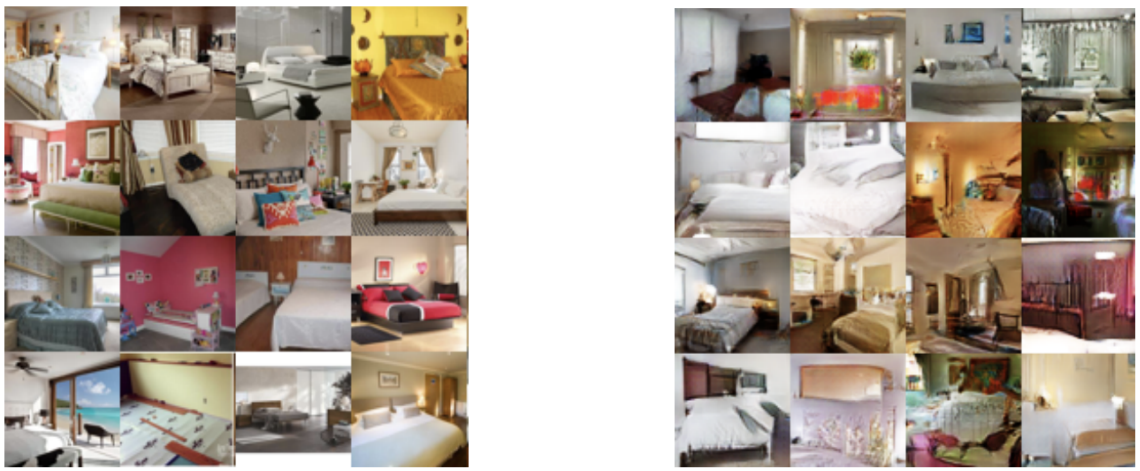
Sampling Methods: From MCMC to Generative Modeling
Generative Modeling - 1

Anna Korba

CREST, ENSAE, Institut Polytechnique de Paris

Outline

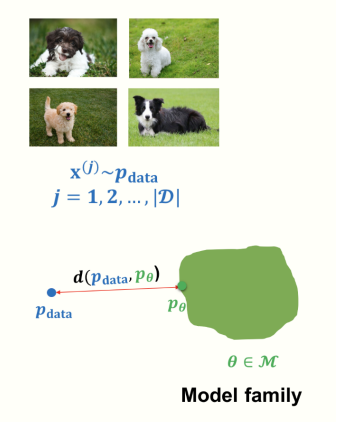
Think of data $x_1, \ldots, x_n \sim p_{data}$ i.i.d. (e.g. images). The goal of generative modeling is to approximate p_{data} given access to a dataset $\mathcal{D} = x_1, \ldots, x_n$.



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.

Outline

KL/Likelihood maximization

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:

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

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

Optimisation can be done through (stochastic) gradient ascent of $\mathcal{L}$

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

where B_t is a batch of data at time t.

Generative process with latent variables

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 $\mathcal{D}$ 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.

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

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.

We will introduce a variational family $\mathcal{Q}$ of distributions that approximate $p(x, z)$. 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 $q_\lambda(\mathbf{z}) \in \mathcal{Q}$:

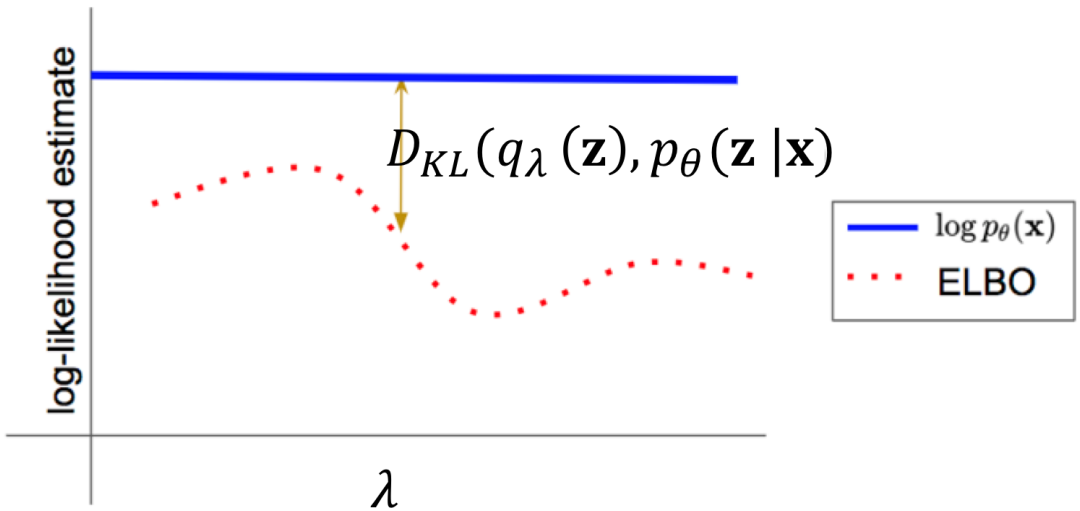
$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int \frac{q_\lambda(\mathbf{z})}{q_\lambda(\mathbf{z})} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&\geq \int q_\lambda(\mathbf{z}) \log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\lambda(\mathbf{z})}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

where we have used Jensen's inequality. Note that $q_\lambda(z)$ stands for $q_\lambda(z|x)$. The Evidence Lower Bound (ELBO) 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 w.r.t. both the (generative) model parameters θ (**decoder**) and the variational parameters λ (**encoder**)

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

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 (typically 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).

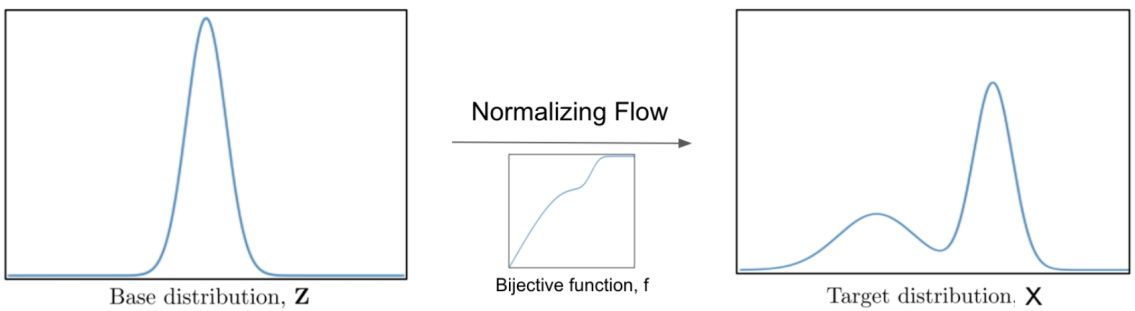
Normalizing Flows

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

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



- 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 $|\det\left(\frac{\partial f(z)}{\partial z}\right)|^{-1} = |\det\left(\frac{\partial f(z)}{\partial z}^{-1}\right)|$, and by the implicit function theorem we have $\frac{\partial f(z)}{\partial z}^{-1} = \frac{\partial f^{-1}(x)}{\partial x}$, 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 $Z \sim p_Z$, $X = f(Z)$.

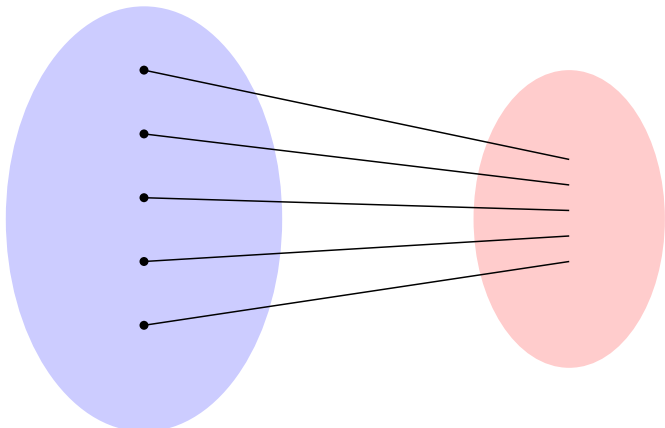
NF vs VAE

NF Generative process is

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} = f(\mathbf{z})$$

VAE generative process

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\boldsymbol{\eta} = f(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\eta})$$



If we want to calculate the density of $\mathbf{x}$, we need to use the **change-of-variable formula** for probability densities.

This requires f to be **invertible** (bijection) and **differentiable**.

In VAEs dim $\mathbf{z}$ < dim $\mathbf{x}$, hence f not bijective

A simple example (1)

For $z \in \mathbb{R}$, let $p(z) = \mathcal{N}(z \mid 0, 1)$ and $x = T(z) = 2z + 1$

Q: What is the density of x?
A: The density is $p_x(x) = \mathcal{N}(x \mid 1, 2^2)$

The change of variable formula for scalar densities:

$$p_x(x) = p_z(z) \left| \frac{dz}{dx} \right| = p_z(T^{-1}(x)) \left| \frac{d}{dx} T^{-1}(x) \right|$$

We have that

$$T^{-1}(x) = \frac{x-1}{2} \quad \text{and} \quad \left| \frac{d}{dx} T^{-1}(x) \right| = \frac{1}{2}$$

Using the Gaussian formula: $\mathcal{N}(z \mid 0, 1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} z^2\right)$

$$\Rightarrow \quad p_x(x) = p_z\left(\frac{x-1}{2}\right) \cdot \frac{1}{2} = \frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-1}{2}\right)^2\right) = \mathcal{N}(x \mid 1, 2^2)$$

A simple example (2)

- Recall **inverse transform sampling**:
  - Procedure for sampling from $p(x)$
  - $p(x)$ is any[1] continuous univariate probability distribution
  - Use the cumulative distribution function (CDF):

$$F(x) = \int_{-\infty}^{x} p(x')\, dx'$$

- We obtain samples x from $p(x)$ by the procedure:

$$z \sim \mathcal{U}(0,1), \quad x = F^{-1}(z)$$

- This is a flow with $p(z) = \mathcal{U}(0,1)$ and $T(z) = F^{-1}(z)$!
- Assuming CDF is invertible, this transforms $\mathcal{U}(0,1)$ to any distribution.
- So, a flow can represent nearly any distribution!
  - A similar argument can be made in multiple dimensions.

[1] $p(x)$ must be differentiable wrt. x and the CDF must be invertible (works if $p(x) > 0$ for all x)

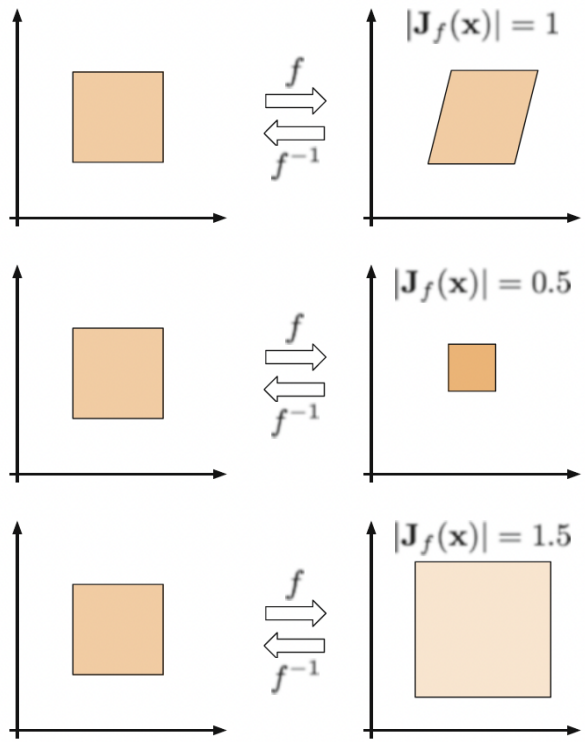
Flows: the Jacobian

Recall Z and X are 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

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

- $\left| \det \left(\frac{\partial f(z)}{\partial z} \right) \right|$ quantifies the relative change of volume around $\mathbf{z}$ due to f
- When $\left| \det \left(\frac{\partial f(z)}{\partial z} \right) \right| = 1$, then transformation is *volume preserving*
- we have the equivalent formulation:

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



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 f_θ such that both:

- $\det\left(\frac{\partial f_\theta^{-1}(x_i)}{\partial x_i}\right)$ is tractable
- f_θ is "expressive" enough

Regarding tractability:

- for MLE we need f_θ^{-1} and $\frac{\partial f_\theta^{-1}(x)}{\partial x}$ (or $\frac{\partial f_\theta(z)}{\partial z}$) (we don't need f_θ)
- for sampling/generating we need f_θ

Stacking: Composable transformation

To get rich enough mappings, one can consider a compositional transformation $X = F(Z) = f_{\theta_k} \circ \cdots \circ f_{\theta_1}(Z)$.

- it is a diffeomorphism (i.e. F is invertible and F, F^{-1} are differentiable) if each of the f_{θ_j} satisfies this property
- In this case the change of variables formula is (chain rule for the Jacobian):

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

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

Hence the name "Normalizing Flows"

- "Flow": Refers to the sequence of transformation $F = f_{\theta_k} \circ \cdots \circ f_{\theta_1}$ where a sample z flows from the base to the output x
- "Normalising": Refers to the reverse flow where a data point x flows back to a normal distributed base, i.e., data is normalised

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 f_θ 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 $\mathcal{O}(d^2)$. instead of $\mathcal{O}(d^3)$.
- The **Jacobian of** F is: $J_F(z) = J_{f_{\theta_k}}(f_{\theta_{k-1}} \circ \cdots \circ f_{\theta_1}(z)) \cdots J_{f_{\theta_1}}(z)$
- Each $J_{f_{\theta_j}}$ has the lower triangular structure as shown above, so their product as well !

Example: Affine coupling layers

- **Affine coupling** layers were introduced by Dinh et al. (2015)
  - Key components in RealNVP[2] (Real-valued Non-Volume Preserving flows)
- The transformation $T : \mathbf{z} \to \mathbf{z}'$ partitions $\mathbf{z} = [\mathbf{z}_{\leq d}, \mathbf{z}_{>d}]$ at $d < D$:

$$\mathbf{z}'_{\leq d} = \mathbf{z}_{\leq d}$$
$$\mathbf{z}'_{>d} = \exp\left(s(\mathbf{z}_{\leq d})\right) \odot \mathbf{z}_{>d} + t(\mathbf{z}_{\leq d})$$

where $s : \mathbb{R}^d \to \mathbb{R}^{D-d}$, $t : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and we denoted
$\forall i \leq d : z'_i = z_i$
$\forall i > d : z'_i = \exp(s(\mathbf{z}_{\leq d})_i) z_i + t(\mathbf{z}_{\leq d})_i$ (Entry-wise product)
i.e., one affine layer "transforms second half using first half of the input".
This builds up dependencies across all variables. It is a special case of "autoregressive floww".

Example: Affine coupling layers

- **Affine coupling** layers were introduced by Dinh et al. (2015)
  - Key components in RealNVP[2] (Real-valued Non-Volume Preserving flows)
- The transformation $T : \mathbf{z} \to \mathbf{z}'$ partitions $\mathbf{z} = [\mathbf{z}_{\leq d}, \mathbf{z}_{>d}]$ at $d < D$:

$$\mathbf{z}'_{\leq d} = \mathbf{z}_{\leq d}$$
$$\mathbf{z}'_{>d} = \exp\left(s(\mathbf{z}_{\leq d})\right) \odot \mathbf{z}_{>d} + t(\mathbf{z}_{\leq d})$$

  where $s : \mathbb{R}^d \to \mathbb{R}^{D-d}$, $t : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and we denoted
  $\forall i \leq d : z'_i = z_i$
  $\forall i > d : z'_i = \exp(s(\mathbf{z}_{\leq d})_i) z_i + t(\mathbf{z}_{\leq d})_i$ (Entry-wise product)
  i.e., one affine layer "transforms second half using first half of the input".
  This builds up dependencies across all variables. It is a special case of "autoregressive floww".
- Since $\mathbf{z}'_{>d}$ is an affine transformation of $\mathbf{z}_{>d}$, it's easily invertible (unlike standard neural networks!):

$$\mathbf{z}_{\leq d} = \mathbf{z}'_{\leq d}$$
$$\mathbf{z}_{>d} = \exp\left(-s(\mathbf{z}_{\leq d})\right) \odot \left(\mathbf{z}'_{>d} - t(\mathbf{z}_{\leq d})\right)$$

[1] Dinh L, Krueger D, and Bengio Y. NICE: Non-linear independent components estimation. ICLR workshop track, 2015.
[2] Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017.

Affine coupling layers, the Jacobian

Let's write it in the following block form:

$$\frac{\partial T(z)}{\partial z} = \begin{bmatrix} \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} \\ \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{>d}} \end{bmatrix}$$

Recall that $\mathbf{z}'_{\leq d} = \mathbf{z}_{\leq d}$, and so we find

$$\frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{\leq d}} = \begin{bmatrix} \frac{\partial z'_1}{\partial z_1} & \cdots & \frac{\partial z'_1}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial z'_d}{\partial z_1} & \cdots & \frac{\partial z'_d}{\partial z_d} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} = I_{d\times d}$$

$$\frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} = \begin{bmatrix} \frac{\partial z'_1}{\partial z_{d+1}} & \cdots & \frac{\partial z'_1}{\partial z_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial z'_d}{\partial z_{d+1}} & \cdots & \frac{\partial z'_d}{\partial z_D} \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} = 0_{d\times(D-d)}$$

Affine coupling layers, the Jacobian

Let's write it in the following block form:

$$\frac{\partial T(z)}{\partial z} = \begin{bmatrix} \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} \\ \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{>d}} \end{bmatrix}$$

Recall that $\forall i > d : z'_i = \exp(s(\mathbf{z}_{\leq d})_i) z_i + t(\mathbf{z}_{\leq d})_i$

$$\frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{>d}} = \begin{bmatrix} \frac{\partial z'_{d+1}}{\partial z_{d+1}} & \cdots & \frac{\partial z'_{d+1}}{\partial z_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial z'_D}{\partial z_{d+1}} & \cdots & \frac{\partial z'_D}{\partial z_D} \end{bmatrix} = \begin{bmatrix} \exp(s(\mathbf{z}_{\leq d})_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \exp(s(\mathbf{z}_{\leq d})_{D-d}) \end{bmatrix}$$
$$= \operatorname{diag}(\exp(s(\mathbf{z}_{\leq d})))$$

Affine coupling layers, the Jacobian

Let's write it in the following block form:

$$\frac{\partial T(z)}{\partial z} = \begin{bmatrix} \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{\leq d}}{\partial \mathbf{z}_{>d}} \\ \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{\leq d}} & \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{>d}} \end{bmatrix}$$

- This means that:

$$J_T(\mathbf{z}) = \begin{bmatrix} I_{d\times d} & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{z}'_{>d}}{\partial \mathbf{z}_{\leq d}} & \mathrm{diag}(\exp(s(\mathbf{z}_{\leq d}))) \end{bmatrix}$$

- Since $J_T(\mathbf{z})$ is triangular, we have:

$$\det J_T(\mathbf{z}) = \prod_{i=1}^{D-d} \exp(s(\mathbf{z}_{\leq d})_i) = \exp\left(\sum_{i=1}^{D-d} s(\mathbf{z}_{\leq d})_i\right)$$

RealNVP

RealNVP[1] stacks many affine coupling layers (with masking or permutations) to get complex transformations.

- Recall one affine coupling layer transforms part of the input while keeping the rest unchanged:
  - Layer 1: transform second half using first half
  - Layer 2: transform second half using first half
  - And so on

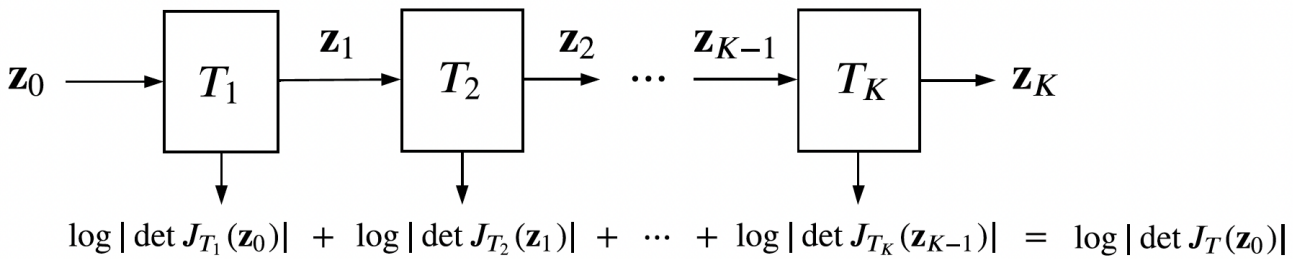
  This lets the model slowly build up complex dependencies across all variables.
- To alternate which part of the input gets transformed, RealNVP uses **(binary)masks**: A mask is just a binary vector like [1, 1, 0, 0], which decides which part of the input is used as-is, and which part is transformed. Masks alternate across layers so that every variable gets updated at some point.

[1] Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP. ICLR, 2017.

Towards continuous time normalising flows

- The residual flow transformation, T, looks a lot like Euler discretization:

$$\mathbf{z}_{t+1} = T(\mathbf{z}_t) = \mathbf{z}_t + g_\phi(\mathbf{z}_t)$$



$$\log|\det J_{T_1}(\mathbf{z}_0)| + \log|\det J_{T_2}(\mathbf{z}_1)| + \cdots + \log|\det J_{T_K}(\mathbf{z}_{K-1})| = \log|\det J_T(\mathbf{z}_0)|$$

- In the limit of infinite number layers and $h \to 0$, we are solving the ODE:

$$\frac{d\mathbf{z}_t}{dt} = g_\phi(\mathbf{z}_t), \quad \text{with continuous time } t \in [0, K]$$

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 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 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 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 $P = Q$.

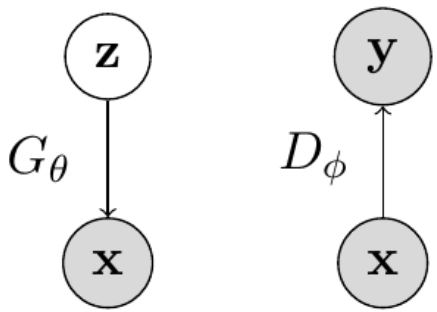
Analogously, we have in our generative modeling setup access to our training set $S_1 = \mathcal{D} = \{\mathbf{x} \sim p_{\text{data}}\}$ and $S_2 = \{\mathbf{x} \sim p_\theta\}$. The key idea is to train the model to minimize a two-sample test objective between S_1 and S_2.

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 **x** from **z**, and the discriminator D_ϕ is a classifier 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. **x** denotes samples (either from data or generator), **z** denotes our noise vector, and **y** denotes the discriminator's prediction about **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:

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

- Discriminator D_ϕ wants to: Output 1 for real data/ Output 0 for fake data/So it maximizes the objective
- Generator G_θ wants to: Fool the discriminator/ Make $D_\phi(G_\theta(z)) \approx 1$/So it minimizes the objective

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:

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

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

$$\begin{aligned} V(G, D) &= \int_x p_{\text{data}}(x) \log D(x)\, dx + \int_z p_z(z) \log(1 - D(g(z)))\, dz \\ &= \int_x p_{\text{data}}(x) \log D(x) + p_G(x) \log(1 - D(x))\, dx \end{aligned} \tag{3}$$

For any $(a, b) \in \mathbb{R}^2 \setminus \{(0, 0)\}$, the function $y \mapsto a \log(y) + b \log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $\text{Supp}(p_{\text{data}}) \cup \text{Supp}(p_G)$, concluding the proof. □

- 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:

$$
\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log D_G^*(x)\right] + \mathbb{E}_{z \sim p_z} \left[\log \left(1 - D_G^*(G(z))\right)\right] \\
&= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log D_G^*(x)\right] + \mathbb{E}_{x \sim p_G} \left[\log \left(1 - D_G^*(x)\right)\right] \\
&= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \left(\frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}\right)\right] + \mathbb{E}_{x \sim p_G} \left[\log \left(\frac{p_G(x)}{p_{\text{data}}(x) + p_G(x)}\right)\right]
\end{aligned}
$$

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

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

Hence, the optimal generator for the GAN objective becomes $p_G = p_{\text{data}}$ since it is solving

$$
\min_\theta D_{\text{JSD}}(p_{\text{data}}, p_{G_\theta}).
$$

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:

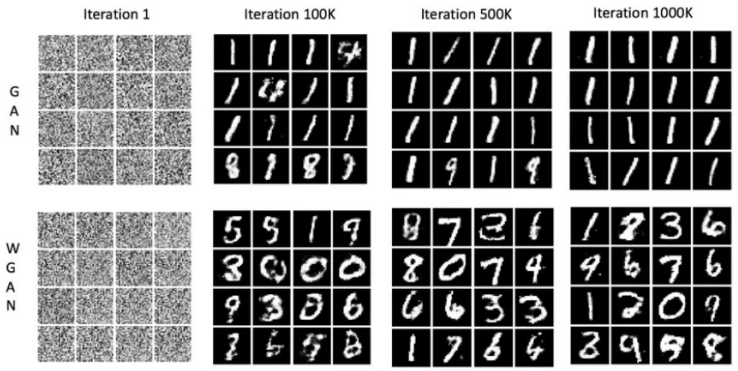
$$\nabla_\theta V(G_\theta, D_\phi) = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \log \left(1 - D_\phi(G_\theta(\mathbf{z}^{(i)}))\right)$$

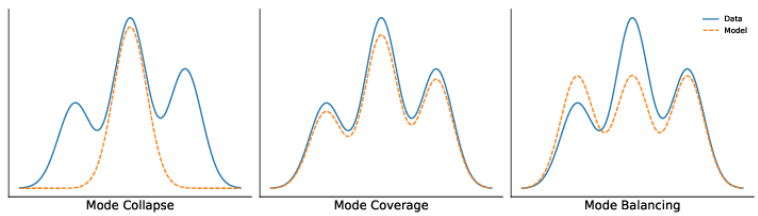
4. Take a gradient ascent step on the discriminator parameters ϕ:

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

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)], Sinkhorn divergence Gans[Genevay et al. (2018)] ...

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*.

Genevay, A., Peyré, G., and Cuturi, M. (2018). Learning generative models with sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics*, pages 1608–1617. PMLR.

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.