Generative networks part 2: GANs
Recap on generative networks

Generative networks provide a way to **sample** from any distribution.

1. Sample $z \sim \mu$, where $\mu$ denotes an efficiently sampleable distribution (e.g., uniform or Gaussian).

2. Output $g(z)$, where $g : \mathbb{R}^d \rightarrow \mathbb{R}^m$ is a deep network.

**Notation:** let $g\#\mu$ (pushforward of $\mu$ through $g$) denote this distribution.
Recap on generative networks

Generative networks provide a way to **sample** from any distribution.

1. Sample $z \sim \mu$, where $\mu$ denotes an efficiently sampleable distribution (e.g., uniform or Gaussian).
2. Output $g(z)$, where $g : \mathbb{R}^d \rightarrow \mathbb{R}^m$ is a deep network.

**Notation:** Let $g\#\mu$ (pushforward of $\mu$ through $g$) denote this distribution.

**Brief remarks:**

- **Can this model any target distribution $\nu$?** Yes, (roughly) for the same reason that $g$ can approximate any $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$.
- **Graphical models let us sample and estimate probabilities; what about here?** Nope.
Univariate examples

\[ g(x) = x, \] the identity function, mapping \( \text{Uniform}([0,1]) \) to itself.
Univariate examples

\[ g(x) = x^2, \]

mapping Uniform([0, 1]) to something \( \propto \frac{2}{\sqrt{x}} \).
$g$ is inverse CDF of Gaussian, input distribution is Uniform([0, 1]) and output is Gaussian.
Another way to visualize generative networks

Given a sample from a distribution (even $g \neq \mu$), here’s the “kernel density” / “Parzen window” estimate of its density:

1. Start with random draw $(x_i)_{i=1}^n$.

2. “Place bumps at every $x_i$”:
   Define $\hat{p}(x) := \frac{1}{n} \sum_{i=1}^n k \left( \frac{x-x_i}{h} \right)$, where $k$ is a kernel function (not the SVM one!), $h$ is the “bandwidth”; for example:
Another way to visualize generative networks

Given a sample from a distribution (even $g \neq \mu$), here’s the “kernel density” / “Parzen window” estimate of its density:

1. Start with random draw $(x_i)_{i=1}^n$.
2. “Place bumps at every $x_i$”:
   Define $\hat{p}(x) := \frac{1}{n} \sum_{i=1}^{n} k \left( \frac{x-x_i}{h} \right)$, where $k$ is a kernel function (not the SVM one!), $h$ is the “bandwidth”; for example:
   - Gaussian: $k(z) \propto \exp \left( -\|z\|^2/2 \right)$;
   - Epanechnikov: $k(z) \propto \max\{0, 1 - \|z\|^2\}$.
Examples — univariate sampling.

Univariate sample, kernel density estimate (kde), GMM E-M.
Examples — univariate sampling.

Univariate sample, kernel density estimate (kde), GAN kde.

This is admittedly very indirect!
As mentioned, there aren’t great ways to get GAN/VAE density information.
Examples — bivariate sampling.

Bivariate sample, GMM E-M.
Examples — bivariate sampling.

Bivariate sample, kernel density estimate (kde).
Examples — bivariate sampling.

Bivariate sample, GAN kde.

**Question:** how will this plot change with network capacity?
Approaches we’ve seen for modeling distributions.

Graphical models: can be interpretable, can encode domain knowledge.

Kernel density estimation: easy to implement, converges to the right thing, suffers a curse of dimension.

Training: easy for KDE, messy for graphical models.

Interpretability: fine for both.

Sampling: easy for both.

Probability measurements: easy for KDE, sometimes easy for graphical model.

Deep networks:

Either we have easy sampling, or we can estimate densities. Doing both seems to have major computational or data costs.
Approaches we’ve seen for modeling distributions.

Let’s survey our approaches to density estimation.

- **Graphical models:**
  - can be interpretable,
  - can encode domain knowledge.

- **Kernel density estimation:**
  - easy to implement, converges to the right thing,
  - suffers a curse of dimension.

- **Training:** easy for KDE, messy for graphical models.
- **Interpretability:** fine for both.
- **Sampling:** easy for both.
- **Probability measurements:** easy for KDE, sometimes easy for graphical model.
Approaches we’ve seen for modeling distributions.

Let’s survey our approaches to density estimation.

- **Graphical models:**
  can be interpretable, 
  can encode domain knowledge.

- **Kernel density estimation:**
  easy to implement, converges to the right thing, 
  suffers a curse of dimension.

- **Training:** easy for KDE, messy for graphical models. 
  **Interpretability:** fine for both. 
  **Sampling:** easy for both. 
  **Probability measurements:** easy for KDE, sometimes easy for graphical model.

Deep networks.

- Either we have easy sampling, or we can estimate densities. 
  Doing both seems to have major computational or data costs.
Brief VAE Recap
(Variational) Autoencoders

**Autoencoder:**

\[ x_i \xrightarrow{\text{map}} \text{latent} \quad z_i = f(x_i) \xrightarrow{\text{map}} \hat{x}_i = g(z_i). \]

Objective: \[ \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, \hat{x}_i). \]
(Variational) Autoencoders

- **Autoencoder**: 
  \[ \mathbf{x}_i \xrightarrow{f_{\text{map}}} \text{latent} \quad \mathbf{z}_i = f(\mathbf{x}_i) \xrightarrow{g_{\text{map}}} \hat{\mathbf{x}}_i = g(\mathbf{z}_i). \]

  Objective: 
  \[ \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i). \]

- **Variational Autoencoder**: 
  \[ \mathbf{x}_i \xrightarrow{f_{\text{map}}} \text{latent distribution} \quad \mu_i = f(\mathbf{x}_i) \xrightarrow{g_{\text{pushforward}}} \hat{\mathbf{x}}_i \sim g\#\mu_i. \]

  Objective: 
  \[ \frac{1}{n} \sum_{i=1}^{n} \left[ \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \lambda \text{KL}(\mu, \mu_i) \right]. \]
\( \hat{x}_i \sim g\#\mu_i \)
$\hat{x}_i \sim g#\mu$ with small $\lambda$
Generative Adversarial Networks (GANs)
Generative network setup and training.

- We are given \((x_i)_{i=1}^{n} \sim \nu\).
- We want to find \(g\) so that \((g(z_i))_{i=1}^{n} \approx (x_i)_{i=1}^{n}\), where \((z_i)_{i=1}^{n} \sim \mu\).

**Problem:** this isn't as simple as fitting \(g(z_i) \approx x_i\).
Generative network setup and training.

➢ We are given $(x_i)_{i=1}^n \sim \nu$.

➢ We want to find $g$ so that $(g(z_i))_{i=1}^n \approx (x_i)_{i=1}^n$, where $(z_i)_{i=1}^n \sim \mu$.

**Problem:** this isn’t as simple as fitting $g(z_i) \approx x_i$.

**Solutions:**

➢ VAE: For each $x_i$, construct distribution $\mu_i$, so that $\hat{x}_i \sim g\#\mu_i$ and $x_i$ are close, as are $\mu_i$ and $\mu$.

To generate fresh samples, get $z \sim \mu$ and output $g(z)$.

➢ GAN: Pick a distance notion between distributions (or between samples $(g(z_i))_{i=1}^n$ and $(x_i)_{i=1}^n$) and pick $g$ to minimize that!
GAN overview

**GAN approach:** we minimize $D(\nu, g\#\mu)$ directly, where “$D$” is some notion of distance/divergence:

- **Jensen-Shannon Divergence** (original GAN paper).
- **Wasserstein distance** (influential follow-up).
GAN overview

**GAN approach:** we minimize $D(\nu, g\#\mu)$ directly, where “$D$” is some notion of distance/divergence:

- **Jensen-Shannon Divergence** (original GAN paper).
- **Wasserstein distance** (influential follow-up).

Each distance is computed with an **alternating/adversarial** scheme:

1. We have some current choice $g_t$, and use it to produce a sample $(\hat{x}_i)_{i=1}^n$ with $\hat{x}_i = g_t(z_i)$.
2. We train a **discriminator/critic** $f_t$ to find differences between $(\hat{x}_i)_{i=1}^n$ and $(x_i)_{i=1}^n$.
3. We then pick a new **generator** $g_{t+1}$, trained to fool $f_t$!
Jensen-Shannon divergence (original GAN)
Let \( p, p_g \) denote density of data and generator, \( \tilde{p} = \frac{p}{2} + \frac{p_g}{2} \).

Original GAN minimizes **Jensen-Shannon Divergence**:

\[
2 \cdot JS(p, p_g) = KL(p, \tilde{p}) + KL(p_g, \tilde{p})
\]

\[
= \int p(x) \ln \frac{p(x)}{\tilde{p}(x)} \, dx + \int p_g(x) \ln \frac{p_g(x)}{\tilde{p}(x)} \, dx
\]

\[
= \mathbb{E}_p \ln \frac{p(x)}{\tilde{p}(x)} + \mathbb{E}_{p_g} \ln \frac{p_g(x)}{\tilde{p}(x)}.
\]
Let $p, p_g$ denote density of data and generator, $\tilde{p} = \frac{p}{2} + \frac{p_g}{2}$.

Original GAN minimizes **Jensen-Shannon Divergence**:

$$2 \cdot JS(p, p_g) = KL(p, \tilde{p}) + KL(p_g, \tilde{p})$$

$$= \int p(x) \ln \frac{p(x)}{\tilde{p}(x)} \, dx + \int p_g(x) \ln \frac{p_g(x)}{\tilde{p}(x)} \, dx$$

$$= \mathbb{E}_p \ln \frac{p(x)}{\tilde{p}(x)} + \mathbb{E}_{p_g} \ln \frac{p_g(x)}{\tilde{p}(x)}.$$

But we’ve been saying we can’t write down $p_g$?
Original GAN formulation

Let \( p, p_g \) denote density of data and generator, \( \tilde{p} = \frac{p}{2} + \frac{p_g}{2} \).

Original GAN minimizes \textbf{Jensen-Shannon Divergence}:

\[
2 \cdot \text{JS}(p, p_g) = \text{KL}(p, \tilde{p}) + \text{KL}(p_g, \tilde{p})
\]

\[
= \int p(x) \ln \frac{p(x)}{\tilde{p}(x)} \, dx + \int p_g(x) \ln \frac{p_g(x)}{\tilde{p}(x)} \, dx
\]

\[
= \mathbb{E}_p \ln \frac{p(x)}{\tilde{p}(x)} + \mathbb{E}_{p_g} \ln \frac{p_g(x)}{\tilde{p}(x)}.
\]

But we’ve been saying we can’t write down \( p_g \)?

Original GAN approach applies alternating minimization to

\[
\inf_{g \in G} \sup_{f \in F \colon f : X \to (0,1)} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln \left(1 - f(g(z_j))\right) \right].
\]
Original GAN objective:

\[
\inf_{g \in G} \sup_{f : X \to (0,1)} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln (1 - f(g(z_j))) \right].
\]

Algorithm alternates these two steps:

1. Hold \( g \) fixed and optimize \( f \). Specifically, generate a sample \((\hat{x}_j)_{j=1}^{m} = (g(z_j))_{j=1}^{m}\), and approximately optimize

\[
\sup_{f : X \to (0,1)} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln (1 - f(\hat{x}_j)) \right].
\]

2. Hold \( f \) fixed and optimize \( g \). Specifically, generate \((z_j)_{j=1}^{m}\) and approximately optimize

\[
\inf_{g \in G} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln (1 - f(g(z_j))) \right].
\]
Some implementation issues

Algorithm alternates these two steps:

1. Hold $g$ fixed and optimize $f$. Specifically, generate a sample $(\hat{x}_j)_{j=1}^m = (g(z_j))_{j=1}^m$, and approximately optimize

$$\sup_{f \in \mathcal{F}} \inf_{f : X \to (0,1)} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln (1 - f(\hat{x}_j)) \right] .$$

2. Hold $f$ fixed and optimize $g$. Specifically, generate $(z_j)_{j=1}^m$ and approximately optimize

$$\inf_{g \in \mathcal{G}} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln (1 - f(g(z_j))) \right] .$$

Remarks.

- Common practice: do many $f$ ascents for each $g$ descent.
- Training has all sorts of instabilities and heuristics fixes; e.g., **mode collapse** ($g$ outputs a small subset of training elements).
- Original intuition was game-theoretic: generator and critic compete.
Optimal discriminator

Given $p$ (of data), $p_g$ (from $g$), $p_z$ (on $z$),

\[
\mathbb{E} \ln f(x) + \mathbb{E} \ln(1 - f(g(z))) \\
= \int \ln f(x)p(x) \, dx + \int \ln(1 - f(g(z)))p_z(z) \, dz \\
= \int \ln f(x)p(x) \, dx + \int \ln(1 - f(x))p_g(x) \, dx. \\
= \int \left( \ln f(x)p(x) + \ln(1 - f(x))p_g(x) \right) \, dx.
\]
Optimal discriminator

Given $p$ (of data), $p_g$ (from $g$), $p_z$ (on $z$),

$$\mathbb{E} \ln f(x) + \mathbb{E} \ln(1 - f(g(z)))$$

$$= \int \ln f(x) p(x) \, dx + \int \ln(1 - f(g(z))) p_z(z) \, dz$$

$$= \int \ln f(x) p(x) \, dx + \int \ln(1 - f(x)) p_g(x) \, dx.$$

$$= \int \left( \ln f(x) p(x) + \ln(1 - f(x)) p_g(x) \right) \, dx.$$

To find maximal $f$, maximize pointwise.
Optimal discriminator

Given \( p \) (of data), \( p_g \) (from \( g \)), \( p_z \) (on \( z \)),

\[
\mathbb{E} \ln f(x) + \mathbb{E} \ln(1 - f(g(z))) \\
= \int \ln f(x)p(x) \, dx + \int \ln(1 - f(g(z)))p_z(z) \, dz \\
= \int \ln f(x)p(x) \, dx + \int \ln(1 - f(x))p_g(x) \, dx. \\
= \int \left( \ln f(x)p(x) + \ln(1 - f(x))p_g(x) \right) \, dx.
\]

To find maximal \( f \), maximize pointwise.

\[ r \mapsto a \ln(r) + b \ln(1 - r) \text{ is concave with maximum } a/(a + b). \]
Given $p$ (of data), $p_g$ (from $g$), $p_z$ (on $z$),

$$E \ln f(x) + E \ln(1 - f(g(z)))$$

$$= \int \ln f(x)p(x) \, dx + \int \ln(1 - f(g(z)))p_z(z) \, dz$$

$$= \int \ln f(x)p(x) \, dx + \int \ln(1 - f(x))p_g(x) \, dx$$

$$= \int \left( \ln f(x)p(x) + \ln(1 - f(x))p_g(x) \right) \, dx.$$

To find maximal $f$, maximize pointwise.

$r \mapsto a \ln(r) + b \ln(1 - r)$ is concave with maximum $a/(a + b)$.

Therefore, optimal discriminator satisfies $f(x) = \frac{p(x)}{p(x) + p_g(x)}$. 
Recovering Jensen-Shannon divergence

Let’s plug in optimal discriminator \( f(\bm{x}) = \frac{p(\bm{x})}{p(\bm{x}) + p_g(\bm{x})} \):

\[
\sup_{f \in \mathcal{F}} \mathbb{E} \ln f(\bm{x}) + \mathbb{E} \ln(1 - f(g(\bm{z})))
= \sup_{f \in \mathcal{F}} \int \left( \ln f(\bm{x})p(\bm{x}) + \ln(1 - f(\bm{x}))p_g(\bm{x}) \right) d\bm{x}.
= \int \left( p(\bm{x}) \ln \frac{p(\bm{x})}{p(\bm{x}) + p_g(\bm{x})} + p_g(\bm{x}) \ln \frac{p_g(\bm{x})}{p(\bm{x}) + p_g(\bm{x})} \right) d\bm{x} - \ln 4
= \text{KL} \left( p, \frac{p + p_g}{2} \right) + \text{KL} \left( p_g, \frac{p + p_g}{2} \right) - \ln 4
= 2 \cdot \text{JS} (p, p_g) - \ln 4.
\]
This derivation is over the true distribution, not the sample! The sample induces a discrete distribution!

- How to regularize/generalize?
- The optimum of memorizing training set is trivial and doesn’t need a GAN to train (just randomly sample the training set).

We pick $f$ from a class of deep networks, and in general can’t set it to arbitrary $\frac{p}{(p+p_g)}$. So Jensen-Shannon connection is strained.

There are many refinements, including architecture choices; e.g., “DCGAN”.
Wasserstein GAN (WGAN)
Wasserstein GAN (WGAN)

Original GAN objective:

$$\inf_{g \in G} \sup_{f \in F} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln (f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln (1 - f(g(z_j))) \right].$$
Wasserstein GAN (WGAN)

Original GAN objective:

\[
\inf_{g \in \mathcal{G}} \sup_{f \in \mathcal{F} \, : \, f : X \to (0,1)} \left[ \frac{1}{n} \sum_{i=1}^{n} \ln(f(x_i)) + \frac{1}{m} \sum_{j=1}^{m} \ln \left(1 - f(g(z_j))\right) \right].
\]

Wasserstein GAN objective:

\[
\inf_{g \in \mathcal{G}} \sup_{f \in \mathcal{F}} \|f\|_{\text{Lip}} \leq 1 \left[ \frac{1}{n} \sum_{i=1}^{n} f(x_i) - \frac{1}{m} \sum_{j=1}^{m} f(g(z_j)) \right],
\]

where “\(\|f\|_{\text{Lip}} \leq 1\)” means \(f\) 1-Lipschitz (\(\|f(x) - f(y)\| \leq \|x - y\|\)).
**WGAN remarks**

**Wasserstein GAN objective:**

\[
\inf_{g \in G} \sup_{f \in F} \mathbb{E}_{x \sim p_{data}, z \sim p_{z}} \left[ \frac{1}{n} \sum_{i=1}^{n} f(x_i) - \frac{1}{m} \sum_{j=1}^{m} f(g(z_j)) \right],
\]

where “\(\|f\|_{\text{Lip}} \leq 1\)” means \(f\) 1-Lipschitz (\(\|f(x) - f(y)\| \leq \|x - y\|\)).
WGAN remarks

**Wasserstein GAN objective:**

\[
\inf_{g \in \mathcal{G}} \sup_{f \in \mathcal{F}} \left\{ \frac{1}{n} \sum_{i=1}^{n} f(x_i) - \frac{1}{m} \sum_{j=1}^{m} f(g(z_j)) \right\},
\]

where \( \|f\|_{\text{Lip}} \leq 1 \) means \( f \) 1-Lipschitz (\( \|f(x) - f(y)\| \leq \|x - y\| \)).

**Remarks.**

- In practice, \( \mathcal{G} \) and \( \mathcal{F} \) are deep networks architectures, \( \|f\|_{\text{Lip}} \) is only approximately enforced.
- This objective is a “Wasserstein distance” or “earth mover distance”; it can be interpreted as how much mass we have to shift to convert one distribution into another (in this case, \( g \# \mu \) and the original).
- The above formulate for Wasserstein distance is the “dual form” given via “Kantorovich-Rubinstein duality”.

Summary and Reflection
We gave two approaches (GAN and VAE) to sample with deep networks by training \( g \) and then sampling from \( g \# \mu \).

There are other ways to sample with deep networks (e.g., fit a density and then use Langevin), but no one talks about them?

**Open question:** how to evaluate GANs?! This is currently a disaster.

On the plus side: community wants evaluation that matches human notion of similarity.

Both GAN and VAE are used extensively; some approaches blend both (e.g., BicycleGAN).

GAN needs alternating minimization, VAE uses regular minimization. Both are finicky, though.
Original papers


- (Wasserstein GAN papers.)
Summary (of part 1).

- The sampling scheme: draw $x \sim \mu$ efficiently, then compute $g(x)$, where $g$ is a deep network.
- The basic VAE scheme and its objective function (The ERM perspective); perhaps recap in part 2 has cleanest presentation.

Summary (of part 2).

- GAN: minimize a distance on probability measures.
- Original: Jensen-Shannon divergence, and the corresponding alternating scheme.
- WGAN.