Advanced Course on Deep Generative Models

Lecture 8: GANs, Samples, Energy/Score-based models

Adapted from Volodymyr Kuleshov

Dan Rosenbaum, CS Haifa

Today

- Likelihood vs. Sampling
- Two-Sample Tests
- Generative Adversarial Network (GAN)
- Evaluating Samples
- Energy Based Models
- Score Based Models

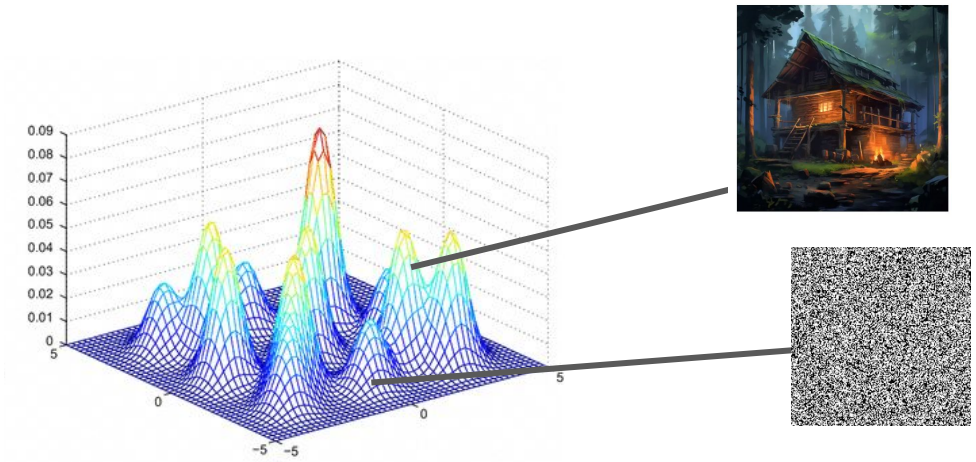
Generative models

Probabilistic model with high dimensional output.

- looking for the parameters θ such that p_θ is close to p_{data}

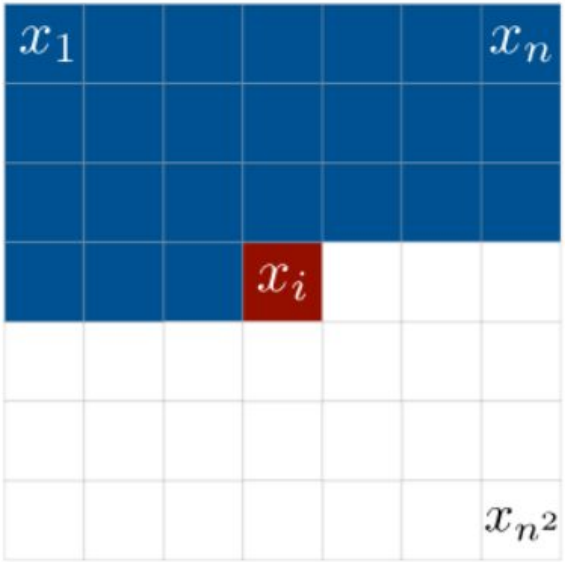
Usage:

- Generate data
- Representation learning
- Uncertainty based decisions
- Probabilistic inference

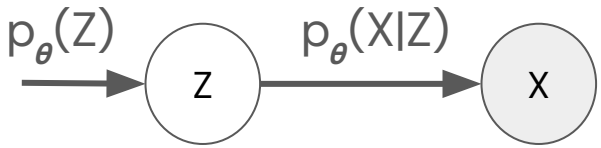


Recap - autoregressive models

- Autoregressive models: $p_\theta(x) = \Pi\, p_\theta(x_i | x_{<i})$
- Probability distributions factorize into a product of factors
- We can efficiently represent **p** via conditional independence and/or neural parameterizations
- Problem: Sampling is slow



Recap - Variational Autoencoders



- Combine simple models into more flexible ones:
  $p(x) = \int p_\theta(x|z)p(z)dz$ with "simple" models $p_\theta(x|z)$, $p(z)$
- *Train via amortized variational inference using an inference network to approximate the posterior* $q_\phi(z \mid x)$
- Directed model permits efficient generation: $z \sim p(z)$, $x \sim p_\theta(x|z)$
- Problem: samples are usually blurry

Good likelihood does not imply good samples

An optimal model will have the best sample quality. But what happens when the model has a good but not optimal likelihood?

- Likelihood is more affected by local rather than global structure

- Consider this model:

$$p_\theta(\mathbf{x}) = 0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x})$$

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \log[0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x})] \\ &\geq \log 0.01 p_{\text{data}}(\mathbf{x}) = \log p_{\text{data}}(\mathbf{x}) - \log 100\end{aligned}$$

Good samples don’t imply good likelihood

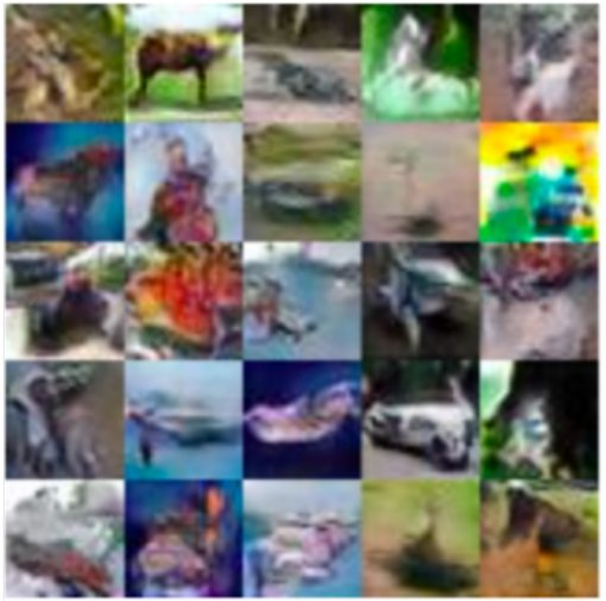
A model that overfits the training data will have

- Great samples
- Very bad likelihood

Idea: If we want care about the sampling ability of the model, optimize it rather than the likelihood.

Optimizing sampling

vs.



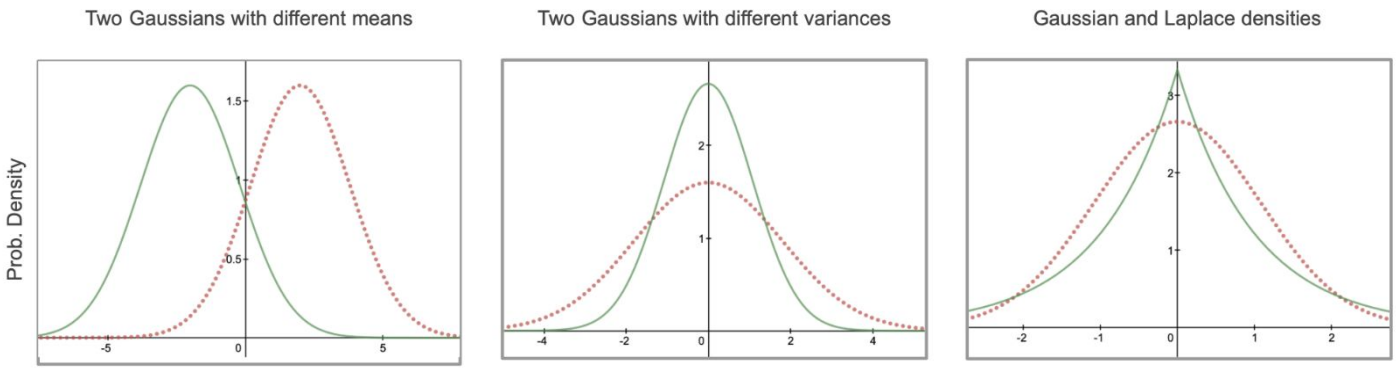
$S_1 = \{\mathbf{x} \sim P\}$

$S_2 = \{\mathbf{x} \sim Q\}$

Two-Sample Tests

- Given $\mathbf{S_1 = \{x \sim P\}}$ and $\mathbf{S_2 = \{x \sim Q\}}$, a two-sample test considers the following hypotheses:
  - Null hypothesis $\mathbf{H_0 : P = Q}$
  - Alternate hypothesis $\mathbf{H_1 : P \neq Q}$
- Test statistic **T** compares $\mathbf{S_1}$ and $\mathbf{S_2}$ e.g., difference in means, variances of the two sets of samples
- If **T** is less than a threshold **α**, then accept $\mathbf{H_0}$ else reject it
- Key observation: Test statistic is likelihood-free since it does not involve the densities **P** or **Q** (only samples)

Two-Sample Tests



- Finding a good two-sample test statistic is not easy
- Key idea: Learn a statistic that maximizes a suitable notion of distance between the two sets of samples $\mathbf{S}_1$ and $\mathbf{S}_2$

- Suppose we have data coming from a balanced mixture of distributions $P(X|Y=0)$ and $P(X|Y=1)$ indexed by Y.
- Let's use supervised learning to predict the component Y from X:

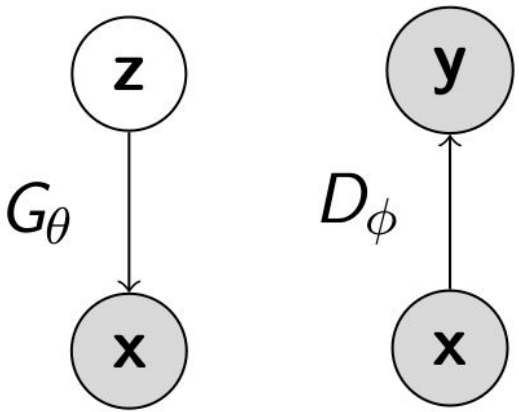
$$P(Y=1|X) = \frac{P(X|Y=1)P(Y=1)}{P(X)} = \frac{P(X|Y=1)}{P(X|Y=0)+P(X|Y=1)}$$
$$= \frac{1}{1+\frac{P(X|Y=0)}{P(X|Y=1)}} = \sigma\left(\log \frac{P(X|Y=0)}{P(X|Y=1)}\right)$$

- Thus, logistic regression trained on $(X,Y) \sim P$ pairs estimates the log odds, which can form the basis of a two-sample test:

$$\log \frac{P(X|Y=0)}{P(X|Y=1)}.$$

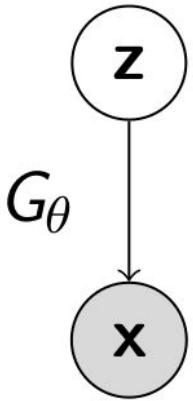
Generative Adversarial Networks

A GAN is defined by two models:
a **generator** and a **discriminator**.



Generator

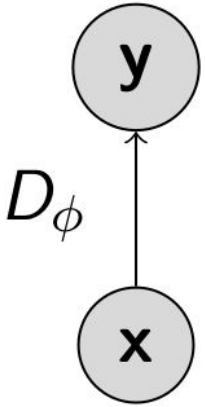
- The generator defines a directed, latent variable model with a deterministic mapping between **z** and **x** given by $\mathbf{G_\theta}$
- Directly outputs samples **x**, not parameters for a distribution over **x** (as in a VAE). Hence probability is defined implicitly.
- Minimizes a two-sample test objective (in support of the null hypothesis $\mathbf{p_{data} = p_\theta}$)



Discriminator

The discriminator is a function (e.g., a neural network) which distinguishes “real” samples from the dataset and “fake” samples generated from the model.

- The discriminator maximizes the two-sample test objective (in support of the alternate hypothesis $p_{data} \neq p_{\theta}$)
- The generator and the discriminator can be seen as playing a two player minimax game between each other.



Discriminator Training Objective

$$\max_D V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G}[\log(1 - D(\mathbf{x}))]$$

- For a fixed generator G , the discriminator is performing binary classification with the cross entropy objective
  - Assign probability 1 to true data points $\mathbf{x} \sim \mathbf{p}_{\text{data}}$
  - Assign probability 0 to fake samples $\mathbf{x} \sim \mathbf{p}_{\mathbf{G}}$
- We can prove that the optimal discriminator equals

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

Generator Training Objective

$$\min_G V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G}[\log(1 - D(\mathbf{x}))]$$

- Same objective as discriminator but in the opposite direction
- A game between the discriminator and generator

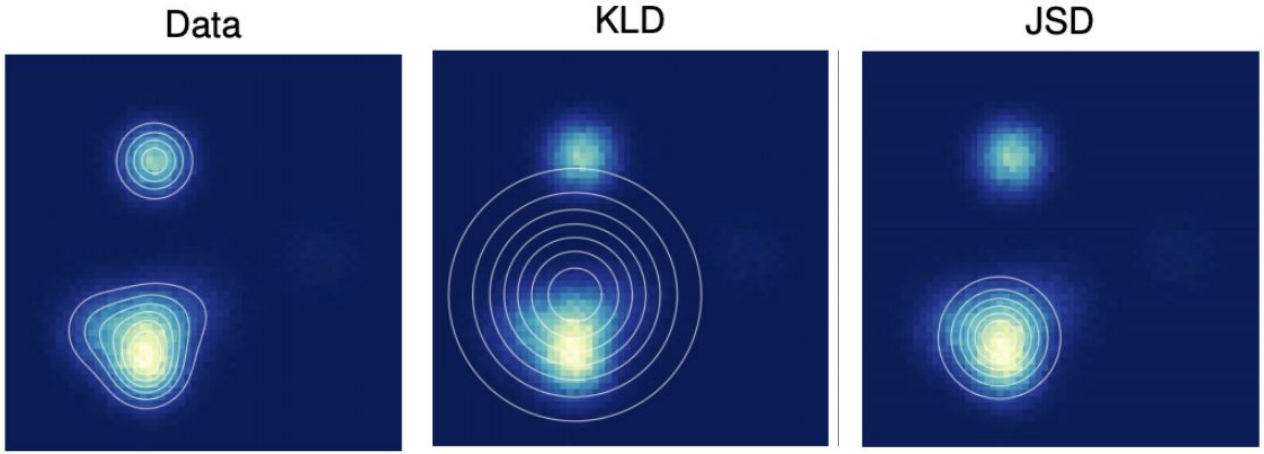
For an Optimal Discriminator

$$
\begin{aligned}
& V(G, D_G^*(\mathbf{x})) \\
&= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}\right] \\
&= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] - \log 4 \\
&= \underbrace{D_{KL}\left[p_{\text{data}}, \frac{p_{\text{data}}+p_G}{2}\right] + D_{KL}\left[p_G, \frac{p_{\text{data}}+p_G}{2}\right]}_{2\times\text{Jenson-Shannon Divergence (JSD)}} - \log 4 \\
&= 2D_{JSD}[p_{\text{data}}, p_G] - \log 4
\end{aligned}
$$

Jensen-Shannon Divergence

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

- Also called the symmetric KL divergence
- Mode seeking



GAN training algorithm

- Sample minibatch of m training points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}$ from $\mathcal{D}$
- Sample minibatch of m noise vectors $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(m)}$ from p_z
- Update the generator parameters θ by stochastic gradient **descent**

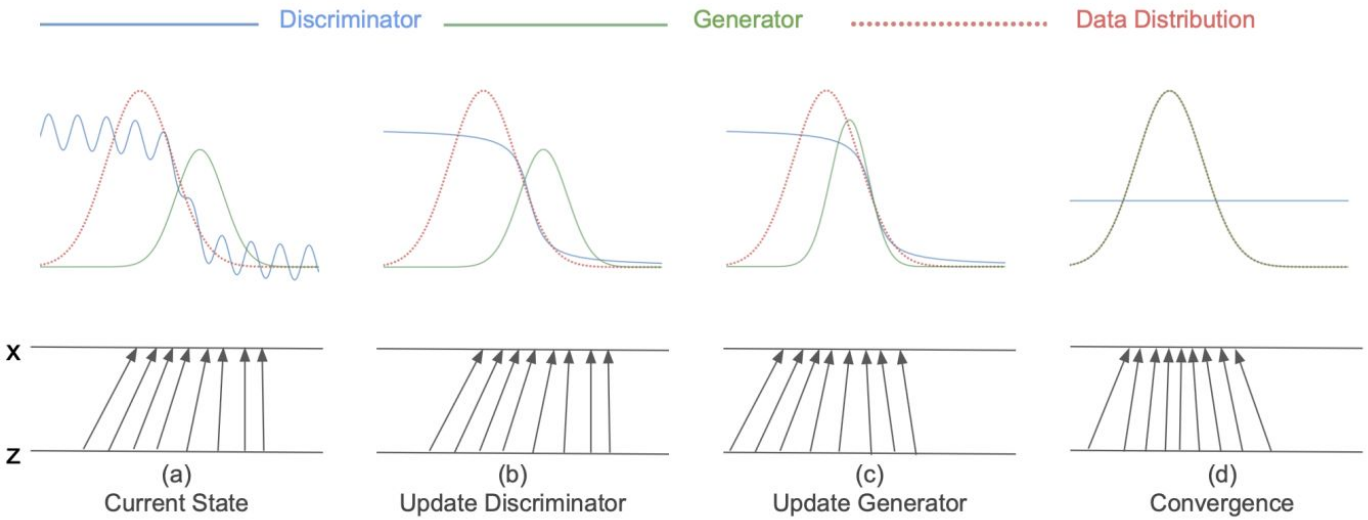
$$\nabla_\theta V(G_\theta, D_\phi) = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \log(1 - D_\phi(G_\theta(\mathbf{z}^{(i)})))$$

- Update the discriminator parameters ϕ by stochastic gradient **ascent**

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

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

We can visualize training in 1D via a picture:

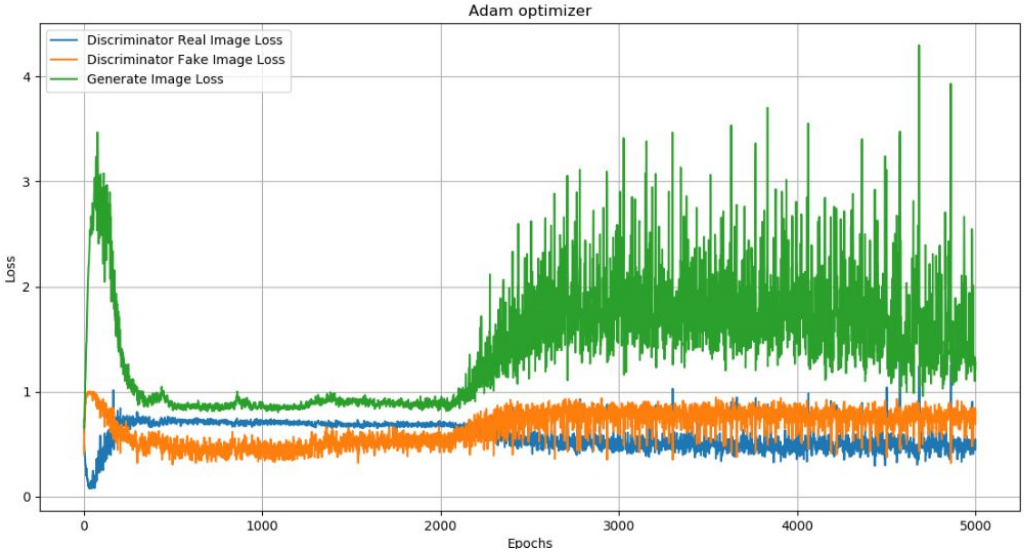


- GANs have been successfully applied to several domains and tasks
- However, working with GANs can be very challenging in practice
  - Unstable optimization
  - Mode collapse
  - Evaluation
- Many bag of tricks applied to train GANs successfully

Optimization Challenges

- Generator and Discriminator are oscillating

- No clear stopping criterion (like in maximum likelihood)



Mode collapse

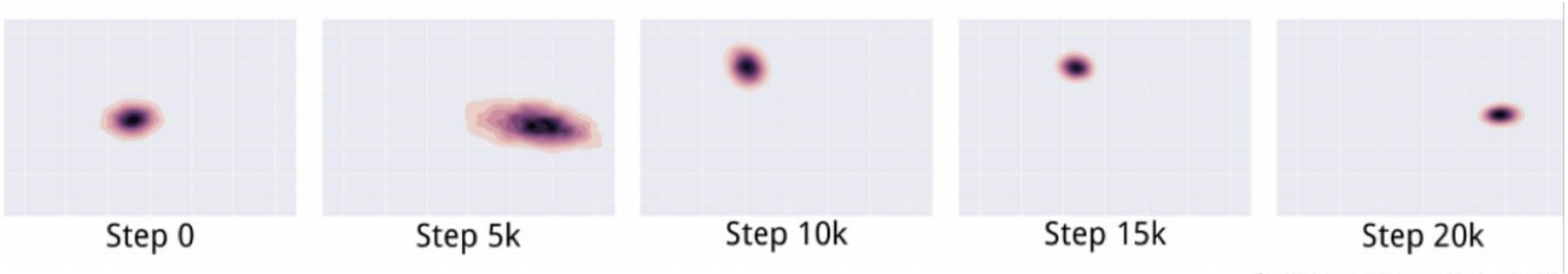
Generator focuses on a few representative examples

Arjovsky et al., 2017

Mode collapse

- True distribution is a GMM

- Generator keeps oscillating between different modes



Source: Metz et al., 2017

Summary of GANs

- GAN Pros:
  - Very high-quality samples.
  - Can optimize a wide range of divergences between probabilities
  - Broadly applicable: only need sampling from G !
- GAN Cons:
  - Difficult to train
  - Suffers from mode collapse

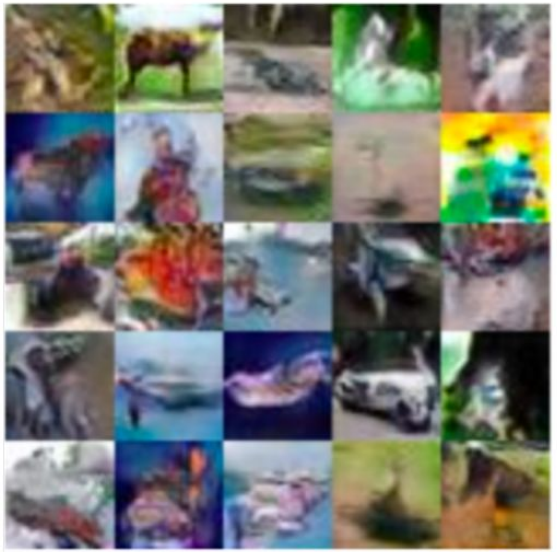
    See https://github.com/soumith/ganhacks for tips and tricks to train GANs

Evaluating Sample Quality

How can we quantitatively measure sampling?

$S_1 = \{\mathbf{x} \sim P\}$

vs.



$S_2 = \{\mathbf{x} \sim Q\}$

Evaluating Sample Quality

- Human evaluations are expensive, biased, hard to reproduce
- Generalization is hard to define and assess: memorizing the training set would give excellent samples but clearly undesirable
- Quantitative evaluation of a qualitative task can have many answers
- Popular metrics: Inception Scores, Frechet Inception Distance

Inception Score

Use a pre-trained image classifier

- **Assumption 1**: We are evaluating sample quality for generative models trained on labelled datasets

- **Assumption 2**: We have a good probabilistic classifier **c(y |x)** for predicting the label **y** for any point **x**

- We want samples from a good generative model to satisfy two criteria: prediction sharpness and prediction diversity

Prediction sharpness

$$S = \exp\left(E_{\mathbf{x}\sim p}\left[\int c(y|\mathbf{x}) \log c(y|\mathbf{x}) \mathrm{d}y\right]\right)$$

- High sharpness implies classifier is confident in making predictions for generated images
- That is, classifier's predictive distribution **c(y |x)** has low entropy

Prediction Diversity

Low diversity

High diversity

$$D = \exp\left(-E_{\mathbf{x}\sim p}\left[\int c(y|\mathbf{x}) \log c(y) \mathrm{d}y\right]\right)$$

where **c(y) = $E_{x\sim p}$ [c(y |x)]** is the classifier's marginal predictive distribution

- High diversity implies **c(y)** has high entropy

Inception Score

$$IS = D \times S = \mathbb{E}_{x \sim p} \left[KL \left(c(y|x) || c(y) \right) \right]$$

- combine the two criteria of sharpness and diversity into a simple metric
- Correlates well with human judgement in practice
- Usually computed with a pre-trained Inception Net (trained on the ImageNet dataset)

Frechet Inception Distance (FID)

- Inception Scores only require samples from $\mathbf{p}_\theta$ and do not take into account the desired data distribution $\mathbf{p}_{\text{data}}$ directly (only implicitly via a classifier)

- Frechet Inception Distance (FID) measures similarities in the feature representations (e.g., those learned by a pretrained classifier) for datapoints sampled from $\mathbf{p}_\theta$ and the test dataset

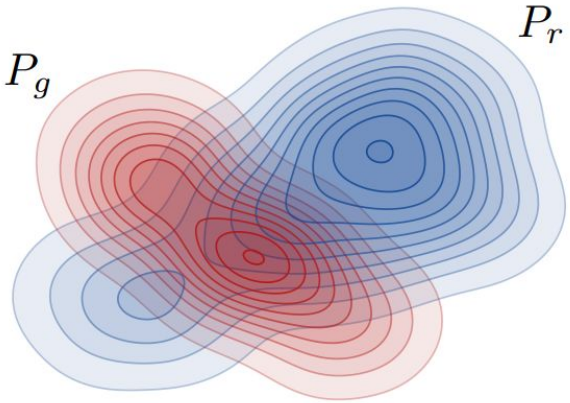
Computing FID

- Let **G** denote the generated samples and **T** denote the test dataset
- Compute feature representations $\mathbf{F_G}$ and $\mathbf{F_T}$ for **G** and **T** respectively (e.g., prefinal layer of Inception Net)
- Fit a multivariate Gaussian to each of $\mathbf{F_G}$ and $\mathbf{F_T}$. Let $(\boldsymbol{\mu_G}, \boldsymbol{\Sigma_G})$ and $(\boldsymbol{\mu_T}, \boldsymbol{\Sigma_T})$ denote the mean and covariances of the two Gaussians
- FID is defined as:

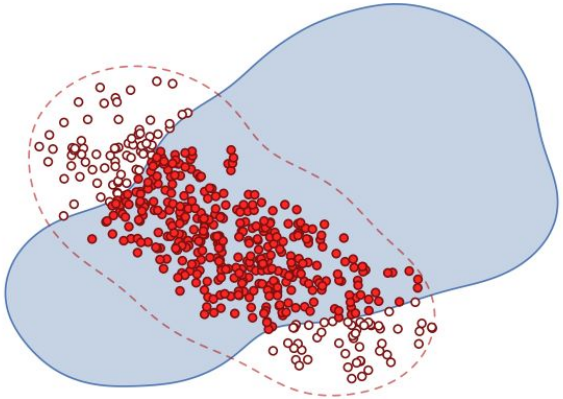
$$\text{FID} = \|\mu_{\mathcal{T}} - \mu_{\mathcal{G}}\|^2 + \text{Tr}(\Sigma_{\mathcal{T}} + \Sigma_{\mathcal{G}} - 2(\Sigma_{\mathcal{T}}\Sigma_{\mathcal{G}})^{1/2})$$

Lower FID implies better sample quality

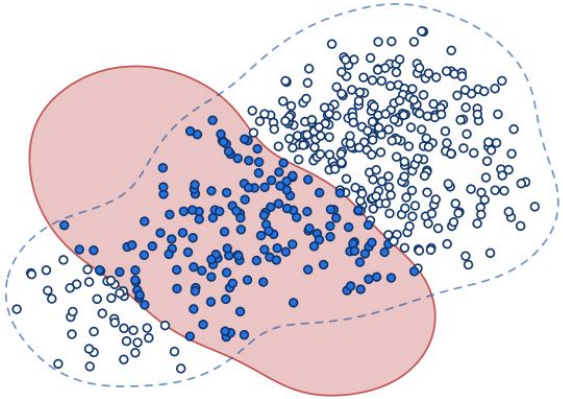
Other measures



(a) Example distributions



(b) Precision



(c) Recall

CelebA-HQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
DC-VAE [63]	15.8	-	-
VQGAN+T. [23] (k=400)	10.2	-	-
PGGAN [39]	8.0	-	-
LSGM [93]	7.22	-	-
UDM [43]	7.16	-	-
LDM-4 (ours, 500-s†)	**5.11**	0.72	0.49

FFHQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
ImageBART [21]	9.57	-	-
U-Net GAN (+aug) [77]	10.9 (7.6)	-	-
UDM [43]	5.54	-	-
StyleGAN [41]	4.16	0.71	0.46
ProjectedGAN [76]	**3.08**	0.65	0.46
LDM-4 (ours, 200-s)	4.98	**0.73**	**0.50**

LSUN-Churches 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
DDPM [30]	7.89	-	-
ImageBART [21]	7.32	-	-
PGGAN [39]	6.42	-	-
StyleGAN [41]	4.21	-	-
StyleGAN2 [42]	3.86	-	-
ProjectedGAN [76]	**1.59**	0.61	0.44
*LDM-8** (ours, 200-s)	4.02	**0.64**	**0.52**

LSUN-Bedrooms 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
ImageBART [21]	5.51	-	-
DDPM [30]	4.9	-	-
UDM [43]	4.57	-	-
StyleGAN [41]	2.35	0.59	0.48
ADM [15]	1.90	**0.66**	**0.51**
ProjectedGAN [76]	**1.52**	0.61	0.34
LDM-4 (ours, 200-s)	2.95	**0.66**	0.48

Energy Based Models

Parameterizing Probability Distributions

- Representing probability distributions **p(x)** is a key challenge in generative
- modeling. Probability distributions share two properties:
  - non-negativity: **$p(x) \geq 0$**
  - Sum-to-one: **$\int p(x)dx = 1$**

Sum-to-one is key. Total "volume" is fixed: increasing **$p(x_{train})$** guarantees that **x_{train}** becomes relatively more likely (compared to the rest).

Problem: coming up with a positive function is easy, but enforcing integral to be one is hard.

Parameterizing Probability Distributions

$$p_\theta(x) = \frac{1}{Volume(g_\theta)} g_\theta(x) = \frac{1}{\int g_\theta(x)dx} g_\theta(x)$$

Previously, we chose $g_\theta(x)$ so that we know the volume *analytically*:

1. **Autoregressive:** Products of normalized objects $p_\theta(x)p_{\theta'(x)}(y)$:
   $\int_x \int_y p_\theta(x)p_{\theta'(x)}(y)dxdy = \int_x p_\theta(x) \underbrace{\int_y p_{\theta'(x)}(y)dy}_{=1} dx = \int_x p_\theta(x)dx = 1$
2. **Latent variables:** Mixtures of normalized objects $\alpha p_\theta(x) + (1-\alpha)p_{\theta'}(x)$:
   $\int_x \alpha p_\theta(x) + (1-\alpha)p_{\theta'}(x)dx = \alpha + (1-\alpha) = 1$
3. **Flows:** Construct p via bijection from a simple distribution and track volume change.

Energy Based Models

Energy-based models specify distributions of the form

$$p_\theta(x) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

The volume/normalization constant

$$Z(\theta) = \int \exp(f_\theta(\mathbf{x}))d\mathbf{x}$$

Energy Based Models

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

Pros:

1. extreme flexibility: can use pretty much any function $f_\theta(\mathbf{x})$ you want

Cons: Computing $Z(\theta)$ numerically is hard.

1. Sampling from $p_\theta(\mathbf{x})$ is in general hard
2. Evaluating and optimizing likelihood $p_\theta(\mathbf{x})$ is hard (learning is hard)

Application of Energy Based Models

Consider an energy-based model of the form:

$$p_\theta(\mathbf{x}) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

Given $\mathbf{x}$, $\mathbf{x}'$ evaluating $p_\theta(\mathbf{x})$ or $p_\theta(\mathbf{x}')$ requires $Z(\theta)$. But evaluating $\frac{p_\theta(\mathbf{x})}{p_\theta(\mathbf{x}')}$ doesn't:

$$\frac{p_\theta(\mathbf{x})}{p_\theta(\mathbf{x}')} = \exp(f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}'))$$

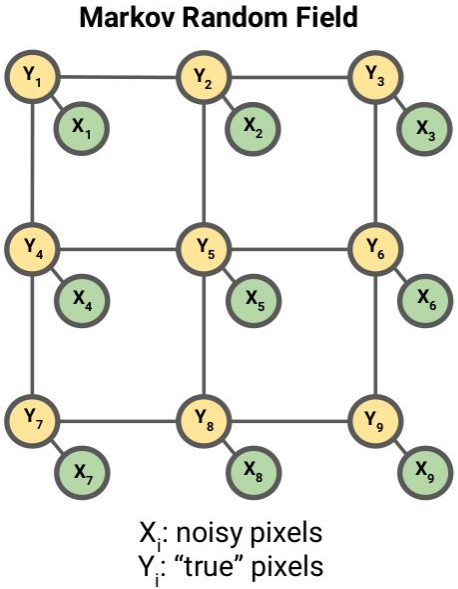
Example: Ising Model

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i \psi_i(x_i, y_i) + \sum_{(i,j) \in E} \psi_{ij}(y_i, y_j) \right)$$

- $\psi_i(x_i, y_i)$: the i-th corrupted pixel depends on the i-th original pixel
- $\psi_{ij}(y_i, y_j)$: neighboring pixels tend to have the same value

- There is a true image $\mathbf{y} \in \{0,1\}^{3\times 3}$, and a corrupted image $\mathbf{x} \in \{0,1\}^{3\times 3}$. We know $\mathbf{x}$, and want to somehow recover $\mathbf{y}$.

- How did the original image $\mathbf{y}$ look like? Solution: maximize $p(\mathbf{y}|\mathbf{x})$.



Learning Energy Based Models

In general, likelihood-based learning of energy-based models is intractable (it's hard to even compute the log likelihood!). We need approximations.

Approximations to the maximum likelihood learning:

- Variational inference: optimization-based approximations
- MCMC-based methods: sampling-based approximations

There also exist efficient methods that don't rely on likelihood:

- Score-based methods

Score-Based Methods

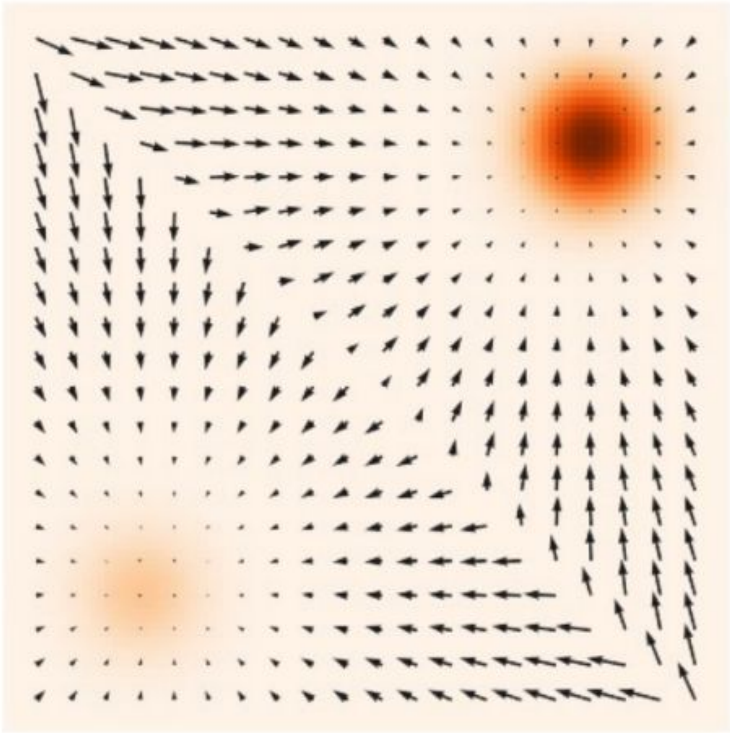
The (Stein) Score Function: Definition

Given a probability density $p_\theta(\mathbf{x})$, its (Stein) score function is defined as

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$$

- This is the gradient of the density with respect to the input.
- Note that this is **different** from the gradient $\nabla_\theta \log p_\theta(\mathbf{x})$ of the log-density with respect to the parameters θ (which is what we use in gradient descent).
- The score function points in the direction of increasing model probability.

Score function: 2D



No Need For Normalization

Suppose we have an energy-based model of the form

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

- Computing a score function of the model does not require $Z(\theta)$:

$$\nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = \nabla_\mathbf{x} f_\theta(\mathbf{x}) - \nabla_\mathbf{x} \log Z(\theta) = \nabla_\mathbf{x} f_\theta(\mathbf{x})$$

- **Idea**: Instead of learning $p_\theta(\mathbf{x})$, learn a model $\nabla_\mathbf{x} \log p_\theta(\mathbf{x})$ of the score function of the data $\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$!

How can we use a score-based model?

Langevin dynamics

Langevin dynamics is an MCMC sampling process that allows us to sample from $p_\theta(\mathbf{x})$ using only its score $\nabla_x \log p_\theta(\mathbf{x})$.

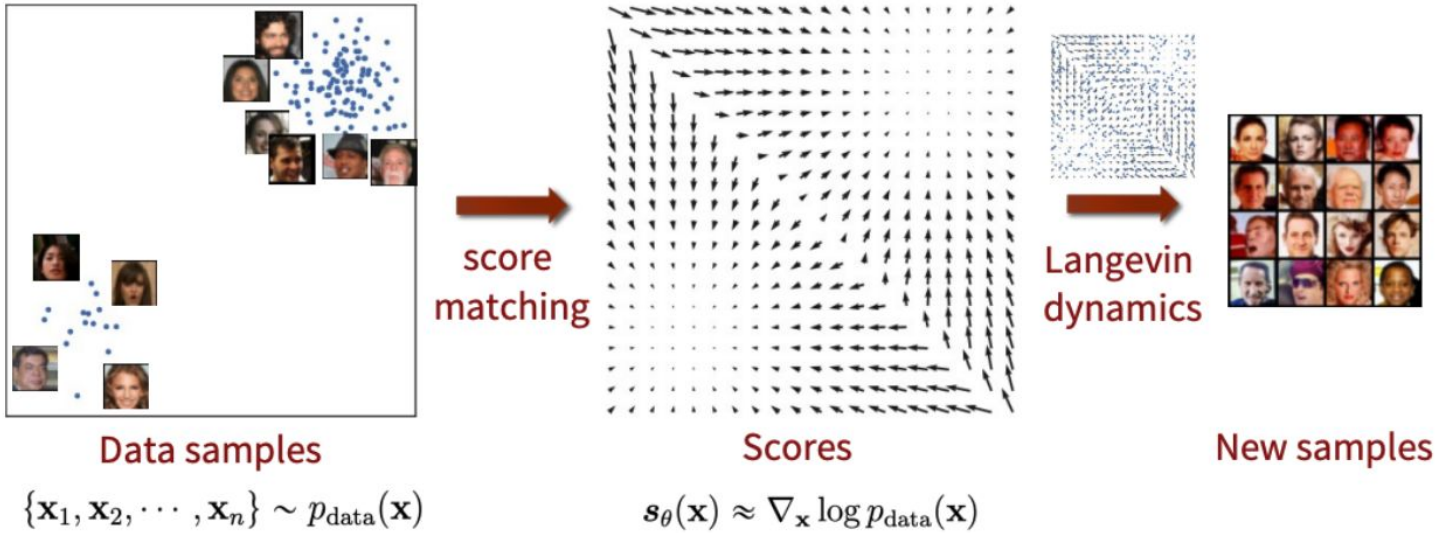
- Initialize $\mathbf{x}_0$ from a noise distribution
- For steps $t = 1, 2, \ldots, T$:
  - Sample a noise variable $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - $\mathbf{x}_t = \mathbf{x}_{t-1} + \frac{\alpha_t}{2} \nabla_x \log p_\theta(\mathbf{x}) + \sqrt{\alpha_t} \epsilon_t$

Note that:

- This process can be seen as gradient ascent on $\log p_\theta(\mathbf{x})$;
- It is guaranteed to produce samples from $p_\theta(\mathbf{x})$ when the step size $\alpha_t \to 0$ as $T \to \infty$ at the right annealing rate.

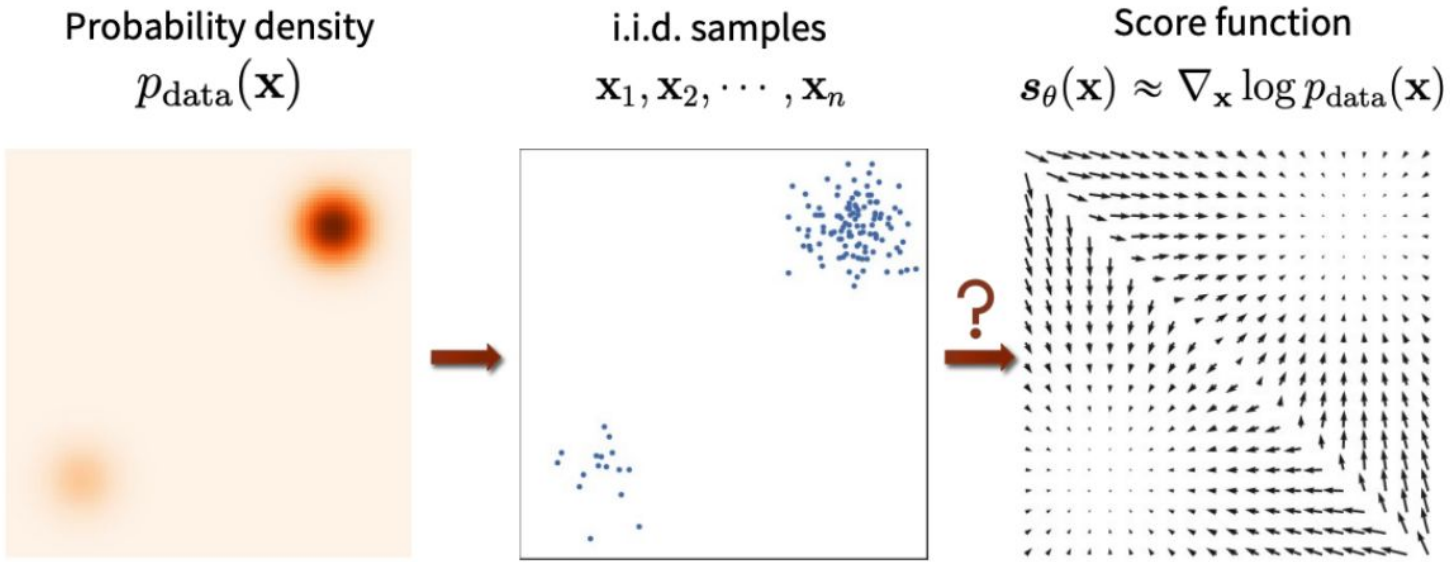
Score Function Modeling

Thus, our high-level strategy is to learn $s_\theta(\mathbf{x}) \approx \nabla_x \log p_\theta(\mathbf{x})$ from a set of samples, and then generate new $\mathbf{x}$ via Langevin dynamics.



Score Function Estimation

The idea of score function estimation is to learn a model $s_\theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$ of the score function from a dataset of sample points.

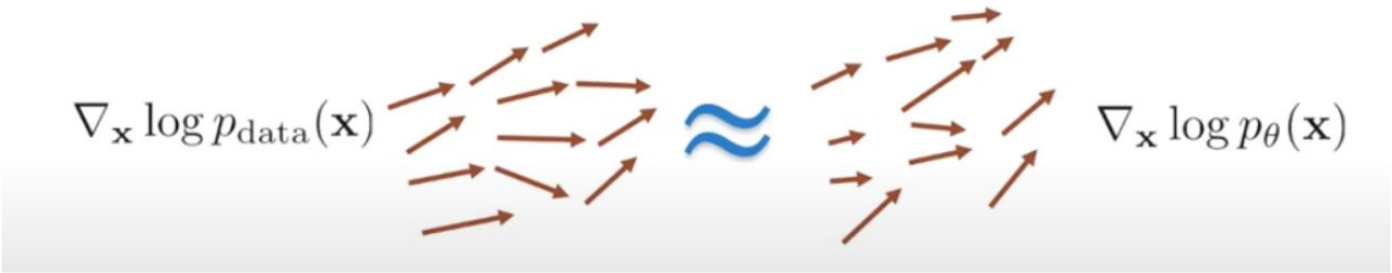


Score Function Estimation

- We are given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
- Our task is to estimate $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$
- Our model is a parameterized function $s_\theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$
- Our objective is that $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \approx s_\theta(\mathbf{x})$

How do we fit score functions?

Data and model scores are vector fields; we want them to be aligned:



Score Matching Via Fisher Divergence

- Formally, we use the Fisher divergence:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[||\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}) - s_\theta(\mathbf{x})||_2^2\right]$$

- It's the average Euclidean distance between two vectors over all $\mathbf{x}$
- The Fisher divergence is zero if and only if $\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}) \approx s_\theta(\mathbf{x})$

Score Matching

- We don't know $\log p_{\text{data}}(\mathbf{x})$, and much less its gradient.
- However, via a change of variables trick, we can obtain an equivalent objective (Hyvarinen, 2005)

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}||s_\theta(\mathbf{x})||_2^2 + \text{tr}\left(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})\right)\right]$$

- This is the *score matching* objective. It can be further approximated via Monte-Carlo given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$:

$$\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}||s_\theta(\mathbf{x}_i)||_2^2 + \text{tr}\left(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}_i)\right)\right]$$

Proof: 1D

We apply integration by parts as follows:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[||\nabla_{\mathbf{x}}\log p(\mathbf{x}) - s_\theta(\mathbf{x})||_2^2\right]$$
$$= \underbrace{\frac{1}{2}\int_{-\infty}^{\infty} p(x)(\nabla_{\mathbf{x}}\log p(x))^2 dx}_{\text{const}} + \frac{1}{2}\int_{-\infty}^{\infty} p(x)s_\theta(x)^2 dx - \int_{-\infty}^{\infty} p(x)\nabla_{\mathbf{x}}\log p(x)s_\theta(x)\ dx$$

We drop the first term and apply integration parts to the third term:

$$\int_{-\infty}^{\infty} p(x)\nabla_{\mathbf{x}}\log p(x)s_\theta(x)\ dx = \int_{-\infty}^{\infty} p(x)\frac{\nabla_{\mathbf{x}}p(x)}{p(x)}s_\theta(x)\ dx = \int_{-\infty}^{\infty} \nabla_{\mathbf{x}}p(x)s_\theta(x)\ dx$$
$$= \underbrace{p(x)s_\theta(x)|_{-\infty}^{\infty}}_{=0} - \int_{-\infty}^{\infty} p(x)\nabla_x s_\theta(x)dx$$

The first term is zero because we assume x has bounded support. Plugging the above into the first expression yields the desired result.

Hard to scale

- We parameterize $s_\theta(\mathbf{x})$ using a neural net and optimize the objective using gradient descent:

$$\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}||s_\theta(\mathbf{x}_i)||_2^2 + \operatorname{tr}\left(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}_i)\right)\right]$$

- However, this doesn't scale: gradient descent optimization requires us to compute nested backprop on each diagonal element of $\nabla_{\mathbf{x}} s_\theta(\mathbf{x})$:

$$\nabla_{\mathbf{x}} s_\theta(\mathbf{x}) = \begin{pmatrix} \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3} \end{pmatrix}$$

Sliced Score Matching

$$\frac{1}{2}\mathbb{E}_{\mathbf{v}\sim q(\mathbf{v})}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[(\mathbf{v}^\top \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) - \mathbf{v}^\top \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}))^2\right]$$

- Where $q(\mathbf{v})$ is a distribution over projection vectors $\mathbf{v}$ that we choose.
- As before, this is not practical because we don't know p_{data}. However, via a change of variables trick, we can obtain an equivalent objective:

$$\mathbb{E}_{\mathbf{v}\sim q(\mathbf{v})}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\mathbf{v}^\top \nabla_{\mathbf{x}}^2 \log p_\theta(\mathbf{x})\mathbf{v} + \frac{1}{2}(\mathbf{v}^\top \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}))^2\right]$$

Can be computed efficiently.

Denoising Score Matching

Another approach to train a score model - via denoising.

Tweedie's Formula:

Given a noisy data point **y** (of the clean data point **x**), and the MMSE estimate $\hat{x} = \mathbb{E}[x|y]$, then:

$$\nabla_y \log p_\sigma(y) = \frac{1}{\sigma^2}(\hat{x} - y)$$

$$where\ p_\sigma(y) = \int p_\sigma(y|x)\, p_{data}(x)\, dx.$$

Proof

$$\begin{aligned}
\nabla_y \log p_\sigma(y) &= \frac{1}{p_\sigma(y)} \frac{\partial}{\partial y} p_\sigma(y) \\
&= \frac{1}{p_\sigma(y)} \frac{\partial}{\partial y} \int p_\sigma(y|x)\, p(x)\, dx \\
&= \int \frac{p(x)}{p_\sigma(y)} \frac{\partial}{\partial y} \mathcal{N}\left(y \,|\, x, I\sigma^2\right) dx
\end{aligned}$$

The derivative of a Gaussian has quite a neat form, which will allow us to continue:

$$\begin{aligned}
\frac{\partial}{\partial y} \mathcal{N}\left(y \,|\, x, I\sigma^2\right) &= \frac{\partial}{\partial y} \frac{1}{Z} e^{-\frac{1}{2\sigma^2}\|x-y\|^2} \\
&= \frac{1}{Z} e^{-\frac{1}{2\sigma^2}\|x-y\|^2} \frac{\partial}{\partial y} \left(-\frac{1}{2\sigma^2}\|x-y\|^2\right) \\
&= \mathcal{N}\left(y \,|\, x, I\sigma^2\right) \frac{1}{\sigma^2}(x-y)
\end{aligned}$$

Proof

Substituting the derivative of a Gaussian:

$$\begin{aligned}
\nabla_y \log p_\sigma(y) &= \int \frac{p(x)\, p_\sigma(y|x)}{p_\sigma(y)} \frac{1}{\sigma^2}(x-y)\, dx \\
&= \frac{1}{\sigma^2} \int p(x|y)(x-y)\, dx \\
&= \frac{1}{\sigma^2} \left[\int x p(x|y)\, dx - y \int p(x|y)\, dx \right] \\
&= \frac{1}{\sigma^2} \left[\mathbb{E}_x[x|y] - y \right] = \frac{1}{\sigma^2}(\hat{x} - y)
\end{aligned}$$

Denoising Score Matching

Denoising score matching (Vincent, 2011) approximates the Fisher divergence between s_θ and $q_\sigma(\tilde{\mathbf{x}})$ as:

$$\frac{1}{2}\mathbb{E}_{\tilde{\mathbf{x}}\sim q_\sigma(\tilde{\mathbf{x}})}\left[||\nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}}) - s_\theta(\tilde{\mathbf{x}})||_2^2\right] = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p,\tilde{\mathbf{x}}\sim q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left[||\nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - s_\theta(\tilde{\mathbf{x}})||_2^2\right]$$

- $\nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ is easy to compute e.g., $\nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = -\frac{\mathbf{x}-\tilde{\mathbf{x}}}{\sigma^2}$ in the above example with Gaussian noise
- This is faster than slicing, but can only learn noised distributions.

Next week we will see how this forms the basis of diffusion models.