Sampling Methods: From MCMC to Generative Modeling
Part III: Generative Modeling

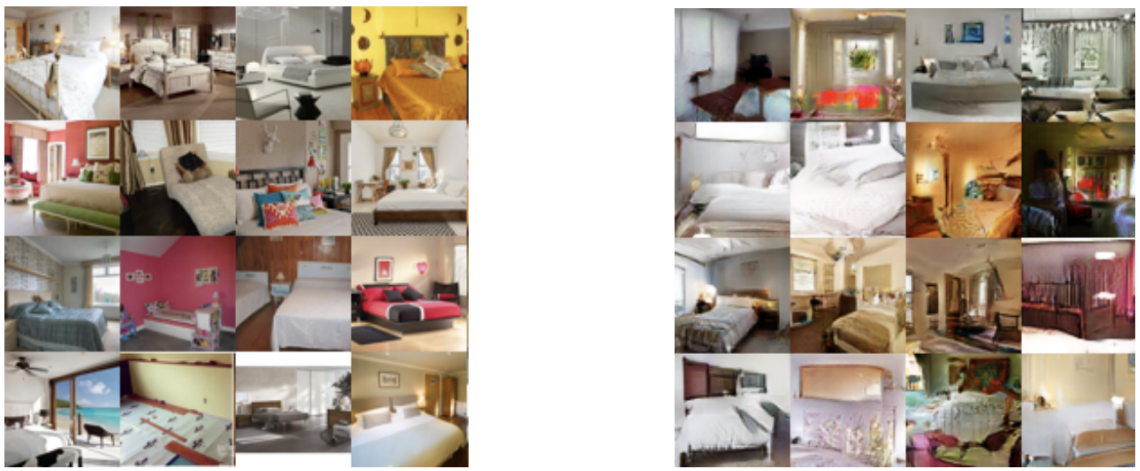
Anna Korba

CREST, ENSAE, Institut Polytechnique de Paris

Ensae - Second semester 2023-2024

Outline

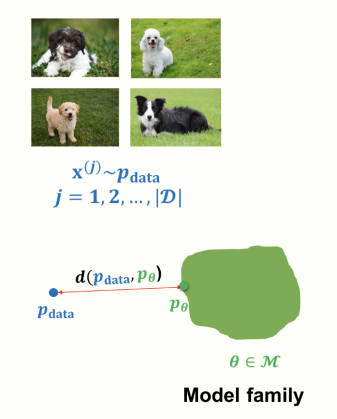
Think of data $x_1, \ldots, x_n \sim p_{data}$ i.i.d. (e.g. images). The goal of generative modeling is to approximate p_{data} given access to a dataset $\mathcal{D} = x_1, \ldots, x_n$.



LSUN bedroom samples vs MMD GAN [Li et al. (2017)].

Learning

Current generative models provide parametric approximations. Pro: parametric methods scale efficiently. Con: may be limited to the choice of model family.



Learning a generative model can be framed as

$$\min_{\theta} D(p_{data}, p_\theta)$$

where D is a distance or divergence between probability distributions.

3 main questions:

- What is the representation for the model family ?
- What is the objective function D?
- What is the optimization procedure for minimizing D?

Inference

We may evaluate a generative model regarding:

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

Inference

We may evaluate a generative model regarding:

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

Disclaimer:

- Quantitative evaluation of generative models is non-trivial (in particular, sampling and representation learning) and an area of active research. Some quantitative metrics exist, but these metrics often fail to reflect desirable qualitative attributes in the generated samples and the learned representations.
- Not all model families permit efficient and accurate inference on all these tasks. Indeed, the trade-offs in the inference capabilities of the current generative models have led to the development of very diverse approaches as we shall see in this course.

Main families of divergences and distances

Let $\mathcal{P}(\mathbb{R}^d)$ denote the space of probability distributions over $\mathbb{R}^d$.

We will pick D a divergence, i.e. s.t. $\mathrm{D}(\mu||\pi) \geq 0$ for any $\mu \in \mathcal{P}(\mathbb{R}^d)$, $\mathrm{D}(\mu||\pi) \Leftrightarrow \mu = \pi$; or a distance (i.e. satisfies triangle inequality).

Main families of divergences and distances are:

- f-divergences:

$$\int f\left(\frac{\mu}{\pi}\right) d\pi, \quad f \text{ convex, } f(1) = 0$$

  defined for $\mu \ll \pi$ (μ absolutely continuous w.r.t. π)

- integral probability metrics (IPM):

$$\sup_{f \in \mathcal{G}} \left| \int f d\mu - \int f d\pi \right|$$

  for $\mathcal{G}$ a class of functions "rich enough"

- optimal transport (OT) distances (cf Marco Cuturi or Austin Stromme's courses)

KL/Likelihood maximization

D could be the (reverse) Kullback-Leibler (KL) divergence:

$$\mathrm{KL}(\mu|\pi) = \begin{cases} \int_{\mathbb{R}^d} \log\left(\frac{\mu}{\pi}(x)\right) d\mu(x) & \text{if } \mu \ll \pi \\ +\infty & \text{otherwise.} \end{cases}$$

We recognize a f-divergence $\int f\left(\frac{\mu}{\pi}\right) d\pi$ where $f(x) = x\log(x)$. Taking $f(x) = -\log(x)$ yields the (forward) KL i.e. $\mathrm{KL}(\pi|\mu)$.

Choosing D as the forward KL, i.e. $\mathrm{D}(\mu_\theta|\pi) = \mathrm{KL}(\pi|\mu_\theta)$ yields Maximum Likelihood, which is useful for fitting a model $(x_1, \ldots, x_n \sim \pi)$ since:

$$\min_\theta \mathrm{KL}(\pi|\mu_\theta) = \int \log\left(\frac{\pi}{\mu_\theta}\right) d\pi$$

$$\Leftrightarrow \min_\theta - \int \log(\mu_\theta(x)) d\pi(x) \approx \sum_{i=1}^{n} \log(\mu_\theta(x_i)) := -\mathcal{L}(\theta|\mathcal{D}).$$

Optimisation is done through (stochastic) gradient ascent of $\mathcal{L}$ (generally non convex)

$$\theta_{t+1} = \theta_t + \gamma_t \nabla_\theta \mathcal{L}(\theta_t|B_t) \tag{1}$$

where B_t is a batch of data at time t. Hyperparameter selection through maximizing likelihood on a validation dataset.

Outline

Representation

Consider a directed, latent variable model as shown below.

In the model above, $\mathbf{z}$ and $\mathbf{x}$ denote the latent and observed variables respectively. The joint distribution expressed by this model is given as

$$p_\theta(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}).$$

From a generative modeling perspective, this model describes a generative process for the observed data $\mathbf{x}$ using the following procedure

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z}).$$

Intuitively, z represents the "high-level" semantic information about x.

Learning

One way to measure how closely $p(\mathbf{x}, \mathbf{z})$ fits the observed dataset D is to measure the Kullback-Leibler (KL) divergence between the data distribution (which we denote as $p_{\text{data}}(\mathbf{x})$) and the model's marginal distribution $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}$.

The distribution that "best" fits the data is thus obtained by minimizing the KL divergence.

$$\min_{p \in \mathcal{P}_{\mathbf{x},\mathbf{z}} := \left\{ p(\mathbf{x},\mathbf{z}) | p(\mathbf{z}) \in \mathcal{P}_{\mathbf{z}}, p(\mathbf{x}|\mathbf{z}) \in \mathcal{P}_{\mathbf{x}|\mathbf{z}} \cdot \right\}} \mathrm{KL}(p_{\text{data}}(\mathbf{x}) || p(\mathbf{x}))$$

As we have seen previously, optimizing an empirical estimate of the KL divergence is equivalent to maximizing the marginal log-likelihood over $\mathcal{D}$:

$$\max_{p \in \mathcal{P}_{\mathbf{x},\mathbf{z}}} \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}.$$

However, it turns out this problem is generally intractable for high-dimensional $\mathbf{z}$ as it involves an integration (or sums in the case $\mathbf{z}$ is discrete) over all the possible latent sources of variation $\mathbf{z}$.

Learning

One option is to estimate the objective via Monte Carlo. For any given datapoint $\mathbf{x}$, we can obtain the following estimate for its marginal log-likelihood

$$\log p(\mathbf{x}) \approx \log \frac{1}{k} \sum_{i=1}^{k} p(\mathbf{x}|\mathbf{z}^{(i)}), \text{ where } \mathbf{z}^{(i)} \sim p(\mathbf{z})$$

.

In practice however, optimizing the above estimate suffers from high variance in gradient estimates.

Rather than maximizing the log-likelihood directly, an alternate is to instead construct a lower bound that is more amenable to optimization.

Next, we introduce a variational family $\mathcal{Q}$ of distributions that approximate the true, but intractable posterior $p(\mathbf{z} \mid \mathbf{x})$. Further henceforth, we will assume a parameteric setting where any distribution in the model family $\mathcal{P}_{\mathbf{x},\mathbf{z}}$ is specified via a set of parameters $\theta \in \Theta$ and distributions in the variational family $\mathcal{Q}$ are specified via a set of parameters $\lambda \in \Lambda$.

Given $\mathcal{P}_{\mathbf{x},\mathbf{z}}$ and $\mathcal{Q}$, we note that the following relationships hold true for any $\mathbf{x}$ and all variational distributions $q_\lambda(\mathbf{z}) \in \mathcal{Q}$:

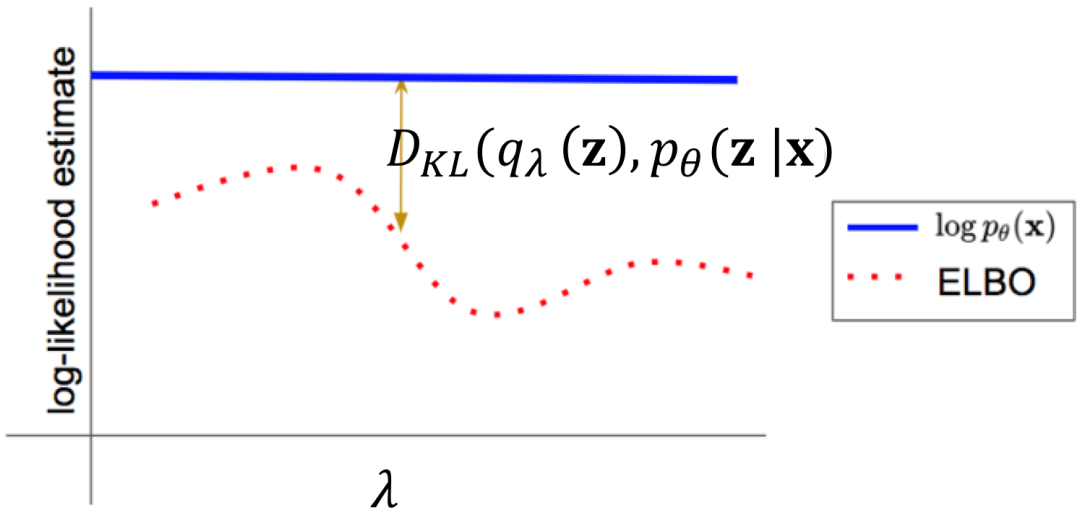
$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int \frac{q_\lambda(\mathbf{z})}{q_\lambda(\mathbf{z})} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&\geq \int q_\lambda(\mathbf{z}) \log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\lambda(\mathbf{z})} d\mathbf{z} \\
&= \mathbb{E}_{q_\lambda(\mathbf{z})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\lambda(\mathbf{z})} \right] \\
&:= \mathrm{ELBO}(\mathbf{x}; \theta, \lambda)
\end{aligned}$$

where we have used Jensen's inequality in the final step. The Evidence Lower Bound or ELBO in short admits a tractable unbiased Monte Carlo estimator

$$\frac{1}{k} \sum_{i=1}^{k} \log \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(i)})}{q_\lambda(\mathbf{z}^{(i)})}, \text{ where } \mathbf{z}^{(i)} \sim q_\lambda(\mathbf{z}), \tag{2}$$

so long as it is easy to sample from and evaluate densities for $q_\lambda(\mathbf{z})$.

Which variational distribution should we pick? Even though the above derivation holds for any choice of variational parameters λ, the tightness of the lower bound depends on the specific choice of q.



In particular, the gap between the original objective (marginal log-likelihood $\log p_\theta(\mathbf{x})$) and the ELBO equals the KL divergence between the approximate posterior $q(\mathbf{z})$ and the true posterior $p(\mathbf{z}|\mathbf{x})$. The gap is zero when the variational distribution $q_\lambda(\mathbf{z})$ exactly matches $p_\theta(\mathbf{z}|\mathbf{x})$.

Black-Box Variational Inference (BBVI)

In summary, we can learn a latent variable model by maximizing the ELBO with respect to both the model parameters θ and the variational parameters λ

$$\max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \max_{\lambda} \mathbb{E}_{q_\lambda(\mathbf{z})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\lambda(\mathbf{z})} \right].$$

First-order stochastic gradient methods

- Step 1: We first do *per-sample* optimization of q by iteratively applying the update

$$\lambda^{(i)} = \lambda^{(i)} + \tilde{\nabla}_\lambda \, \mathrm{ELBO}(\mathbf{x}^{(i)}; \theta, \lambda^{(i)}),$$

  where $\mathrm{ELBO}(\mathbf{x}; \theta, \lambda) = \mathbb{E}_{q_\lambda(\mathbf{z})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\lambda(\mathbf{z})} \right]$, and $\tilde{\nabla}_\lambda$ denotes an unbiased estimate of the ELBO gradient. This step seeks to approximate the log-likelihood $\log p_\theta(\mathbf{x}^{(i)})$.
- Step 2: We then perform a single update step based on the mini-batch

$$\theta = \theta + \tilde{\nabla}_\theta \sum_i \mathrm{ELBO}(\mathbf{x}^{(i)}; \theta, \lambda^{(i)}),$$

  which corresponds to the step that hopefully moves p_θ closer to p_{data}.

Gradient estimation - REINFORCE trick

The gradients $\nabla_\lambda \mathrm{ELBO}$ and $\nabla_\theta \mathrm{ELBO}$ can be estimated via Monte Carlo sampling. While it is straightforward to construct an unbiased estimate of $\nabla_\theta \mathrm{ELBO}$ by simply pushing ∇_θ through the expectation operator, the same cannot be said for ∇_λ. Instead, we see that

$$\nabla_\lambda \, \mathbb{E}_{q_\lambda(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\lambda(\mathbf{z})}\right] = \mathbb{E}_{q_\lambda(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\lambda(\mathbf{z})}\right) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z})\right].$$

This equality follows from the log-derivative trick (also commonly referred to as the REINFORCE trick). The gradient estimator $\tilde{\nabla}_\lambda \mathrm{ELBO}$ is thus

$$\frac{1}{k}\sum_{i=1}^{k}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z}^{(i)})}{q_\lambda(\mathbf{z}^{(i)})}\right) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z}^{(i)})\right], \text{ where } \mathbf{z}^{(i)} \sim q_\lambda(\mathbf{z}).$$

However, it is often noted that this estimator suffers from high variance.

The Reparametrization trick

The reparameterization trick introduces a fixed, auxiliary distribution $p(\epsilon)$ and a differentiable function $T(\epsilon; \lambda)$ such that the procedure

$$\begin{aligned} \epsilon &\sim p(\epsilon) \\ \mathbf{z} &\leftarrow T(\epsilon; \lambda), \end{aligned}$$

is equivalent to sampling from $q_\lambda(\mathbf{z})$. We can see that

$$\nabla_\lambda \mathbb{E}_{q_\lambda(\mathbf{z})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\lambda(\mathbf{z})} \right] = \mathbb{E}_{p(\epsilon)} \left[\nabla_\lambda \log \frac{p_\theta(\mathbf{x}, T(\epsilon; \lambda))}{q_\lambda(T(\epsilon; \lambda))} \right].$$

In contrast to the REINFORCE trick, the reparameterization trick is often noted empirically to have lower variance and thus results in more stable training.

VAE [Kingma and Welling (2013)]

So far, we have described $p_\theta(\mathbf{z})$, $p_\theta(\mathbf{x}|\mathbf{z})$, and $q_\lambda(\mathbf{z})$ abstractly.

- A popular choice for $p_\theta(\mathbf{z})$ is the unit Gaussian $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, I)$.
- $p_\theta(\mathbf{x}|\mathbf{z})$ is where we introduce a deep neural network:

$$p_\theta(\mathbf{x}|\mathbf{z}) = p_\omega(\mathbf{x}) \text{ , where } \omega = g_\theta(\mathbf{z}),\ g_\theta : \mathcal{Z} \to \Omega.$$

  The function g_θ (tyicall a deep NN) is also referred to as the decoding distribution since it maps a latent *code* $\mathbf{z}$ to the parameters of a distribution over observed variables $\mathbf{x}$.
  Example: $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$.
- choose q_λ such that the reparametrization trick is possible. For instance, Gaussians:

$$\begin{aligned} \lambda &= (\mu, \Sigma) \\ q_\lambda(\mathbf{z}) &= \mathcal{N}(\mathbf{z}|\mu, \Sigma) \\ p(\epsilon) &= \mathcal{N}(\epsilon|0, I) \\ T(\epsilon; \lambda) &= \mu + \Sigma^{1/2}\epsilon, \end{aligned}$$

  where $\Sigma^{1/2}$ is the Cholesky decomposition of Σ. For simplicity, practitioners often restrict Σ to be a diagonal matrix (which restricts the distribution family to that of factorized Gaussians).

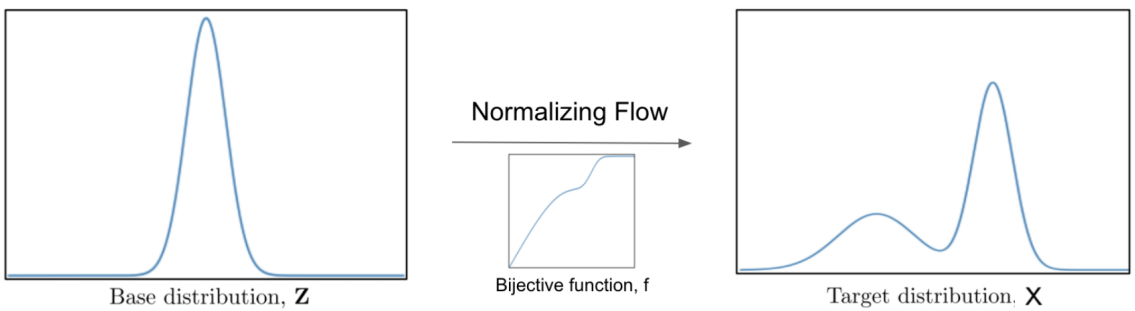
Normalizing Flows

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 describe how to evaluate densities of a random variable that is a deterministic transformation from another variable.

Let Z and X be random variables which are related by an (invertible) mapping $f : \mathbb{R}^d \to \mathbb{R}^d$ such that $X = f(Z)$ and $Z = f^{-1}(X)$. Then

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



There are several things to note here.

- x and z need to be continuous and have the same dimension. $\frac{\partial f^{-1}(x)}{\partial x}$ is the Jacobian matrix at x, i.e. a matrix of dimension $d \times d$, where each entry at location (i, j) is defined as $\frac{\partial f_i^{-1}(x)}{\partial x_j}$.
- $\det(A)$ denotes the determinant of a square matrix A
- For any invertible matrix A, $\det(A^{-1}) = \det(A)^{-1}$, so for $z = f^{-1}(x)$ we have

$$p_X(x) = p_Z(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| \tag{3}$$

- Typically we choose Z such with a known density function p_Z (to be able to evaluate (3) in the likelihood) and easy to sample from, to generate new samples as $Z \sim p_Z$, $X = f(Z)$.

Normalizing Flows: generic principle

In a Normalizing flow model, the mapping between Z and X is chosen as $f_\theta : \mathbb{R}^d \to \mathbb{R}^d$; it is deterministic and invertible such that $X = f_\theta(Z)$ and $Z = f_\theta^{-1}(X)$. Assume we have a dataset $x_1, \ldots, x_n \sim p_X$. We can define the MLE:

$$\hat{\theta} = \arg\max_\theta \sum_{i=1}^n \log(p_Z(f_\theta^{-1}(x_i))) + \log\left(\left|\det\left(\frac{\partial f_\theta^{-1}(x_i)}{\partial x_i}\right)\right|\right).$$

We need both f_θ such that:

- $\det\left(\frac{\partial f_\theta^{-1}(x_i)}{\partial x_i}\right)$ is tractable (complexity of inverting a $d \times d$ matrix: $\mathcal{O}(d^3)$)
- f_θ is "expressive" enough

To get rich enough mappings, one can consider a compositional transformation $X = F(Z) = f_{\theta_k} \circ \cdots \circ f_{\theta_1}(Z)$. In this case the change of variables formula is:

$$p_X(x) = p_Z(z_1) \prod_{j=1}^k \left|\det\left(\frac{\partial f_{\theta_j}(z_j)}{\partial z_j}\right)\right|^{-1}$$

where $z_1 = z$, $z_{j+1} = f_{\theta_j}(z_j)$ for $j = 1, \ldots, k$.

Example: Triangular maps

We will consider $F = f_{\theta_k} \circ \cdots \circ f_{\theta_1}$ to be a triangular map, i.e. for $x = (x^1, \ldots, x^d) \in \mathbb{R}^d$ and $z = (z^1, \ldots, z^d) \in \mathbb{R}^d$, we want $f_{\theta_j}(z)$ to be a function of the first j coordinates $(z^1, \ldots, z^j)$ (instead of all d coordinates).

The Jacobian (denoted J) for a triangular map is shown below.

$$JF_\theta(z) = \begin{bmatrix} \frac{\partial f_{\theta_1}(z)}{\partial z^1} & 0 & \cdots & 0 \\ \frac{\partial f_{\theta_2}(z)}{\partial z^1} & \frac{\partial f_{\theta_2}(z)}{\partial z^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{\theta_d}(z)}{\partial z^1} & \frac{\partial f_{\theta_d}(z)}{\partial z^2} & \cdots & \frac{\partial f_{\theta_d}(z)}{\partial z^d} \end{bmatrix}$$

The determinant is simply the product of the diagonals and has a complexity of $\mathcal{O}(d)$ instead of $\mathcal{O}(d^2)$. The complexity for the calculation of the inverse is $\mathcal{O}(d^2)$ instead of $\mathcal{O}(d^3)$.

Other examples

- Choose f_{θ_j} as a structured invertible neural network
- if one lets $k \to \infty$, we can see F as being the solution of a "continuous time normalizing flow" [Chen et al. (2018)], or "flow matching" [Lipman et al. (2022)]
- NF have also been used as proposals for Importance Sampling [Müller et al. (2019)] ("Neural IS").

Outline

Likelihood-free training

We now move onto another family of generative models called generative adversarial networks (GANs).

Likelihood-free training

We now move onto another family of generative models called generative adversarial networks (GANs).

GANs are unique from all the other model families that we have seen so far, such as autoregressive models, VAEs, and normalizing flow models, because **we do not train them using maximum likelihood**.

Likelihood-free training

We now move onto another family of generative models called generative adversarial networks (GANs).

GANs are unique from all the other model families that we have seen so far, such as autoregressive models, VAEs, and normalizing flow models, because **we do not train them using maximum likelihood**.

Why not? In fact, it is not so clear that better likelihood numbers necessarily correspond to higher sample quality. We know that the *optimal generative model* will give us the best sample quality and highest test log-likelihood. However, models with high test log-likelihoods can still yield poor samples, and vice versa.

Likelihood-free training

We now move onto another family of generative models called generative adversarial networks (GANs).

GANs are unique from all the other model families that we have seen so far, such as autoregressive models, VAEs, and normalizing flow models, because **we do not train them using maximum likelihood**.

Why not? In fact, it is not so clear that better likelihood numbers necessarily correspond to higher sample quality. We know that the *optimal generative model* will give us the best sample quality and highest test log-likelihood. However, models with high test log-likelihoods can still yield poor samples, and vice versa.

To see why, consider pathological cases in which our model is comprised almost entirely of noise, or our model simply memorizes the training set. Therefore, we turn to *likelihood-free training* with the hope that optimizing a different objective will allow us to disentangle our desiderata of obtaining high likelihoods as well as high-quality samples.

Two sample test and proxy

Recall that maximum likelihood required us to evaluate the likelihood of the data under our model p_θ. A natural way to set up a likelihood-free objective is to consider the two-sample test, a statistical test that determines whether or not a finite set of samples from two distributions are from the same distribution using only samples from P and Q.

Concretely, given $S_1 = \{\mathbf{x} \sim P\}$ and $S_2 = \{\mathbf{x} \sim Q\}$, we compute a test statistic T according to the difference in S_1 and S_2 that, when less than a threshold α, accepts the null hypothesis that $P = Q$.

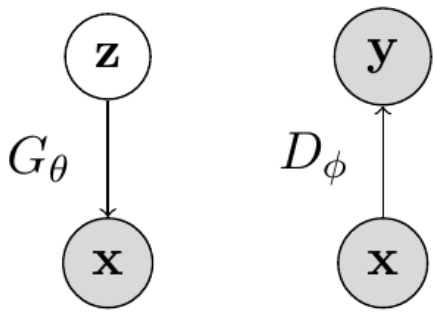
Analogously, we have in our generative modeling setup access to our training set $S_1 = \mathcal{D} = \{\mathbf{x} \sim p_{\text{data}}\}$ and $S_2 = \{\mathbf{x} \sim p_\theta\}$. The key idea is to train the model to minimize a two-sample test objective between S_1 and S_2.

But this objective becomes extremely difficult to work with in high dimensions, so we choose to optimize a surrogate objective that instead maximizes some distance between S_1 and S_2.

GAN - Generative Adversarial Network

We thus arrive at the generative adversarial network formulation. There are two components in a GAN: (1) a generator and (2) a discriminator.

The generator G_θ is a directed latent variable model that deterministically generates samples $\mathbf{x}$ from $\mathbf{z}$, and the discriminator D_ϕ is a function whose job is to distinguish samples from the real dataset and the generator.



The image above is a graphical model of G_θ and D_ϕ. $\mathbf{x}$ denotes samples (either from data or generator), $\mathbf{z}$ denotes our noise vector, and $\mathbf{y}$ denotes the discriminator's prediction about $\mathbf{x}$.

GAN objective

The generator and discriminator both play a two player minimax game, where the generator minimizes a two-sample test objective ($p_{\text{data}} = p_\theta$) and the discriminator maximizes the objective ($p_{\text{data}} \neq p_\theta$).

Intuitively, the generator tries to fool the discriminator to the best of its ability by generating samples that look indistinguishable from p_{data}.

Formally, the GAN objective can be written as:

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

GAN objective

- For a fixed generator G_θ, the discriminator is maximizing the objective w.r.t. ϕ: it assigns probability 1 to data points from the training set $\mathbf{x} \sim p_{\text{data}}$, and assigns probability 0 to generated samples $\mathbf{x} \sim p_G$. In this setup, the optimal discriminator is:

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

- For a fixed discriminator D_ϕ, the generator minimizes this objective .

And after performing some algebra, plugging in the optimal discriminator $D_G^*(\cdot)$ into the overall objective $V(G_\theta, D_G^*(\mathbf{x}))$ gives us:

$$2D_{\text{JSD}}[p_{\text{data}}, p_G] - \log 4$$

The D_{JSD} term is the Jenson-Shannon Divergence:

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

With this distance metric, the optimal generator for the GAN objective becomes $p_G = p_{\text{data}}$. Proof: see `https://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/gan-foundations.pdf`.

GAN training

the way in which we train a GAN is as follows:
For epochs $1, \ldots, N$ do:

1. Sample minibatch of size m from data: $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \sim \mathcal{D}$
2. Sample minibatch of size m of noise: $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)} \sim p_z$
3. Take a gradient descent step on the generator parameters θ:

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

4. Take a gradient ascent step on the discriminator parameters ϕ:

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

Challenges

During optimization, the generator and discriminator loss often continue to oscillate without converging to a clear stopping point.

Due to the lack of a robust stopping criteria, it is difficult to know when exactly the GAN has finished training. Additionally, the generator of a GAN can often get stuck producing one of a few types of samples over and over again (mode collapse).

Most fixes to these challenges are empirically driven, and there has been a significant amount of work put into developing new architectures, regularization schemes, and noise perturbations in an attempt to circumvent these issues.

GANS Variants

We have seen GAN's original objective writes $D_{\mathrm{JSD}}[p_{\mathrm{data}}, p_G]$. Several variants exist

- f-GANs leverage the variational formulation of f-divergences

$$D_f(p, q) = \mathbb{E}_{\mathbf{x}\sim q}\left[f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right)\right] \geq \sup_{T\in\mathcal{T}}\left(\mathbb{E}_{x\sim p}[T(\mathbf{x})] - \mathbb{E}_{x\sim q}[f^*(T(\mathbf{x}))]\right)$$

  f-Gans objective:

$$\min_{\theta}\max_{\phi} F(\theta, \phi) = \mathbb{E}_{x\sim p_{\mathrm{data}}}[T_\phi(\mathbf{x})] - \mathbb{E}_{x\sim p_{G_\theta}}[f^*(T_\phi(\mathbf{x}))]$$

  where $f^*(x) = \sup_s \langle s, x\rangle - f(s)$. Intuitively, we can think about this objective as the generator trying to minimize the divergence estimate, while the discriminator tries to tighten the lower bound.

- IPMs Gans: MMD Gans [Bińkowski et al. (2018)], Wasserstein Gans [Arjovsky et al. (2017)] ...

Outline

Forward process

In score-based generative models or denoising diffusion probabilistic modeling (DDPM), a forward process transforms the unknown data distribution to a prior distribution (which is generally a Gaussian distribution).

The forward process is generally modeled as a solution to an Itô stochastic differential equation (SDE):

$$dX_t = f(t, X_t)dt + g(t)dB_t, \quad X_0 \sim q_0$$

where B_t is the standard Brownian motion in $\mathbb{R}^d$, $f(t, X_t)$ is called the drift coefficient, $g(t)$ is known as the diffusion coefficient, and q_0 denotes the unknown data distribution in $\mathbb{R}^d$.

The goal of the forward process is to ensure that X_t converges to a fixed prior distribution that does not depend on q_0 or other coefficients.

Although there are multiple options to enable such a convergence, the most tractable scheme is the DDPM . It sets the drift coefficient to be an affine function or simply sets

$$dX_t = -X_t dt + g(t)dB_t, \quad X_0 \sim q_0, (2)$$

which ensures the conditional distribution of X_t on X_0 is always a Gaussian distribution and can be calculated in closed form

In this case, if the distribution or law of X_t is denoted by q_t, then q_t converges to a unit Gaussian distribution $\mathcal{N}(0, \mathrm{Id})$ exponentially fast.

Variance of the forward process

When the drift coefficient is set, the variance of the forward process is determined by the diffusion coefficient $g(t)$.

Specifically, the unique solution to the previous SDE can be obtained as

$$X_t = e^{-t}X_0 + \int_0^t e^{s-t} g^2(s) dB_s$$

and the conditional distribution of X_t given X_0 is then

$$q_{t|0}(X_t|X_0) = \mathcal{N}(e^{-t}X_0, 2\int_0^t e^{s-t} g^2(s) ds).$$

When the diffusion coefficient g is smaller, the forward process has a smaller variance.

Reverse process

If we reverse the forward process in in time, we then obtain a process that transforms a prior distribution into q0. Under mild conditions, the solution X_t to the reverse process SDE:

$$dX_t^{\leftarrow} = (-X_t^{\leftarrow} - g(t)^2 \nabla_X \log q_t^{\leftarrow}(X_t^{\leftarrow}))dt + g(t)dB_t$$

where $q_t^{\leftarrow} = q_{T-t}$. $\nabla_X \log q_t$ represents the score function for q_t.

If the score functions were precisely known for all $t \in [0, T]$, the reverse process would be uniquely determined, allowing the distribution transformation.

Score matching (ideally)

Since the score functions are unknown in practice, a neural network model $s_\theta(X, t)$ is trained to approximate these score functions, by minimizing the loss

$$J(\theta, g) = \int_0^T \mathbb{E}_{q_t(X)}[g^2(t)\|\nabla \log q_t(X) - s_\theta(X, t)\|^2]dt$$

and the score function estimation is set to be s_{θ^*} with $\theta^* = \arg\min_\theta J(\theta, g)$.

Now, replace the unknown score functions $\nabla \log q_t$ by s_{θ^*} in the reverse process, and we obtain the approximated reverse SDE:

$$dX_t = (-X_t - g(t)2s_{\theta^*}(X_t, t))dt + g(t)dB_t$$

Time-discretization

The previous objective function considers the score matching error over the whole time period $[0, T]$, making the training difficult and expensive.

Idea: partition the time horizon $[0, T]$ into N sub-intervals: $t_0 = 0 < t_1 < \cdots < t_N = T$, and freeze the time value t in the score function when $t \in [t_k, t_{k+1})$. The reverse SDE then reduces to

$$dX_t = (-X_t - g(t)^2 s_\theta(X_{t_k}, t_k))dt + g(t)dB_t, \quad t \in [t_k, t_{k+1})$$

In this scenario, the estimation of the score function is only required for N times.

References I

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

Bińkowski, M., Sutherland, D. J., Arbel, M., and Gretton, A. (2018). Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. *Advances in neural information processing systems*, 31.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Li, C.-L., Chang, W.-C., Cheng, Y., Yang, Y., and Póczos, B. (2017). Mmd gan: Towards deeper understanding of moment matching network. *Advances in neural information processing systems*, 30.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. (2022). Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. (2019). Neural importance sampling. *ACM Transactions on Graphics (ToG)*, 38(5):1–19.