}{\partial x_i}\underbrace{\left(\sum_{j=1}^{d}\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_j}v_j\right)}_{:=f(\mathbf{x})}v_i, \tag{61}$$

where $f(\mathbf{x})$ is the same for different values of i. Therefore, we only need to compute it once with $O(d)$ computation, *plus* another $O(d)$ computation for the outer sum to evaluate Eq. (61), whereas the original SM objective requires $O(d^2)$ computation. For many choices of $p(\mathbf{v})$, part of the SSM



a) Estimated scores are only accurate in high-density regions



b) Estimated scores are accurate everywhere for the noise-perturbed data distribution

Figure 9: Illustration of the effect of noise perturbation, which leads to accurate estimation everywhere due to reduced low data density regions.

objective (Eq. (60)) can be evaluated in closed form, potentially leading to lower variance. For example, when $p(\mathbf{v}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, we have

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{p(\mathbf{v})}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i}v_i\right)^2\right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i}\right)^2\right]$$

and as a result,

$$\begin{aligned} &D_{SF}(p_{\text{data}}(\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{x})) \\ &\qquad = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\mathbf{v}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i}\right)^2 + \sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j}v_i v_j\right] + \text{constant}. \end{aligned} \tag{62}$$

Multi-scale Noise Perturbation

The naive score-based models generally have 3 pitfalls: 1) Manifold hypothesis: Data score is under-defined 2) Lagevin dynamic fail to weight different mode correctly. 3) Challenge in low data density regions: Let us assume trained score-based model $s_{\boldsymbol{\theta}}(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$, then the Fisher divergence being minimized can be written as:

$$\mathbb{E}_{p(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - s_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2\right] = \int p(\mathbf{x})\,\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - s_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2\,d\mathbf{x} \tag{63}$$

Since the ℓ_2 differences between the true data score function and score-based model are weighted by $p(\mathbf{x})$, they are largely ignored in low-density regions where $p(\mathbf{x})$is small, and lead to subpar results as in Fig. 9.

To solve such a problem, a general idea is to add multi-scale noise perturbation. We first perturb the data distribution $p(\mathbf{x})$ with each of the Gaussian noise $\mathcal{N}(\mathbf{0}, \sigma_i^2 \boldsymbol{I}), i = 1, 2 \dots L$ to obtain a noise-perturbed distribution

$$p_{\sigma_i} = \int p(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 \boldsymbol{I}) d\mathbf{y} \tag{64}$$

Next, we estimate the score function of each noise-perturbed distribution, $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ by training a Noise Conditional Score-based Model $s_{\boldsymbol{\theta}}(\mathbf{x}, i)$, (also called a Noise Conditional Score Network, or NCSN, when parameterized with a neural network) with score matching, such that $s_{\boldsymbol{\theta}}(\mathbf{x}, i) \approx \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ for all $i = 1, 2, \dots, L$. The training objective for $s_{\boldsymbol{\theta}}(\mathbf{x}, i)$ is a weighted sum of Fisher divergences for all noise scales:

$$\sum_{i=1}^{L} \lambda(i) \mathbb{E}_{p_{\sigma_i}(\mathbf{x})} \left[\| \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}) - s_{\boldsymbol{\theta}}(\mathbf{x}, i) \|_2^2 \right], \tag{65}$$

where $\lambda(i)$ is a positive weighting function, often chosen to be $\lambda(i) = \sigma_i^2$. After training our noise-conditional score-based model $s_{\boldsymbol{\theta}}(\mathbf{x}, i)$, we can produce samples from it by running Langevin dynamics for $i = L, L-1, \dots, 1$, which is called annealed Langevin dynamics since noise scale σ_i decreases gradually over time.

Score-based Models with Stochastic Differential Equations (SDEs)

Perturbing data with an SDE. When the number of noise scales approaches infinity, we essentially perturb the data distribution with continuously growing levels of noise. In this case, the noise perturbation procedure is a continuous-time stochastic process. Stochastic processes are solutions of stochastic differential equations (SDEs). In general, an SDE possesses the following form:

$$\mathrm{d}\mathbf{x} = f(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\omega, \tag{66}$$

where $f(\cdot, t)$ is drift coefficient. $g(t)$ is the diffusion coefficient. $\mathbf{w}$ denotes a standard Brownian motion, and $\mathrm{d}\mathbf{w}$ can be viewed as infinitesimal white noise.

The solution of a stochastic differential equation is a continuous collection of random variables $\{\mathbf{x}(t)\}_{t \in [0,T]}$. Let $p_t(\mathbf{x})$ denote the (marginal) probability density function of $\mathbf{x}(t)$. Here $t \in [0, T]$ is analogous to $i = 1, 2, \dots L$ when we had a finite number of noise scales, and $p_t(\mathbf{x})$ analogous to $p_{\sigma_i}(\mathbf{x})$. Note that $p_T(\mathbf{x})$ is analogous to $p_{\sigma_L}(\mathbf{x})$ in the case of finite noise scales, which corresponds to applying the largest noise perturbation σ_L to the data.

Reversing the SDE for sample generation. Recall that with a finite number of noise scales, we can generate samples by reversing the perturbation process with annealed Langevin dynamics, i.e., sequentially sampling from each noise-perturbed distribution using Langevin dynamics. For infinite noise scales, we can analogously reverse the perturbation process for sample generation by using the reverse SDE:

$$\mathrm{d}\mathbf{x} = \left[f(\mathbf{x}, t) - g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right] \mathrm{d}t + g(t)\mathrm{d}\omega, \tag{67}$$

Here $\mathrm{d}t$ represents a negative infinitesimal time step since the SDE needs to be solved backward in time ($T \to 0$). In order to compute the reverse SDE, we need to estimate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, which is exactly the score function of $p_t(\mathbf{x})$.

Estimating the reverse SDE with score-based models and score matching . Solving the reverse SDE requires us to know the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. In order to estimate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, we train a Time-Dependent Score-Based Model $s_{\boldsymbol{\theta}}(\mathbf{x}, t)$ such that $s_{\boldsymbol{\theta}}(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. This is analogous to the noise-conditional score-based model $s_\theta(\mathbf{x}, i)$ used for finite noise scales, trained such that $s_\theta(\mathbf{x}, i) \approx \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$. The training objective is a continuous weighted combination of Fisher divergences, given by

$$\mathbb{E}_{t \in \mathcal{U}(0,T)} \mathbb{E}_{p_t(\mathbf{x})} \left[\lambda(t) \left\| \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - s_{\boldsymbol{\theta}}(\mathbf{x}, t) \right\|_2^2 \right], \tag{68}$$

where $\mathcal{U}(0, T)$ denotes a uniform distribution over the time interval $[0, T]$, and λ is a positive weighting function. As before, the weighted combination of Fisher divergences can be efficiently optimized with score matching methods, such as DSM, and SSM.