Generative networks part 2: GANs

Recap on generative networks

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

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

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

Recap on generative networks

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

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

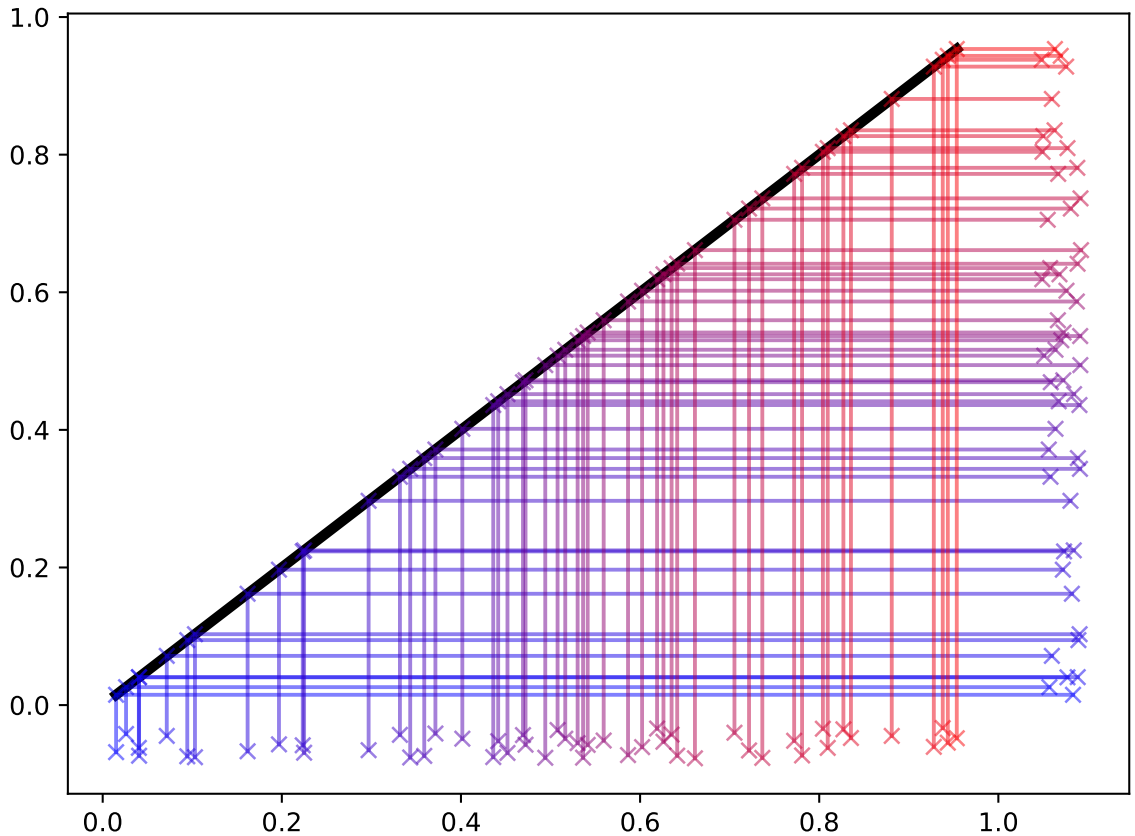
Notation: let $g\#\mu$ (pushforward of μ through g) denote this distribution.

Brief remarks:

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

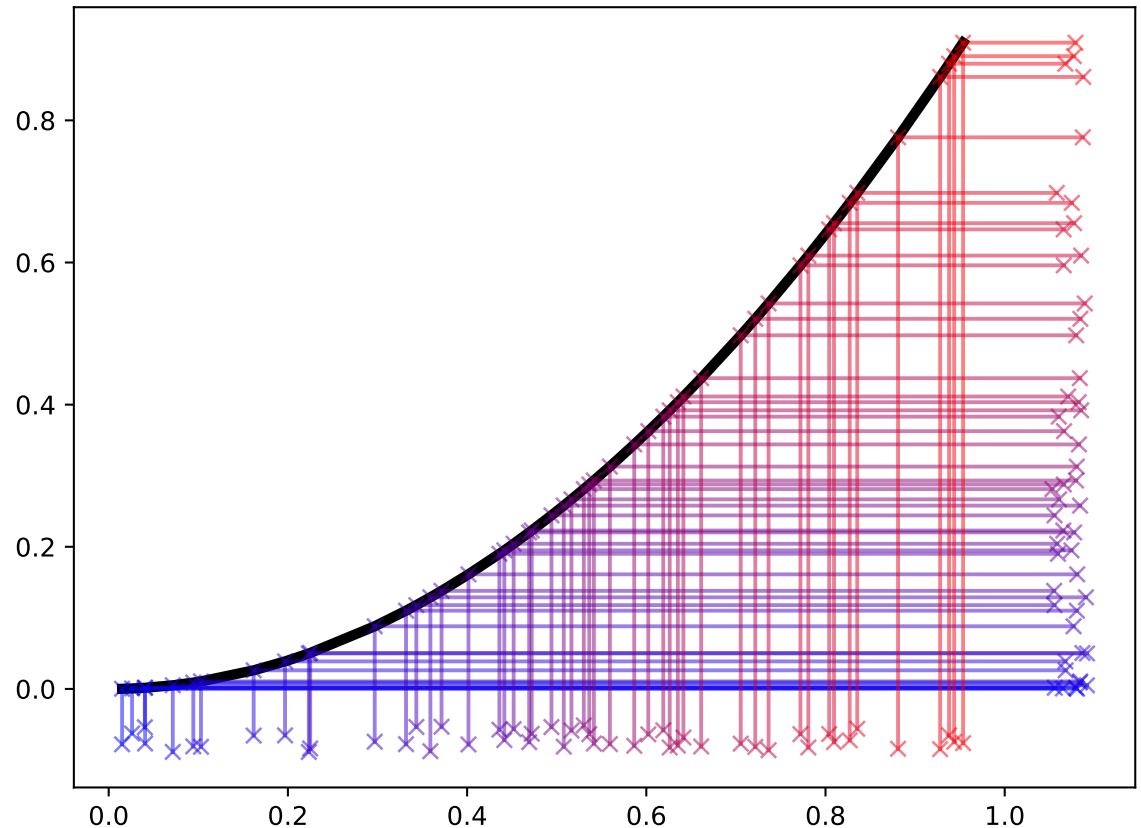
Univariate examples

$g(x) = x$, the identity function,
mapping Uniform$([0, 1])$ to itself.



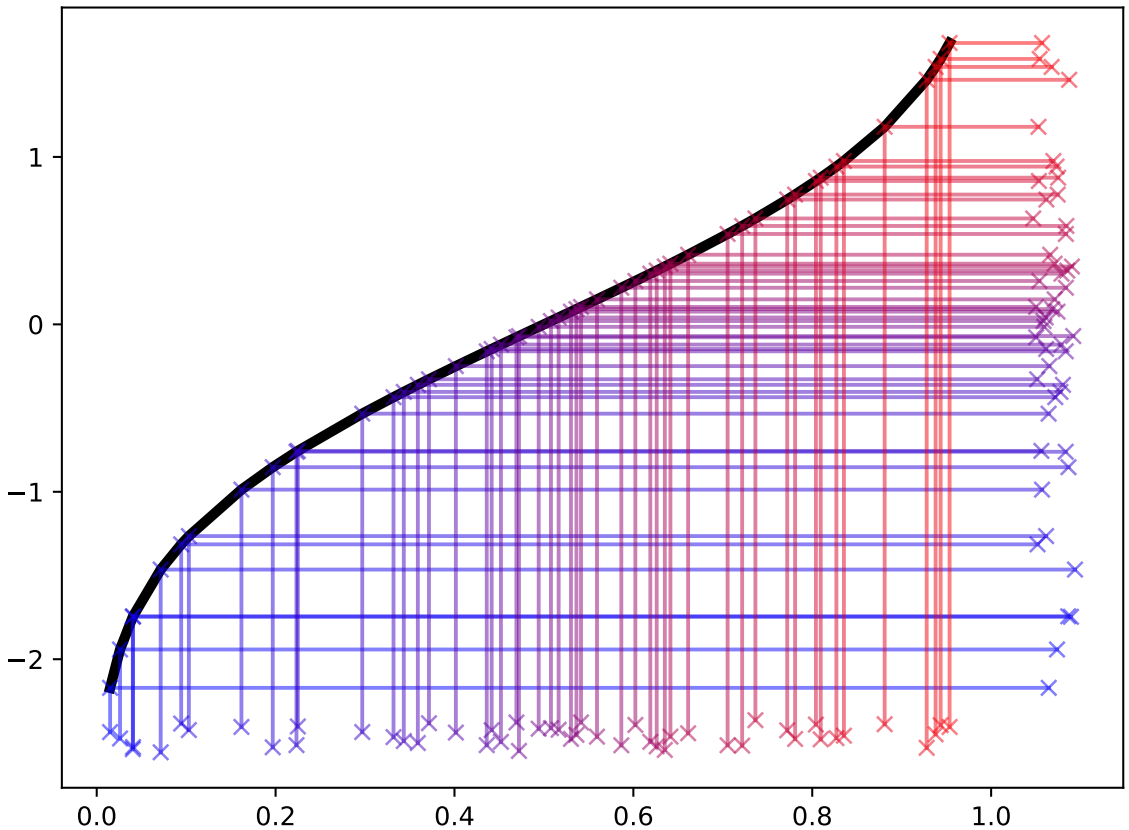
Univariate examples

$g(x) = x^2$,
mapping Uniform$([0, 1])$ to something $\propto \frac{2}{\sqrt{x}}$.



Univariate examples

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 $g\#\mu$),
here's the **"kernel density" / "Parzen window"** estimate of its density:

1. Start with random draw $(\boldsymbol{x}_i)_{i=1}^n$.
2. "Place bumps at every $\boldsymbol{x}_i$":
   Define $\hat{p}(\boldsymbol{x}) := \frac{1}{n} \sum_{i=1}^n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)$,
   where k is a **kernel function** (not the SVM one!),
   h is the "bandwidth"; for example:

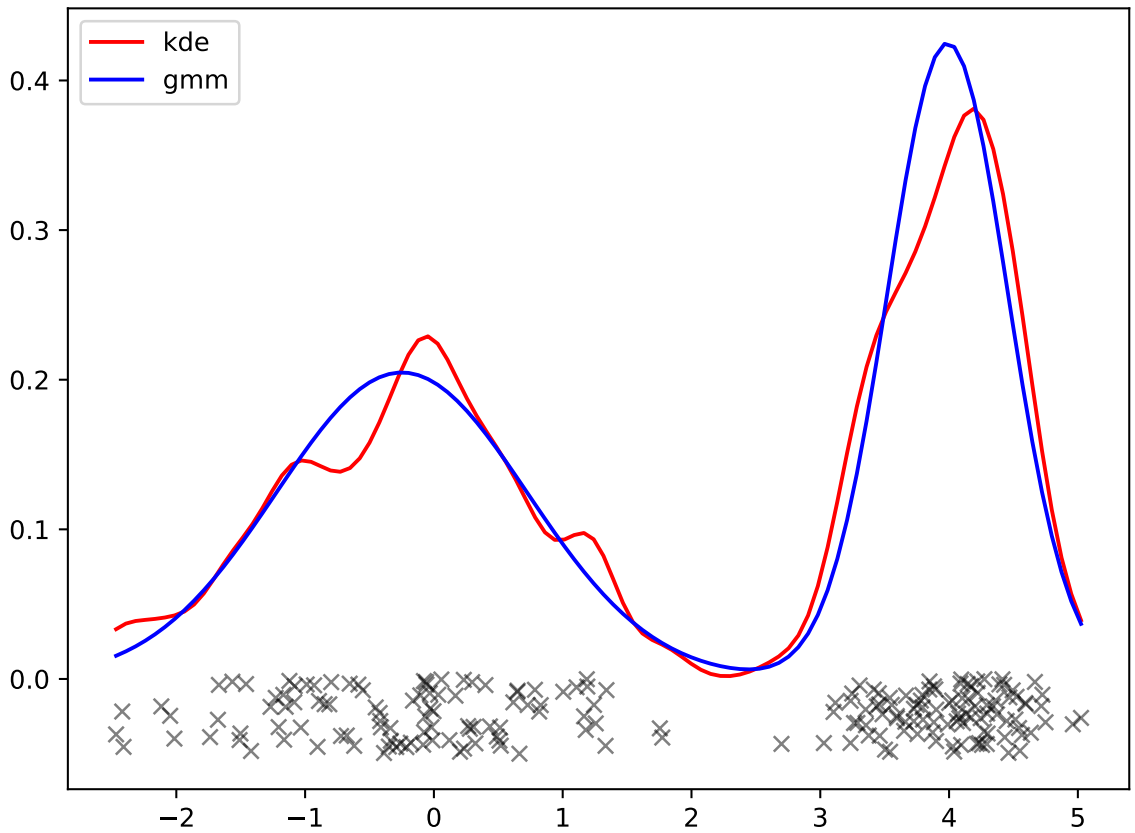
Another way to visualize generative networks

Given a sample from a distribution (even $g\#\mu$),
here's the **"kernel density" / "Parzen window"** estimate of its density:

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

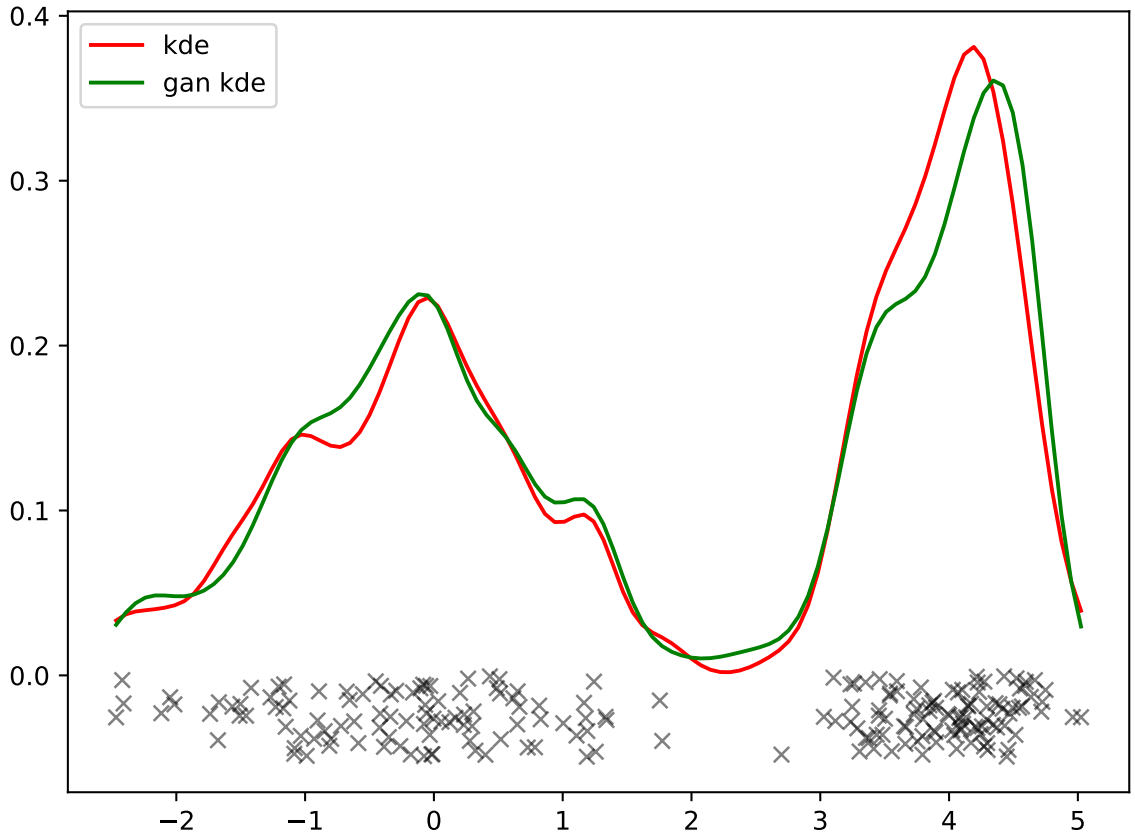
Examples — univariate sampling.

Univariate sample, kernel density estimate (kde), GMM E-M.



Examples — univariate sampling.

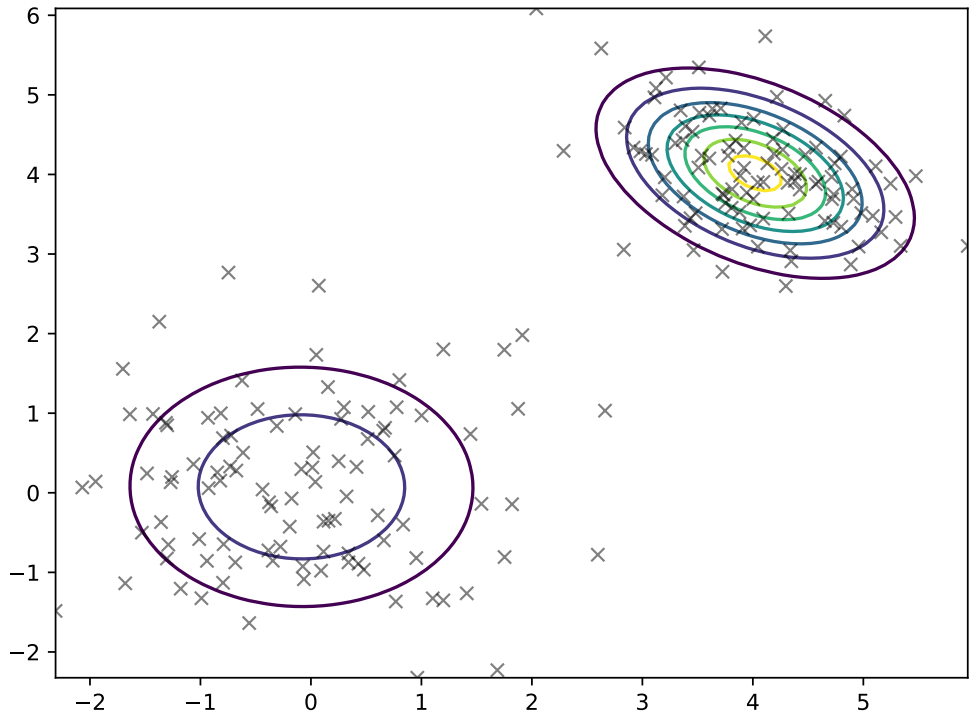
Univariate sample, kernel density estimate (kde), GAN kde.



This is admittedly very indirect!
As mentioned, there aren't great ways to get GAN/VAE density information.

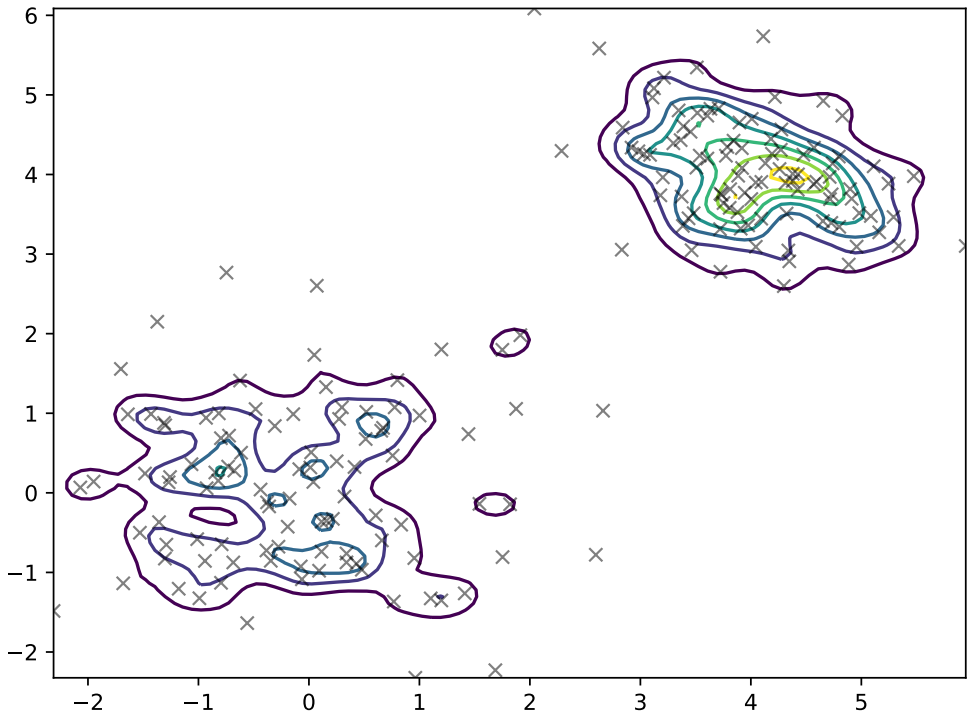
Examples — bivariate sampling.

Bivariate sample, GMM E-M.



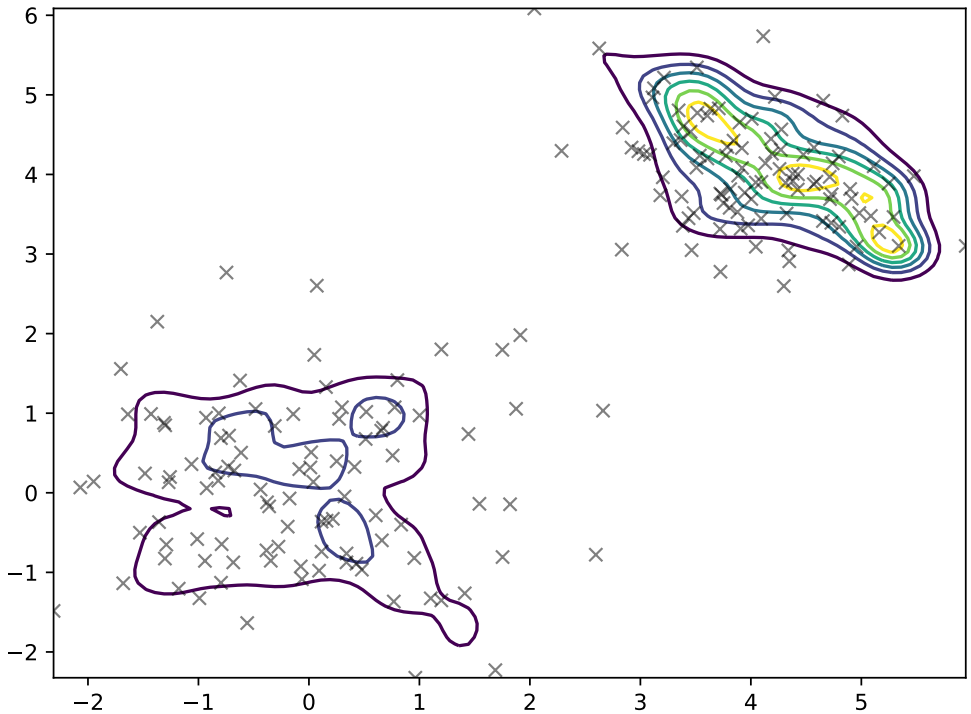
Examples — bivariate sampling.

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.

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**:

$$\boldsymbol{x}_i \xrightarrow[\text{map}]{f} \text{latent } \boldsymbol{z}_i = f(\boldsymbol{x}_i) \xrightarrow[\text{map}]{g} \hat{\boldsymbol{x}}_i = g(\boldsymbol{z}_i).$$

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

(Variational) Autoencoders

- **Autoencoder**:

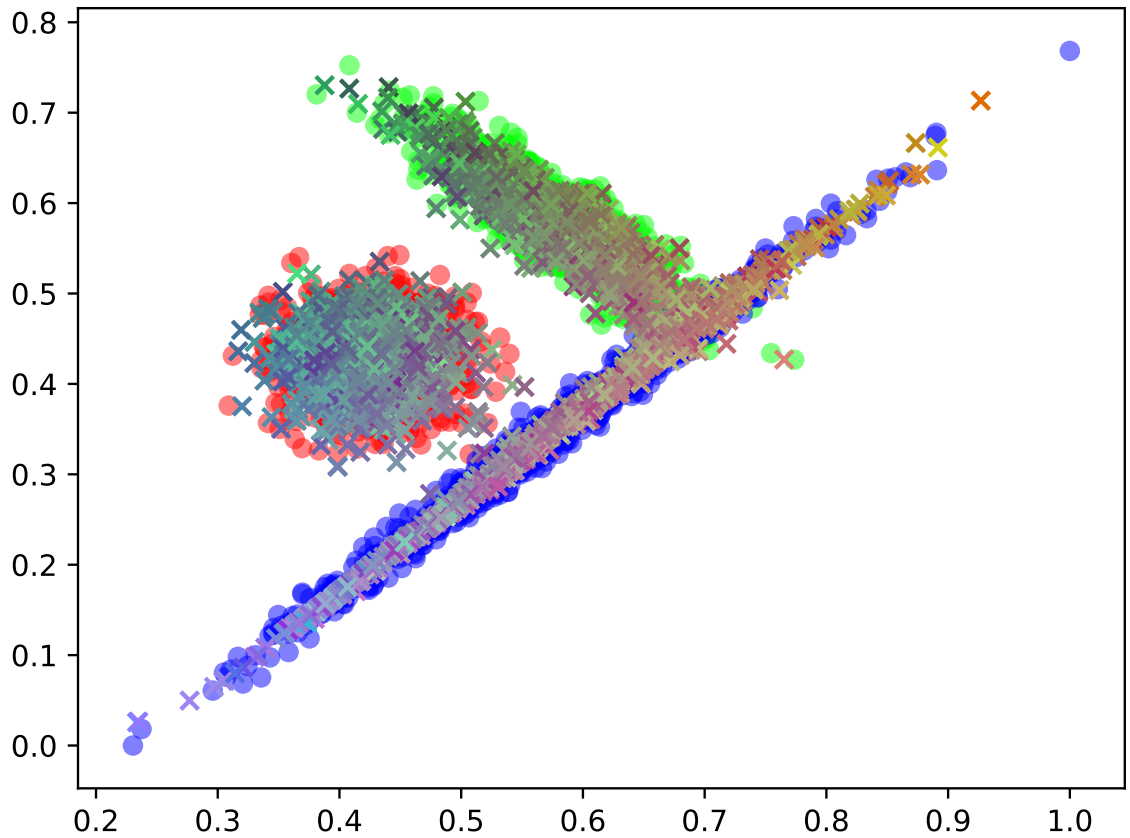
$$\boldsymbol{x}_i \xrightarrow[\text{map}]{f} \text{latent } \boldsymbol{z}_i = f(\boldsymbol{x}_i) \xrightarrow[\text{map}]{g} \hat{\boldsymbol{x}}_i = g(\boldsymbol{z}_i).$$

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

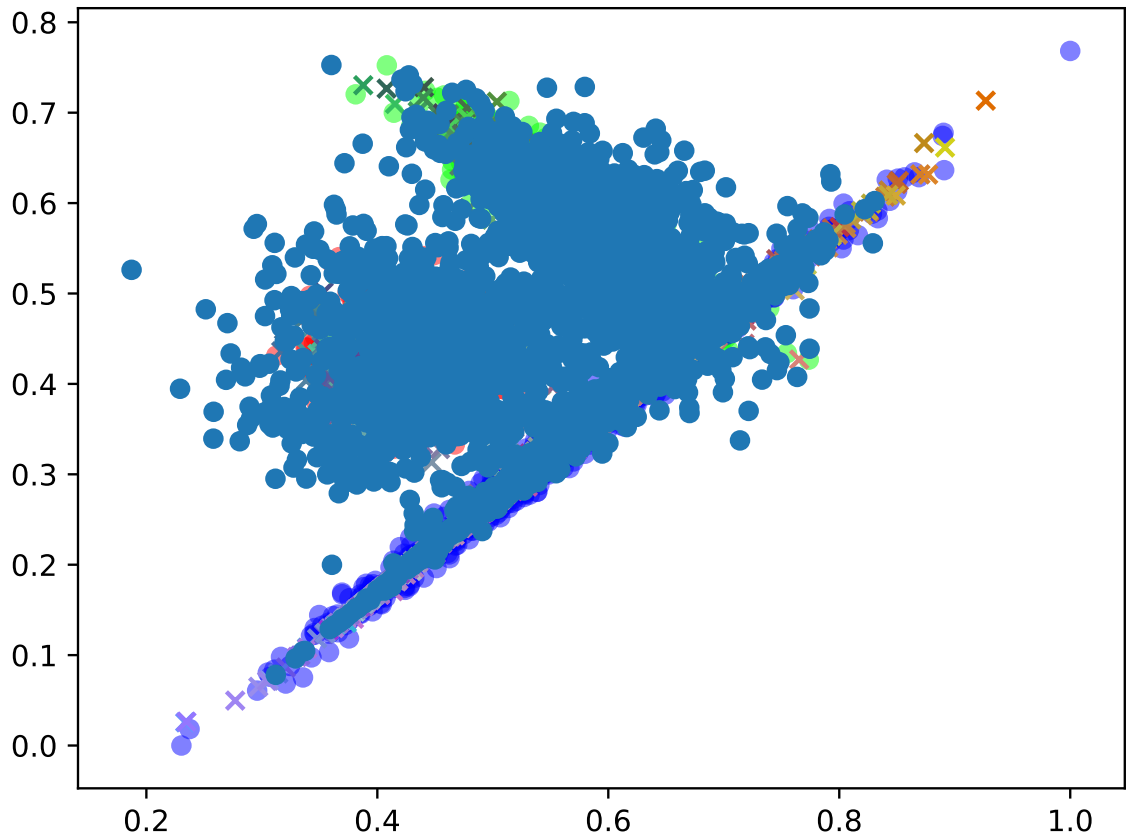
- **Variational Autoencoder**:

$$\boldsymbol{x}_i \xrightarrow[\text{map}]{f} \text{latent distribution } \mu_i = f(\boldsymbol{x}_i) \xrightarrow[\text{pushforward}]{g} \hat{\boldsymbol{x}}_i \sim g\#\mu_i.$$

Objective: $\frac{1}{n}\sum_{i=1}^{n} \left[\ell(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i) + \lambda \mathsf{KL}(\mu, \mu_i)\right]$.



$$\hat{\boldsymbol{x}}_i \sim g\#\mu_i$$



$\hat{\boldsymbol{x}}_i \sim g\#\mu$ with small λ

Generative Adversarial Networks (GANs)

Generative network setup and training.

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

Problem: this isn't as simple as fitting $g(\boldsymbol{z}_i) \approx \boldsymbol{x}_i$.

Generative network setup and training.

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

Problem: this isn't as simple as fitting $g(\boldsymbol{z}_i) \approx \boldsymbol{x}_i$.

Solutions:

- VAE: For each $\boldsymbol{x}_i$, construct distribution μ_i, so that $\hat{\boldsymbol{x}}_i \sim g\#\mu_i$ and $\boldsymbol{x}_i$ are close, as are μ_i and μ.
  To generate fresh samples, get $\boldsymbol{z} \sim \mu$ and output $g(\boldsymbol{z})$.
- GAN: Pick a distance notion between distributions (or between samples $(g(\boldsymbol{z}_i))_{i=1}^n$ and $(\boldsymbol{x}_i)_{i=1}^n$) 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 $(\hat{\boldsymbol{x}}_i)_{i=1}^n$ with $\hat{\boldsymbol{x}}_i = g_t(\boldsymbol{z}_i)$.
2. We train a *discriminator/critic* f_t to find differences between $(\hat{\boldsymbol{x}}_i)_{i=1}^n$ and $(\boldsymbol{x}_i)_{i=1}^n$.
3. We then pick a new *generator* g_{t+1}, trained to fool f_t!

Jensen-Shannon divergence (original GAN)

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 **Jensen-Shannon Divergence**:

$$\begin{aligned} 2 \cdot \mathsf{JS}(p, p_g) &= \mathsf{KL}(p, \tilde{p}) + \mathsf{KL}(p_g, \tilde{p}) \\ &= \int p(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} + \int p_g(\boldsymbol{x}) \ln \frac{p_g(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} \\ &= \mathbb{E}_p \ln \frac{p(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} + \mathbb{E}_{p_g} \ln \frac{p_g(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})}. \end{aligned}$$

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 **Jensen-Shannon Divergence**:

$$\begin{aligned} 2 \cdot \mathsf{JS}(p, p_g) &= \mathsf{KL}(p, \tilde{p}) + \mathsf{KL}(p_g, \tilde{p}) \\ &= \int p(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} + \int p_g(\boldsymbol{x}) \ln \frac{p_g(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} \\ &= \mathbb{E}_p \ln \frac{p(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} + \mathbb{E}_{p_g} \ln \frac{p_g(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})}. \end{aligned}$$

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 **Jensen-Shannon Divergence**:

$$\begin{aligned} 2 \cdot \mathsf{JS}(p, p_g) &= \mathsf{KL}(p, \tilde{p}) + \mathsf{KL}(p_g, \tilde{p}) \\ &= \int p(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} + \int p_g(\boldsymbol{x}) \ln \frac{p_g(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} \\ &= \mathbb{E}_p \ln \frac{p(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})} + \mathbb{E}_{p_g} \ln \frac{p_g(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})}. \end{aligned}$$

But we've been saying we can't write down p_g?

Original GAN approach applies alternating minimization to

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

Original GAN formulation and algorithm.

Original GAN objective:

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

Algorithm alternates these two steps:

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

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

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

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

Some implementation issues

Algorithm alternates these two steps:

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

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

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

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

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), $p_{\boldsymbol{z}}$ (on $\boldsymbol{z}$),

$$
\begin{aligned}
&\mathbb{E} \ln f(\boldsymbol{x}) + \mathbb{E} \ln(1 - f(g(\boldsymbol{z}))) \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(g(\boldsymbol{z}))) p_{\boldsymbol{z}}(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \\
&= \int \Big(\ln f(\boldsymbol{x}) p(\boldsymbol{x}) + \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \Big) \, \mathrm{d}\boldsymbol{x}.
\end{aligned}
$$

Optimal discriminator

Given p (of data), p_g (from g), $p_{\boldsymbol{z}}$ (on $\boldsymbol{z}$),

$$\begin{aligned}
&\mathbb{E} \ln f(\boldsymbol{x}) + \mathbb{E} \ln(1 - f(g(\boldsymbol{z}))) \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(g(\boldsymbol{z}))) p_{\boldsymbol{z}}(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \\
&= \int \Big(\ln f(\boldsymbol{x}) p(\boldsymbol{x}) + \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \Big) \, \mathrm{d}\boldsymbol{x}.
\end{aligned}$$

To find maximal f, maximize pointwise.

Optimal discriminator

Given p (of data), p_g (from g), $p_{\boldsymbol{z}}$ (on $\boldsymbol{z}$),

$$\begin{aligned}
&\mathbb{E} \ln f(\boldsymbol{x}) + \mathbb{E} \ln(1 - f(g(\boldsymbol{z}))) \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(g(\boldsymbol{z}))) p_{\boldsymbol{z}}(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \\
&= \int \Big(\ln f(\boldsymbol{x}) p(\boldsymbol{x}) + \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \Big) \, \mathrm{d}\boldsymbol{x}.
\end{aligned}$$

To find maximal f, maximize pointwise.

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

Optimal discriminator

Given p (of data), p_g (from g), $p_{\boldsymbol{z}}$ (on $\boldsymbol{z}$),

$$\begin{aligned}
&\mathbb{E} \ln f(\boldsymbol{x}) + \mathbb{E} \ln(1 - f(g(\boldsymbol{z}))) \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(g(\boldsymbol{z}))) p_{\boldsymbol{z}}(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \\
&= \int \ln f(\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \\
&= \int \Big(\ln f(\boldsymbol{x}) p(\boldsymbol{x}) + \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \Big) \, \mathrm{d}\boldsymbol{x}.
\end{aligned}$$

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 $f(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{p(\boldsymbol{x}) + p_g(\boldsymbol{x})}$.

Recovering Jensen-Shannon divergence

Let's plug in optimal discriminator $f(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{p(\boldsymbol{x})+p_g(\boldsymbol{x})}$:

$$
\begin{aligned}
&\sup_{f \in \mathcal{F}} \mathbb{E} \ln f(\boldsymbol{x}) + \mathbb{E} \ln(1 - f(g(\boldsymbol{z}))) \\
&= \sup_{f \in \mathcal{F}} \int \Big(\ln f(\boldsymbol{x}) p(\boldsymbol{x}) + \ln(1 - f(\boldsymbol{x})) p_g(\boldsymbol{x}) \Big) \, \mathrm{d}\boldsymbol{x}. \\
&= \int \left(p(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{p(\boldsymbol{x}) + p_g(\boldsymbol{x})} + p_g(\boldsymbol{x}) \ln \frac{p_g(\boldsymbol{x})}{p(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) \mathrm{d}\boldsymbol{x} \\
&= \int \left(p(\boldsymbol{x}) \ln \frac{2p(\boldsymbol{x})}{p(\boldsymbol{x}) + p_g(\boldsymbol{x})} + p_g(\boldsymbol{x}) \ln \frac{2p_g(\boldsymbol{x})}{p(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) \mathrm{d}x - \ln 4 \\
&= \mathsf{KL}\left(p, \frac{p + p_g}{2} \right) + \mathsf{KL}\left(p_g, \frac{p + p_g}{2} \right) - \ln 4 \\
&= 2 \cdot \mathsf{JS}\left(p, p_g \right) - \ln 4.
\end{aligned}
$$

Technical remarks.

- 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 $p/(p+p_g)$.
  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:

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

Wasserstein GAN (WGAN)

Original GAN objective:

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

Wasserstein GAN objective:

$$\inf_{g\in\mathcal{G}} \sup_{\substack{f\in\mathcal{F}\\ \|f\|_{\mathsf{Lip}}\leq 1}} \left[\frac{1}{n}\sum_{i=1}^{n}f(\boldsymbol{x}_i)-\frac{1}{m}\sum_{j=1}^{m}f(g(\boldsymbol{z}_j))\right],$$

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

WGAN remarks

Wasserstein GAN objective:

$$\inf_{g \in \mathcal{G}} \sup_{\substack{f \in \mathcal{F} \\ \|f\|_{\text{Lip}} \leq 1}} \left[\frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}_i) - \frac{1}{m} \sum_{j=1}^{m} f(g(\boldsymbol{z}_j)) \right],$$

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

WGAN remarks

Wasserstein GAN objective:

$$\inf_{g \in \mathcal{G}} \sup_{\substack{f \in \mathcal{F} \\ \|f\|_{\text{Lip}} \leq 1}} \left[\frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}_i) - \frac{1}{m} \sum_{j=1}^{m} f(g(\boldsymbol{z}_j)) \right],$$

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

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

- **(Original VAE paper.)** Diederik P Kingma, Max Welling. "Auto-Encoding Variational Bayes". 2013.
- **(Original GAN paper.)** Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. "Generative adversarial nets". 2014.
- **(Wasserstein GAN papers.)**
  - Arjovsky, Martin, Chintala, Soumith, and Bottou, Leon. "Wasserstein generative adversarial networks". 2017.
  - Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, and Courville, Aaron C. "Improved training of Wasserstein GANs". 2017.

Summary (of part 1).

- The sampling scheme: draw $\boldsymbol{x} \sim \mu$ efficiently, then compute $g(\boldsymbol{x})$, 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.