High-Dimensional Distribution Learning with Generative Models

Instructor: Lei Wu [1]

Mathematical Introduction to Machine Learning

Peking University, Fall 2024

[1]School of Mathematical Sciences; Center for Machine Learning Research

Example: Generate natural images

Figure 1: images generated by BigGAN (Brock et al., 2018)

.

Example: Style transfer

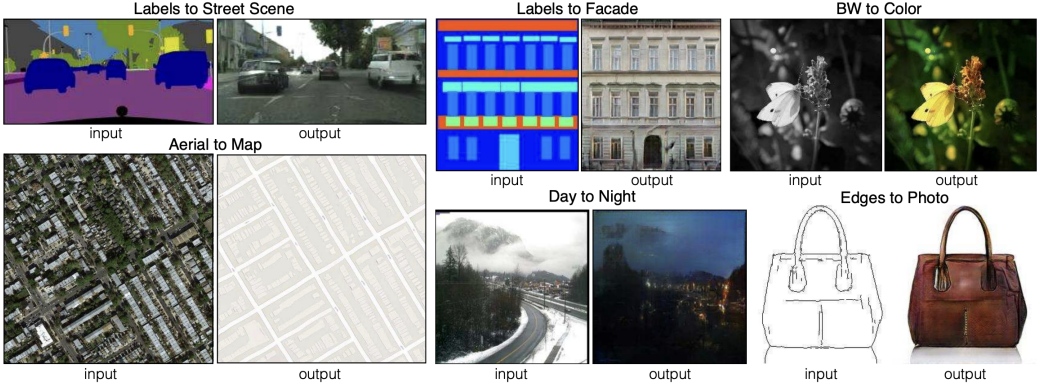


Figure 2: Style transfers with pix2pix (Isola, et. al., 2017)

Example: Generate images from text description

Try this https://huggingface.co/spaces/stabilityai/stable-diffusion.

Example: Generate images from text description

Figure 3: Generated by https://beta.dreamstudio.ai with the prompt "Great wall in mountains, stars, Vincent van Gogh".

Distribution Learning

General goal: Given $\{x_i\}_{i=1}^n$ drawn from unknown ρ^*, "estimate" ρ^* using these samples.

Task:

- Estimate the likelihood (**classical**, density estimation).
- Generate new samples (**generative model**).

Figure 4: "What I cannot create, I do not understand!"

Distribution Learning

General goal: Given $\{x_i\}_{i=1}^n$ drawn from unknown ρ^*, "estimate" ρ^* using these samples.

Task:

- Estimate the likelihood (**classical**, density estimation).
- Generate new samples (**generative model**).

Mathematical problem: How can we efficiently model high-dimensional probability distribution (including parametrization and learning)?

Figure 4: "What I cannot create, I do not understand!"

History

- Gaussian mixture model, histogram estimator, and kernel density estimator. All these models take the following basis-expansion form:

$$\hat{f}_h(x; a, \mu) = \sum_{i=1}^{m} a_j k_h(x, \mu_i),$$

  with $\sum_{j=1}^{m} a_j = 1$ and $a_j \geq 0,\ \forall j = 1, \ldots, m$. Here h denotes the "bandwidth".
- In modern ML tasks, ρ^* is a high-dimensional distribution.

- Classical linear methods all suffer from the curse of dimensionality in representing ρ^*.

How do we represent distribution?

- **Represent prob. distributions through functions**.

Energy-based models

Density function: Let $V_\theta : \mathcal{X} \mapsto \mathbb{R}$ be a parametric potential energy function. Then, the Gibbs distribution:

$$p_\theta(x) = \frac{e^{-V_\theta(x)}}{\int e^{-V_\theta(x)} dx} = e^{-V_\theta(x)} / Z_\theta.$$

is a density function.

Energy-based models

Density function: Let $V_\theta : \mathcal{X} \mapsto \mathbb{R}$ be a parametric potential energy function. Then, the Gibbs distribution:

$$p_\theta(x) = \frac{e^{-V_\theta(x)}}{\int e^{-V_\theta(x)} dx} = e^{-V_\theta(x)} / Z_\theta.$$

is a density function.

- Learning a distribution is reduced to learn a energy function V_θ. Hence, It is often referred as an energy-based model (http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf).
- $Z_\theta = \int e^{-V_\theta(x)} \, \mathrm{d}x$ is the called *partition function*. We usually are unable to evaluate the density $p_\theta(x)$ since Z_θ is hard to compute.
- We can sample p_θ with MCMC sampler but this might be not efficient.

Transform-based models

Pushforward distribution: Let $Z \sim Q$ be a simple distribution, e.g., $Q = \mathcal{N}(0, I_D)$ and $Q = \mathsf{Unif}([0,1]^D)$. Let $G : \mathbb{R}^D \mapsto \mathbb{R}^d$ be a transform (also called **generator**). Then, the distribution P is generated through the transform G:

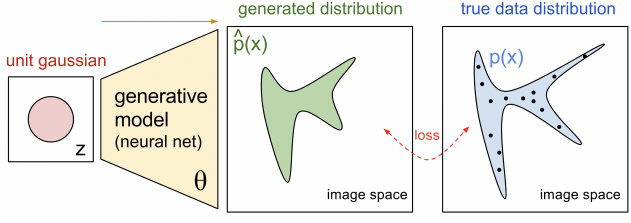
$$P = \mathsf{Law}(X),\ X = G_\theta(Z),\ Z \sim Q.$$

Transform-based models

Pushforward distribution: Let $Z \sim Q$ be a simple distribution, e.g., $Q = \mathcal{N}(0, I_D)$ and $Q = \text{Unif}([0,1]^D)$. Let $G : \mathbb{R}^D \mapsto \mathbb{R}^d$ be a transform (also called **generator**). Then, the distribution P is generated through the transform G:

$$P = \text{Law}(X),\ X = G_\theta(Z),\ Z \sim Q.$$

- In this modeling, the complex distribution P is generated from a simple distribution Q. Learning P is reduced to learn a generator G.



- We can use neural networks to parameterize G.

Transform-based models (cont'd)

- One can choose $D \ll d$. In such a case, P is a singular distribution without a density function. In particular, P concentrates on a D-dimensional sub-manifold in $\mathbb{R}^d$:

$$\mathsf{Supp}(P) = \mathsf{Range}(G).$$

Transform-based models (cont'd)

- One can choose $D \ll d$. In such a case, P is a singular distribution without a density function. In particular, P concentrates on a D-dimensional sub-manifold in $\mathbb{R}^d$:

$$\mathsf{Supp}(P) = \mathsf{Range}(G).$$

- [Pros] It is fast to draw samples from P.
  - Draw $z_1, \ldots, z_n$ independently from Q.
  - Then, $\{x_i = G(z_i)\}_{i=1}^n$ are i.i.d. samples from P.

Transform-based models (cont'd)

- One can choose $D \ll d$. In such a case, P is a singular distribution without a density function. In particular, P concentrates on a D-dimensional sub-manifold in $\mathbb{R}^d$:

$$\mathsf{Supp}(P) = \mathsf{Range}(G).$$

- [Pros] It is fast to draw samples from P.
  - Draw $z_1, \ldots, z_n$ independently from Q.
  - Then, $\{x_i = G(z_i)\}_{i=1}^n$ are i.i.d. samples from P.
- Computing expectation:

$$\mathbb{E}_{X \sim P}[f(X)] = \mathbb{E}_{Z \sim Q}[f(G(Z))] \approx \frac{1}{n} \sum_{i=1}^{n} f(G(z_i)).$$

The density of transform-based models

- By abuse of notation, let $Q(\cdot)$ denote the density function of Q. When $d = D$, X has the following density function:

$$P(x) = Q(G^{-1}(x))|\det(\nabla G^{-1}(x))|.$$

The density of transform-based models

- By abuse of notation, let $Q(\cdot)$ denote the density function of Q. When $d = D$, X has the following density function:

$$P(x) = Q(G^{-1}(x))|\det(\nabla G^{-1}(x))|.$$

- **A simple derivation:** For any testing function h, we have

$$\begin{aligned}\int h(x)P(x)\,\mathrm{d}x &= \int h(G(z))Q(z)\,\mathrm{d}z \qquad (\textbf{definition of push-forward distribution})\\ &= \int h(y)Q(G^{-1}(y))\frac{\mathrm{d}z}{\mathrm{d}y}\,\mathrm{d}y \qquad (\textbf{change of variable})\\ &= \int h(y)Q(G^{-1}(y))|\det\nabla G^{-1}(y)|\,\mathrm{d}y.\end{aligned}$$

  Hence, $P(x) = Q(G^{-1}(x))|\det\nabla G^{-1}(x)|$.

- This formula is useful when we would like to estimate likelihoods, or train the model via MLE.

How can we construct G such that

- The $G^{-1}(x)$ and $\det \nabla G^{-1}(x)$ can be computed **efficiently**.

The flow-based models provide a principled approach to design this kind of G!

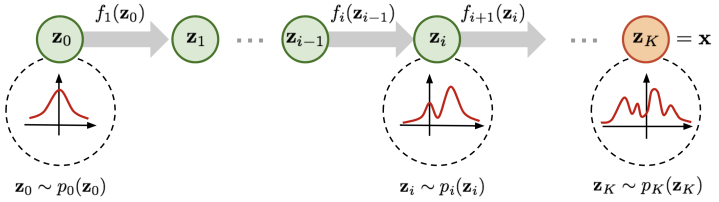
Flow-based models

Suppose $f_w : \mathbb{R}^d \mapsto \mathbb{R}^d$ be a simple invertible map. Flow-based models construct complex transforms through a "flow" of simple transform f_w by

$$G_\theta = f_{w_K} \circ f_{w_{K-1}} \circ \cdots \circ f_{w_1} : \mathbb{R}^d \mapsto \mathbb{R}^d,$$

where $\theta = (w_1, \ldots, w_K)$. It can be rewritten as

$$\begin{aligned} z_0 &= z \\ z_t &= f_{w_t}(z_{t-1}), \quad t = 1, \ldots, K \\ G_\theta(z) &= z_K \end{aligned}$$



Flow-based models (cont'd)

- The inverse is computed with $G_\theta^{-1}(x) = f_{w_1}^{-1} \circ f_{w_2}^{-1} \circ \cdots \circ f_{w_K}^{-1}(x)$.

Flow-based models (cont'd)

- The inverse is computed with $G_\theta^{-1}(x) = f_{w_1}^{-1} \circ f_{w_2}^{-1} \circ \cdots \circ f_{w_K}^{-1}(x)$.
- Note that $|\det \nabla G_\theta^{-1}(x)| = 1/|\det \nabla G_\theta(z)|$ and

$$\det \nabla G_\theta(z) = \left(\det \frac{\mathrm{d}z_L}{\mathrm{d}z_{L-1}}\right)\left(\det \frac{\mathrm{d}z_L}{\mathrm{d}z_{L-1}}\right)\cdots\left(\det \frac{\mathrm{d}z_1}{\mathrm{d}z_0}\right).$$

Hence,

$$\log|\det \nabla_z G_\theta(z)| = \sum_{t=0}^{L-1} \log|\nabla f_{w_t}(z_{t-1})|$$

Flow-based models (cont'd)

- The inverse is computed with $G_\theta^{-1}(x) = f_{w_1}^{-1} \circ f_{w_2}^{-1} \circ \cdots \circ f_{w_K}^{-1}(x)$.
- Note that $|\det \nabla G_\theta^{-1}(x)| = 1/|\det \nabla G_\theta(z)|$ and

$$\det \nabla G_\theta(z) = \left(\det \frac{\mathrm{d}z_L}{\mathrm{d}z_{L-1}}\right)\left(\det \frac{\mathrm{d}z_L}{\mathrm{d}z_{L-1}}\right)\cdots\left(\det \frac{\mathrm{d}z_1}{\mathrm{d}z_0}\right).$$

  Hence,

$$\log|\det \nabla_z G_\theta(z)| = \sum_{t=0}^{L-1} \log|\nabla f_{w_t}(z_{t-1})|$$

- Note that the computation cost of $\det \nabla f_w(z)$ can as slow as $O(d^3)$. We need to design f_w such that its inverse and the determinant of Jacobian can be efficiently computed.

Variants of flow-based models

We choose f_w through the following criteria:

- It is easy to compute the inverse map: f_w^{-1}.
- It is efficient to compute the determinant: $|\det \nabla f_w|$.

Variants of flow-based models

We choose f_w through the following criteria:

- It is easy to compute the inverse map: f_w^{-1}.
- It is efficient to compute the determinant: $|\det \nabla f_w|$.

In the literature, there are many choices:

- Normalizing flow (Tabak & Vanden-Eijnden, 2010)
- **NICE:** nonlinear independent components estimation (Dinh et al., 2014)
- **Real-NVP**: real-valued Non-volume preserving (Dinh et al., 2017)
- Masked autogressive flow (Papamakarios et al., 2017)
- Inverse autogressive flow (Kingma et al., 2016)
- **Continuous normalizing flow (CNF)** (Chen et al., 2019).
- **Diffusion model**.

Variants of flow-based models

We choose f_w through the following criteria:

- It is easy to compute the inverse map: f_w^{-1}.
- It is efficient to compute the determinant: $|\det \nabla f_w|$.

We will only cover details of NICE and real-NVP, whose designing principle is to ensure:

$$\nabla f_w \textbf{ is lower triangular}.$$

In this case, the computational cost is $O(d)$.

NICE (Non-linear Independent Component Estimation)

Decompose z into two disjoint subsets: $z = (z_{1:s}, z_{s+1:d})$. Then, NICE proposes the following **additive coupling transform** $x = f(z)$:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} + v(z_{1:s})$$

Here $v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ can be parameterized with neural networks.

NICE (Non-linear Independent Component Estimation)

Decompose z into two disjoint subsets: $z = (z_{1:s}, z_{s+1:d})$. Then, NICE proposes the following **additive coupling transform** $x = f(z)$:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} + v(z_{1:s})$$

Here $v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ can be parameterized with neural networks.

- **The inverse map:** $z = f^{-1}(x)$ is given by

$$z_{1:s} = x_{1:s}$$
$$z_{s+1:d} = x_{s+1:d} - v(x_{1:s}).$$

NICE (Non-linear Independent Component Estimation)

Decompose z into two disjoint subsets: $z = (z_{1:s}, z_{s+1:d})$. Then, NICE proposes the following **additive coupling transform** $x = f(z)$:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} + v(z_{1:s})$$

Here $v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ can be parameterized with neural networks.

- **The inverse map:** $z = f^{-1}(x)$ is given by

$$z_{1:s} = x_{1:s}$$
$$z_{s+1:d} = x_{s+1:d} - v(x_{1:s}).$$

- **The Jacobian** is lower triangular:

$$\nabla_z f_w(z) = \begin{pmatrix} I_s & 0 \\ \nabla v & I_{n-s}. \end{pmatrix}$$

Hence, $|\det \nabla_z f_w(z)| = 1$, i.e., the NICE implements volume-preserving transforms.

NICE (Non-linear Independent Component Estimation)

Decompose z into two disjoint subsets: $z = (z_{1:s}, z_{s+1:d})$. Then, NICE proposes the following **additive coupling transform** $x = f(z)$:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} + v(z_{1:s})$$

Here $v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ can be parameterized with neural networks.

- **The inverse map:** $z = f^{-1}(x)$ is given by

$$z_{1:s} = x_{1:s}$$
$$z_{s+1:d} = x_{s+1:d} - v(x_{1:s}).$$

- **The Jacobian** is lower triangular:

$$\nabla_z f_w(z) = \begin{pmatrix} I_s & 0 \\ \nabla v & I_{n-s}. \end{pmatrix}$$

  Hence, $|\det \nabla_z f_w(z)| = 1$, i.e., the NICE implements volume-preserving transforms.

- We do not need to compute the determinant of Jacobian for NICE. Great!! But the volume-preserving property also restricts the expressive power.

Real-NVP (Real-valued Non-Volume Preserving)

Real-NVP adds scaling factors to NICE:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} \odot e^{u(z_{1:s})} + v(z_{1:s}),$$

where $u, v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ are parameterized with neural networks. Here, $\odot$ and e^z should be understood in an element-wise manner.

Real-NVP (Real-valued Non-Volume Preserving)

Real-NVP adds scaling factors to NICE:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} \odot e^{u(z_{1:s})} + v(z_{1:s}),$$

where $u, v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ are parameterized with neural networks. Here, $\odot$ and e^z should be understood in an element-wise manner.

- **The inverse map:**

$$z_{1:s} = x_{1:s}$$
$$z_{s+1:d} = (x_{s+1:d} - v(x_{1:s})) \odot e^{-u(x_{1:s})}.$$

Real-NVP (Real-valued Non-Volume Preserving)

Real-NVP adds scaling factors to NICE:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} \odot e^{u(z_{1:s})} + v(z_{1:s}),$$

where $u, v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ are parameterized with neural networks. Here, $\odot$ and e^z should be understood in an element-wise manner.

- **The inverse map:**

$$z_{1:s} = x_{1:s}$$
$$z_{s+1:d} = (x_{s+1:d} - v(x_{1:s})) \odot e^{-u(x_{1:s})}.$$

- **The Jacobian**:

$$\nabla f(z) = \begin{pmatrix} I_s & 0 \\ * & \text{diag}(e^{u(z_{1:s})}). \end{pmatrix}$$
$$\log |\det \nabla f(z)| = |\sum_{j=1}^{d-s} u_j(z_{1:s})|.$$

The computation cost is $O(d)$.

Real-NVP (Real-valued Non-Volume Preserving)

Real-NVP adds scaling factors to NICE:

$$x_{1:s} = z_{1:s}$$
$$x_{s+1:d} = z_{s+1:d} \odot e^{u(z_{1:s})} + v(z_{1:s}),$$

where $u, v : \mathbb{R}^s \mapsto \mathbb{R}^{d-s}$ are parameterized with neural networks. Here, $\odot$ and e^z should be understood in an element-wise manner.

- **The inverse map:**

$$z_{1:s} = x_{1:s}$$
$$z_{s+1:d} = (x_{s+1:d} - v(x_{1:s})) \odot e^{-u(x_{1:s})}.$$

- **The Jacobian**:

$$\nabla f(z) = \begin{pmatrix} I_s & 0 \\ * & \text{diag}(e^{u(z_{1:s})}). \end{pmatrix}$$
$$\log |\det \nabla f(z)| = |\sum_{j=1}^{d-s} u_j(z_{1:s})|.$$

  The computation cost is $O(d)$.
- Real-NVP is not volume-preserving.

Remarks

- Note that the additive coupling transform leaves part of its input unchanged. To fix this issue, we need to exchange the role of two subsets for different steps.

The choice of loss function

Let $L(\cdot, \cdot)$ be a metric measuring the difference between two distributions. We expect

$$\min_{\theta} L(P_\theta, P^*). \tag{1}$$

The choice of loss function

Let $L(\cdot, \cdot)$ be a metric measuring the difference between two distributions. We expect

$$\min_{\theta} L(P_\theta, P^*). \tag{1}$$

But, we can only

$$\min_{\theta} L(P_\theta, \hat{P}_n) + \lambda_n R(\theta), \tag{2}$$

where $\hat{P}_n = \frac{1}{n} \sum_{i=1}^{n} \delta(\cdot - x_i)$ and $R(\cdot)$ denotes certain regularization.

The choice of loss function

Let $L(\cdot, \cdot)$ be a metric measuring the difference between two distributions. We expect

$$\min_{\theta} L(P_\theta, P^*). \tag{1}$$

But, we can only

$$\min_{\theta} L(P_\theta, \hat{P}_n) + \lambda_n R(\theta), \tag{2}$$

where $\hat{P}_n = \frac{1}{n} \sum_{i=1}^{n} \delta(\cdot - x_i)$ and $R(\cdot)$ denotes certain regularization.

- Different from supervised learning, choosing $L(\cdot, \cdot)$ is highly non-trivial. **There are no such thing called fitting error at the i-th sample**.

The choice of loss function

Let $L(\cdot, \cdot)$ be a metric measuring the difference between two distributions. We expect

$$\min_{\theta} L(P_\theta, P^*). \tag{1}$$

But, we can only

$$\min_{\theta} L(P_\theta, \hat{P}_n) + \lambda_n R(\theta), \tag{2}$$

where $\hat{P}_n = \frac{1}{n} \sum_{i=1}^{n} \delta(\cdot - x_i)$ and $R(\cdot)$ denotes certain regularization.

- Different from supervised learning, choosing $L(\cdot, \cdot)$ is highly non-trivial. **There are no such thing called fitting error at the i-th sample**.
- There are many variants of norm, divergence, distance for comparing two distributions:
  - P_θ may not have a density function, e.g., the transform-based models.
  - Computing the density of P_θ may be intractable or expensive, e.g., the energy-based models.

- **Designing loss functions**

What is a practical loss function?

- Consider the L^p distance:

$$\int |P_\theta(x) - \hat{P}_n(x)|^p \, \mathrm{d}x.$$

- The total variation:

$$TV(P_\theta, \hat{P}_n).$$

What is a practical loss function?

- Consider the L^p distance:
$$\int |P_\theta(x) - \hat{P}_n(x)|^p \, \mathrm{d}x.$$
- The total variation:
$$TV(P_\theta, \hat{P}_n).$$

We are unable to evaluate these losses since we only have samples $x_1, \ldots, x_n$ from ρ^*.

A pratical choice of metric must be an expectation in $\hat{P}_n$; otherwise, the metric is not computable.

Strong form

- Strong form: Need P_θ to have a density function.

$$\begin{aligned} \mathrm{KL}(\hat{P}_n || P_\theta) &= \int \log \frac{\hat{P}_n(x)}{P_\theta(x)} \, \mathrm{d}\hat{P}_n(x) \\ &= constant - \mathbb{E}_{\hat{P}_n}[\log P_\theta(x)] \\ &= constant - \frac{1}{n} \sum_{i=1}^{n} \log P_\theta(x_i) \end{aligned} \tag{3}$$

- It is equivalent to maximizing the likelihood.
- In fact, (3) is the only practical density-based loss (homework).

Weak form

Weak Form: View P as a linear functional over certain function classes.

$$L(P, P') = \sup_{f \in \mathcal{F}} \left(\mathbb{E}_P[f] - \mathbb{E}_{P'}[f] \right) \tag{4}$$

Here f is called the test function and $\mathcal{F}$ is the set of test functions.

Weak form

Weak Form: View P as a linear functional over certain function classes.

$$L(P, P') = \sup_{f \in \mathcal{F}} \left(\mathbb{E}_P[f] - \mathbb{E}_{P'}[f] \right) \tag{4}$$

Here f is called the test function and $\mathcal{F}$ is the set of test functions.
Intuitively speaking, weak metrics measure the differences of two distributions by comparing their "generalized" moments .

Weak form

Weak Form: View P as a linear functional over certain function classes.

$$L(P, P') = \sup_{f \in \mathcal{F}} \left(\mathbb{E}_P[f] - \mathbb{E}_{P'}[f] \right) \tag{4}$$

Here f is called the test function and $\mathcal{F}$ is the set of test functions.
Intuitively speaking, weak metrics measure the differences of two distributions by comparing their "generalized" moments .

There are many different choices of moments class.

- $\mathcal{F} = \{x, x^2, x^3, \dots, \} \rightarrow$ the classic moment methods.
- $\mathcal{F} = \{\|f\|_{L^\infty} \leq 1\} \rightarrow$ the total variation norm.
- $\mathcal{F} = \{\|f\|_{\text{Lip}} \leq 1\} \rightarrow$ the 1-Wasserstein metric.
- $\mathcal{F} =$ unit ball in RKHS space $\rightarrow$ the maximum mean discrepancy distance.
- $\mathcal{F} =$ neural networks (with certain constraints) $\rightarrow$ the neural distance.

The models

Loss functions:

- Strong: log-likelihood
$$\min_P -\mathbb{E}_{P^*}[\log P(x)].$$
- Weak: dual norm
$$\min_P \max_{f \in \mathcal{F}} \left(\mathbb{E}_P[f] - \mathbb{E}_{P^*}[f]\right).$$

Representations:

- Generator/Pushforward: $P = G\#Q$.
- Potential/Gibbs: $P = e^{-V} / \int e^{-V} \, \mathrm{d}Q$.

The models

Loss functions:

- Strong: log-likelihood
$$\min_P -\mathbb{E}_{P^*}[\log P(x)].$$
- Weak: dual norm
$$\min_P \max_{f \in \mathcal{F}} \left(\mathbb{E}_P[f] - \mathbb{E}_{P^*}[f]\right).$$

Representations:

- Generator/Pushforward: $P = G\#Q$.
- Potential/Gibbs: $P = e^{-V} / \int e^{-V} \, \mathrm{d}Q$.

Combinations: Different combinations lead to diffent models.

- Weak metric + generator = GAN (Generative adversarial network):
- Strong metric: Variational autoencoder (VAE), normalizing flow, diffusion-based generative model, autoregressive models, etc.

Generative adversarial network (GAN)

Rename the test function as the discriminator D.

- Weak formulation of Jensen-Shannon divergence (symmetrized KL):

$$\begin{aligned}\mathrm{JS}(P, P') &= \frac{1}{2} D_{KL}\left(P \middle|\middle| \frac{P+P'}{2}\right) + \frac{1}{2} D_{KL}\left(P' \middle|\middle| \frac{P+P'}{2}\right) \\ &= \sup_q \left(\mathbb{E}_P[\log q(x)] + \mathbb{E}_{P'}[\log(1-q(x))]\right)\end{aligned}$$

where the supremum is taken with all measurable functions $q : \mathbb{R}^d \mapsto [0, 1]$.

Generative adversarial network (GAN)

Rename the test function as the discriminator D.

- Weak formulation of Jensen-Shannon divergence (symmetrized KL):

$$\begin{aligned}\mathrm{JS}(P, P') &= \frac{1}{2} D_{KL}\left(P \middle|\middle| \frac{P+P'}{2}\right) + \frac{1}{2} D_{KL}\left(P' \middle|\middle| \frac{P+P'}{2}\right) \\ &= \sup_q \left(\mathbb{E}_P[\log q(x)] + \mathbb{E}_{P'}[\log(1-q(x))]\right)\end{aligned}$$

  where the supremum is taken with all measurable functions $q : \mathbb{R}^d \mapsto [0, 1]$.

- Neural network formulation:

$$L(P, P') = \sup_D \left(\mathbb{E}_P[\log(1 - D(x))] + \mathbb{E}_{P'}[\log D(x)]\right),$$

  where $D : \mathbb{R}^d \mapsto (0, 1)$ is a neural network. It is essentially a binary classifier.

Generative adversarial network (GAN)

Rename the test function as the discriminator D.

- Weak formulation of Jensen-Shannon divergence (symmetrized KL):

$$\begin{aligned}\mathrm{JS}(P, P') &= \frac{1}{2} D_{KL}\left(P \middle|\middle| \frac{P+P'}{2}\right) + \frac{1}{2} D_{KL}\left(P' \middle|\middle| \frac{P+P'}{2}\right) \\ &= \sup_q \left(\mathbb{E}_P[\log q(x)] + \mathbb{E}_{P'}[\log(1-q(x))]\right)\end{aligned}$$

  where the supremum is taken with all measurable functions $q : \mathbb{R}^d \mapsto [0, 1]$.

- Neural network formulation:

$$L(P, P') = \sup_D \left(\mathbb{E}_P[\log(1 - D(x))] + \mathbb{E}_{P'}[\log D(x)]\right),$$

  where $D : \mathbb{R}^d \mapsto (0, 1)$ is a neural network. It is essentially a binary classifier.

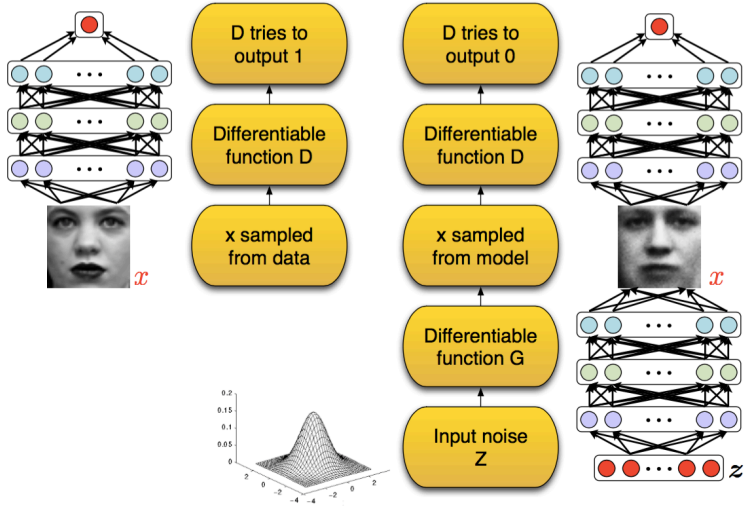
- Consider the generative model $P = G\#Q$. Then, the problem becomes a minimax problem:

$$\min_G \max_D \left(\mathbb{E}_{z\sim Q}[\log(1 - D(G(z)))] + \frac{1}{n}\sum_{i=1}^{n} \log D(x_i)\right).$$

GAN: The original game motivation

A game between the generator and discriminator:

- Discriminator: Distinguish the fake and real data.
- Generator: generate fake data $G(z)$ such that $\{G(z)\}_z$ are undistinguishable with the real data $\{x_i\}_i$.



Wasserstein GAN

Choose test functions as constraint neural networks.

$$\min_{\theta_1 \in U} \max_{\theta_2} \left(\mathbb{E}_z[f_{\theta_1}(G_{\theta_2}(z))] - \frac{1}{n} \sum_{i=1}^{n} f_{\theta_1}(x_i) \right) \tag{5}$$

- Both f_{θ_1} and G_{θ_2} are neural networks.
- In the original Wasserstein GAN, $U = \{\theta : \max_i |\theta_i| \leq \delta\}$ with the δ tunned for each problems.
- There are many other choices of U, such as gradient penalty, spectral normalization, etc.

Evaluate our models

In supervised learning, we evaluate our model by using a test dataset. However, for unsupervised learning models, it is hard to evaluate the model's goodness.

Evaluate our models

In supervised learning, we evaluate our model by using a test dataset. However, for unsupervised learning models, it is hard to evaluate the model's goodness.

Weak metrics:

- Human judgement.
- Wasserstein W_2 metric: Computation suffers from the curse of dimensionality.
- Frechet inception distance: Approximating W_2 with only means and covariances.

Evaluate our models

In supervised learning, we evaluate our model by using a test dataset. However, for unsupervised learning models, it is hard to evaluate the model's goodness.

Weak metrics:

- Human judgement.
- Wasserstein W_2 metric: Computation suffers from the curse of dimensionality.
- Frechet inception distance: Approximating W_2 with only means and covariances.

Strong metrics:

- Log-likelihood.
- Inception score: Let $C(x)$ be an ImageNet classifier. If $C(x)$ has small entropy on x, then the classifier is confident about the label of x. This implies that x looks like an image (at least for $C(x)$).

Training procedure

- Strong form:

$$\min_{\theta} -\sum_{i} \log p_{\theta}(x_i)$$

  Train with SGD/ADAM.

- Weak form:

$$\min_{\theta_2} \max_{\theta_1} \left(\mathbb{E}_{x \sim \hat{P}_n}[D(x; \theta_1)] - \mathbb{E}_z[D(G(z; \theta_2); \theta_1)] \right)$$

  This is not a standard optimization but a minimax problem.

Solve the minimax problem

$$\min_{\theta_2} \max_{\theta_1} \Big(\mathbb{E}_{x \sim \hat{P}_n} [D(x; \theta_1)] - \mathbb{E}_z [D(G(z; \theta_2); \theta_1)] \Big)$$

Each step $(\theta_1(t), \theta_2(t)) \mapsto (\theta_1(t+1), \theta_2(t+1))$ updates as follows.

Solve the minimax problem

$$\min_{\theta_2} \max_{\theta_1} \Big(\mathbb{E}_{x \sim \hat{P}_n}[D(x; \theta_1)] - \mathbb{E}_z[D(G(z; \theta_2); \theta_1)] \Big)$$

Each step $(\theta_1(t), \theta_2(t)) \mapsto (\theta_1(t+1), \theta_2(t+1))$ updates as follows.

- **Maximization-step:** Let $\theta_1(t, 0) = \theta_1(t)$.

Solve the minimax problem

$$\min_{\theta_2} \max_{\theta_1} \left(\mathbb{E}_{x \sim \hat{P}_n}[D(x; \theta_1)] - \mathbb{E}_z[D(G(z; \theta_2); \theta_1)] \right)$$

Each step $(\theta_1(t), \theta_2(t)) \mapsto (\theta_1(t+1), \theta_2(t+1))$ updates as follows.

- **Maximization-step:** Let $\theta_1(t, 0) = \theta_1(t)$.
  - For $k = 1, \ldots, m$, randomly sample $z_{1,k}, \ldots, z_{B,k}$ and update the generator as follows

$$\theta_1(t, k) = \theta_1(t, k-1) + \eta_1 \nabla_{\theta_1} \left(\frac{1}{B_1} \sum_i D(x_i; \theta_1(t)) - \frac{1}{B_2} \sum_j D(G(z_j; \theta_2(t)); \theta_1(t, k-1)) \right)$$

Solve the minimax problem

$$\min_{\theta_2}\max_{\theta_1}\left(\mathbb{E}_{x\sim\hat{P}_n}[D(x;\theta_1)]-\mathbb{E}_z[D(G(z;\theta_2);\theta_1)]\right)$$

Each step $(\theta_1(t),\theta_2(t))\mapsto(\theta_1(t+1),\theta_2(t+1))$ updates as follows.

- **Maximization-step:** Let $\theta_1(t,0)=\theta_1(t)$.
  - For $k=1,\ldots,m$, randomly sample $z_{1,k},\ldots,z_{B,k}$ and update the generator as follows

$$\theta_1(t,k)=\theta_1(t,k-1)+\eta_1\nabla_{\theta_1}\left(\frac{1}{B_1}\sum_i D(x_i;\theta_1(t))-\frac{1}{B_2}\sum_j D(G(z_j;\theta_2(t));\theta_1(t,k-1))\right)$$

  - Return $\theta_1(t+1)=\theta_1(t,m)$.

Solve the minimax problem

$$\min_{\theta_2} \max_{\theta_1} \left(\mathbb{E}_{x \sim \hat{P}_n}[D(x; \theta_1)] - \mathbb{E}_z[D(G(z; \theta_2); \theta_1)] \right)$$

Each step $(\theta_1(t), \theta_2(t)) \mapsto (\theta_1(t+1), \theta_2(t+1))$ updates as follows.

- **Maximization-step:** Let $\theta_1(t, 0) = \theta_1(t)$.
  - For $k = 1, \ldots, m$, randomly sample $z_{1,k}, \ldots, z_{B,k}$ and update the generator as follows

$$\theta_1(t, k) = \theta_1(t, k-1) + \eta_1 \nabla_{\theta_1} \left(\frac{1}{B_1} \sum_i D(x_i; \theta_1(t)) - \frac{1}{B_2} \sum_j D(G(z_j; \theta_2(t)); \theta_1(t, k-1)) \right)$$

  - Return $\theta_1(t+1) = \theta_1(t, m)$.
- **Minimization-step**: Update the discriminator:

$$\theta_2(t+1) = \theta_2(t) - \eta_2 \nabla_{\theta_2} \left(\frac{1}{B} \sum_j D(G(z_{j,k}; \theta_2(t, k)); \theta_1(t)) \right),$$

where $\{x_i\}$ and $\{z_j\}$ are the minibatch samples.

Issues

- **The training of weak models is very unstable**, in particular when the maximization step is updated only a few steps—a choice preferred in practice.
  Moreover, we do not have a good criterion to monitor the training progress since the weak norm cannot be estimated in a reasonable way.

Issues

- **The training of weak models is very unstable**, in particular when the maximization step is updated only a few steps—a choice preferred in practice.
  Moreover, we do not have a good criterion to monitor the training progress since the weak norm cannot be estimated in a reasonable way.
- **Mode collapse**: Are there metrics that can detect the mode collapse?

Figure 5: Left: Images from [Zhao et al., 2017] Energy-based GAN

Summary

Distribution learning: Normalizing flow, GAN, etc.

- Representation:
  - Energy-based models
  - Transform-based models: flow-based models (NICE, real-NVP, etc.)
- Loss designing:
  - Strong form: MLE/KL-divergence;
  - Weak form: The choice of test functions.
- Evaluation: Weak and strong metrics.

Note: Variational Autoencoders (VAEs) are important generative models but are not covered in this slide. Additionally, we will dedicate a separate lecture to discussing diffusion models.

Questions to Aid Understanding

- What are the advantages and disadvantages of weak models?
- What are the advantages and disadvantages of strong models?
- Training flow-based model is still challenging. Why?

Supplementary Wasserstein metric

- Define a distance between two sets of points $\{\mathbf{x}_i\}_{i=1}^n$ and $\{\mathbf{y}_j\}_{j=1}^n$:

$$\min_{\pi \in S_n} \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left\| \mathbf{x}_i - \mathbf{y}_{\pi(i)} \right\|^2}$$

- Generalize to probability measures P and Q : the matching becomes a joint distribution $\pi(\mathbf{x}, \mathbf{y})$

$$\Pi(P, Q) := \left\{ \pi \in \mathcal{P}\left(\mathbb{R}^d \times \mathbb{R}^d\right), \pi_{\mathbf{x}} = P, \pi_{\mathbf{y}} = Q \right\}$$

  Define the Wasserstein metric W_p

$$W_p(P, Q) := \min_{\pi \in \Pi(P,Q)} \left(\mathbb{E}_{\pi(\mathbf{x},\mathbf{y})} \left[\|\mathbf{x} - \mathbf{y}\|^p \right] \right)^{1/p}$$

- For W_1, we have the Kantorovich-Rubinstein theorem:

$$W_1(P, Q) = \sup_{\|f\|_{Lip} \leq 1} \mathbb{E}_P[f] - \mathbb{E}_Q[f]$$

  Duality holds for W_n in general, but the formula for W_1 is simplest.