Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

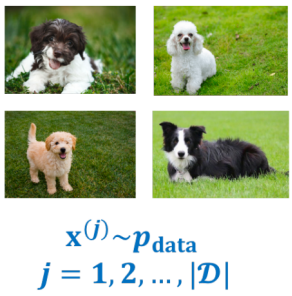
Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

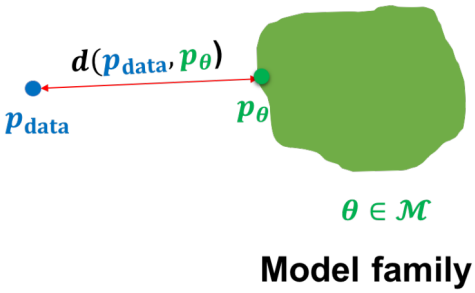
Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

GANs

1 Refresher





Model Families

- **Autoregressive Models:**

$$p_G(x) = \prod_{i=1}^{n} p_G(x_i \mid x_{<i})$$

- **Variational Autoencoders:**

$$p_G(x) = \int p_G(x, z)\, dz$$

- **Normalizing Flow Models:**

$$p_X(x; \theta) = p_Z\left(f_\theta^{-1}(x)\right) \left| \det \left(\frac{\partial f_\theta^{-1}(x)}{\partial x} \right) \right|$$

All the above families are trained by minimizing KL divergence $D_{KL}(p_{\text{data}} \| p_G)$ or equivalently maximizing likelihoods (or approximations).

a lots of this machinery kinda involves ways of setting up models such that you can evaluate likelihood easily.

$$\hat{\theta} = \arg\max_\theta \sum_{i=1}^{M} \log p_G(x_i), \quad x_1, x_2, \cdots, x_M \sim p_{\text{data}}(x)$$

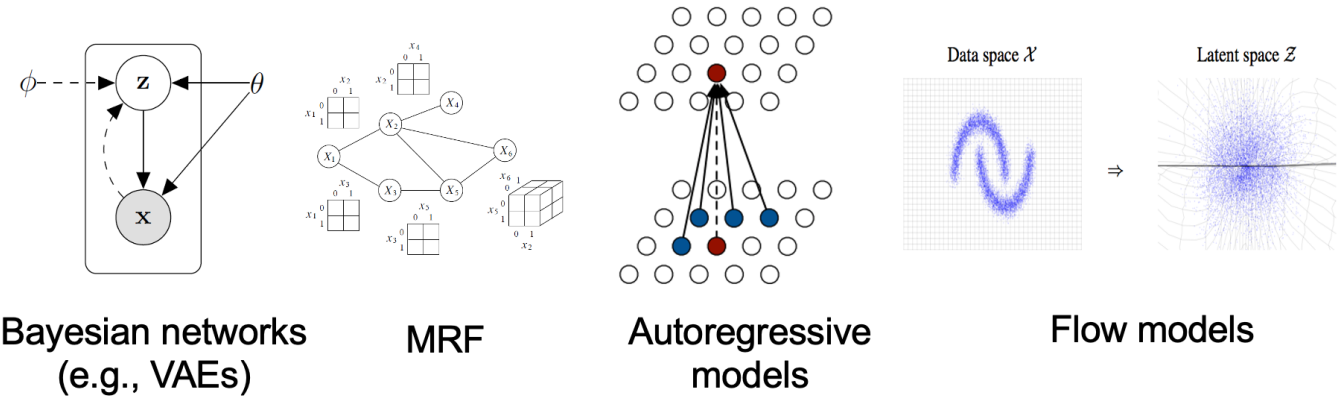
There are some good reasons why we have been using MLE objectives before:

- **Optimal statistical efficiency:**

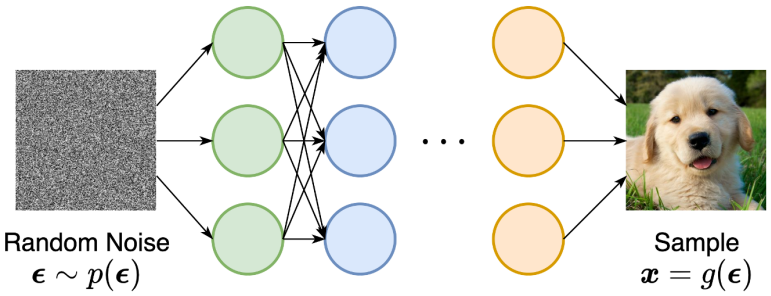
- Which means under some assumptions (are not true in practice) of a sufficient model capacity, such that there exists a unique $\theta^{\in}\mathcal{M}$ that satisfies $p_\theta = p_{\text{data}}$.
- The convergence of $\hat{\theta}$ to θ when $M \to \infty$ is the fastest among all statistical methods when using maximum likelihood training. (You are making the best possible use of your data)

- **Higher likelihood = better lossless compression.** We have seen before that compression is a reasonable kind of objective, which means if you can compress the data then you can predict what could happen with the data.

Is likelihood a good indicator of the quality of samples generated by the model?



Bayesian networks, Markov random fields (MRF), autoregressive models, and normalizing flow models are all examples of likelihood-based models. All these models represent the probability density or mass function of a distribution.



GAN is an example of implicit models. It implicitly represents a distribution over all objects that can be produced by the generator network.

2 Towards Likelihood-Free Learning

2.0.1 Case 1: Optimal Generative Model

- An **optimal generative model** will achieve the best:
  - **Sample quality:** The generated samples closely resemble the true data.
  - **Log-likelihood:** The model achieves the highest likelihood on the test data.
- For **imperfect models**, achieving high log-likelihood does not necessarily imply good sample quality, and vice-versa.
- This discrepancy between likelihood and sample quality was highlighted in Theis et al. (2016).

2.0.2 Case 2: Great Test Log-Likelihoods but Poor Samples

- Consider the following example of a **problematic model**: a discrete noise mixture model:

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

  - In this model, 99% of the samples are just noise.

- Taking the log of this model, we use the **logarithmic inequality**:
  - The logarithm is a monotonic function, which means:

$$\log(a + b) \geq \log(a) \quad \text{if } a > 0 \text{ and } b \geq 0.$$

  - Applying this inequality:

$$\log p_G(x) = \log\left(0.01 p_{\text{data}}(x) + 0.99 p_{\text{noise}}(x)\right) \geq \log\left(0.01 p_{\text{data}}(x)\right)$$

- Simplifying further:

$$\log\left(0.01 p_{\text{data}}(x)\right) = \log\left(p_{\text{data}}(x)\right) + \log\left(0.01\right)$$

  Since $\log(0.01) = -\log(100)$, this becomes:

$$\log p_G(x) \geq \log p_{\text{data}}(x) - \log 100$$

- For **expected likelihoods**, we derive the following bounds:
  - **Lower Bound:**

$$\mathbb{E}_{p_{\text{data}}}[\log p_G(x)] \geq \mathbb{E}_{p_{\text{data}}}[\log p_{\text{data}}(x)] - \log 100$$

  - **Upper Bound:** (via non-negativity of KL divergence)
    To compute the upper bound, we use the property of the **KL divergence**, which is always non-negative:

$$D_{\text{KL}}(p_{\text{data}} \| p_G) = \mathbb{E}_{p_{\text{data}}}\left[\log p_{\text{data}}(x) - \log p_G(x)\right] \geq 0.$$

    Thus, the **upper bound** is:

$$\mathbb{E}_{p_{\text{data}}}[\log p_{\text{data}}(x)] \geq \mathbb{E}_{p_{\text{data}}}[\log p_G(x)]$$

- As the dimension of x increases:
  - The absolute value of $\log p_{\text{data}}(x)$ increases proportionally.
  - However, the constant $\log 100$ remains fixed.

- This implies in very high dimensions (x has high dimension) , making the lower bound insignificant.:

$$\mathbb{E}_{p_{\text{data}}}[\log p_G(x)] \approx \mathbb{E}_{p_{\text{data}}}[\log p_{\text{data}}(x)]$$

  Which means the likelihood of this model which generates noise 99% of the time is pretty close to the best you can possible achieve.

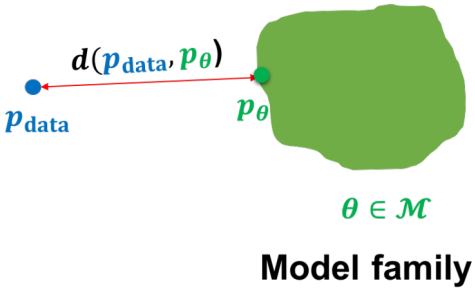
2.0.3 Case 3: Great Samples, Poor Test Log-Likelihoods

- Example: Memorizing the training set
  - Samples look **exactly** like the training set (cannot do better!).
  - Test set will have **zero probability** assigned (cannot do worse!).

- The above cases suggest that it might be useful to **disentangle** likelihoods and samples.

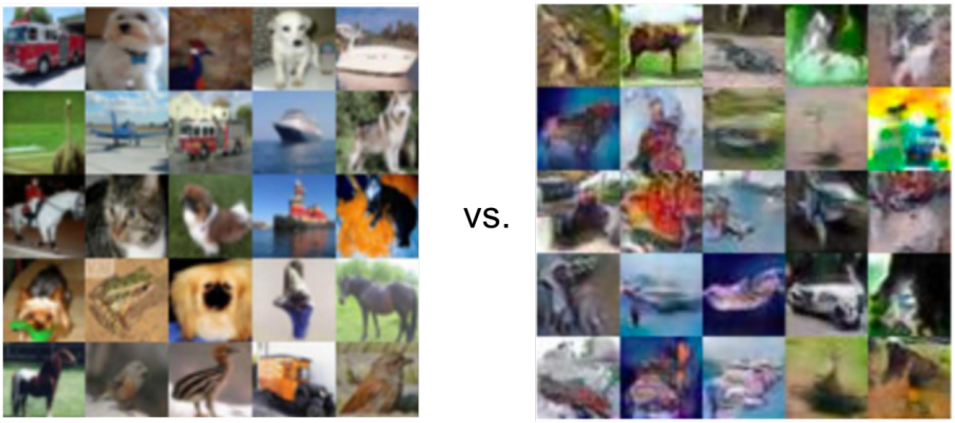
2.1 Likelihood-Free Learning

$\mathbf{x}^{(j)} \sim p_{\text{data}}$
$j = 1, 2, \ldots, |\mathcal{D}|$



- Consider objectives that do not depend directly on a likelihood function and not using KL divergence.
- To use alternative approach for comparing $d(p_{date}|p_G)$

2.1.1 Comparing Distributions via Samples



$S_1 = \{\mathbf{x} \sim P\}$ $S_2 = \{\mathbf{x} \sim Q\}$

- Given a finite set of samples from two distributions:

$$S_1 = \{x \sim P\}, \quad S_2 = \{x \sim Q\}$$

- How can we tell if these samples are from the same distribution? (i.e., $P = Q$?)

2.1.2 Two-Sample Tests

- Suppose we are given two sets of samples:

$$S_1 = x \sim P, \quad S_2 = y \sim Q.$$

A **two-sample test** aims to determine whether the samples (S_1) and (S_2) come from the same underlying distribution. This involves evaluating the following hypotheses:

  - **Null hypothesis ((H_0)):** The two distributions are identical, i.e., ($P = Q$).
  - **Alternative hypothesis** (H_1)**:** The distributions differ, i.e., ($P \neq Q$).

- The test relies on a **test statistic** T, which compares the two sets of samples S_1 and S_2. A commonly used statistic for comparing the difference in means is the two-sample **T-statistic**, defined as:

$$T(S_1, S_2) = \frac{\left| \frac{1}{n} \sum_{i=1}^{n} x_i - \frac{1}{m} \sum_{j=1}^{m} y_j \right|}{\sqrt{\frac{s_x^2}{n} + \frac{s_y^2}{m}}},$$

  where $n = |S_1|$ and $m = |S_2|$, and s_x^2 and s_y^2 are the sample variances of S_1 and S_2, respectively. This normalization by variance is critical for statistically valid inference.

- Assumptions for the t-test include independence of samples, approximate normality of the underlying distributions, and homogeneity of variances.

- Based on the value of T, we make a decision:
  - Compute a critical threshold T_α from the Student's t-distribution with $n + m - 2$ degrees of freedom, given a significance level α.
  - If $T > T_\alpha$, we reject the null hypothesis (H_0) and conclude that $P \neq Q$.
  - Otherwise, we fail to reject H_0 and conclude that there is insufficient evidence to claim the distributions differ.

- **Key observation:** The test statistic T is often **likelihood-free**, meaning it does not require explicit forms of P or Q. Instead, it relies only on observed samples S_1 and S_2. Common examples include the Kolmogorov–Smirnov, Mann–Whitney U test, or Maximum Mean Discrepancy (MMD).

2.2 Generative Modeling and Two-Sample Tests

- In generative modeling, we typically assume direct access to a set of real data samples:

$$S_1 = \mathcal{D} = \{x \sim p_{\text{data}}\}.$$

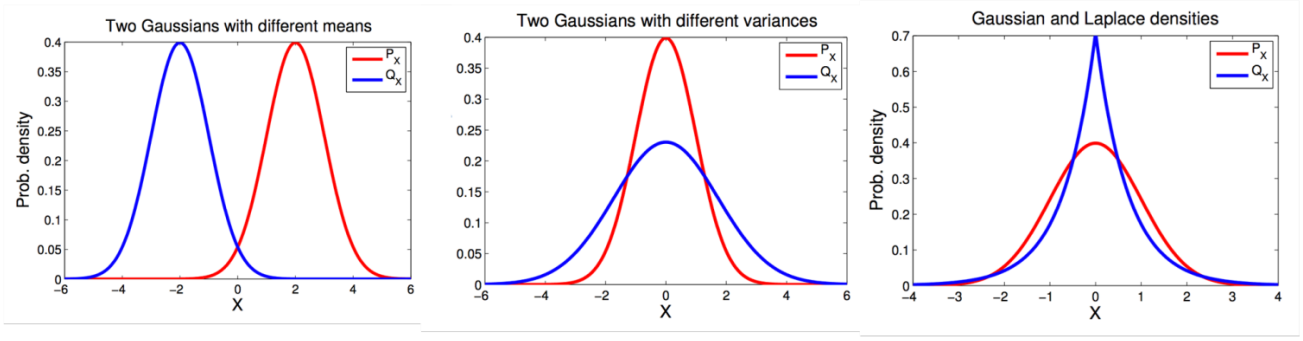
- Additionally, we have a model that defines a distribution p_G from which we can efficiently generate samples. This gives us another set of samples:

$$S_2 = \{z \sim p_G\}.$$

- A **two-sample test** can then be used to compare these two sets of samples S_1 and S_2 to evaluate whether the model distribution p_G matches the true data distribution p_{data}.

- **Alternate perspective:** Instead of directly measuring a conventional distance (e.g., KL divergence) between the distributions, we train the generative model to minimize a two-sample test objective. This objective measures how distinguishable the samples S_1 and S_2 are.

2.3 Two-Sample Test via a Discriminator



Source: Arthur Gretton

- Designing a two-sample test in high-dimensional spaces is particularly challenging due to the complexity of the data and the curse of dimensionality.

- In the context of generative modeling, we know that:
  - S_1 represents samples from the real data distribution p_{data}.
  - S_2 represents samples from the model distribution p_G.
- **Key idea:** Learn a test statistic (e.g., via a neural network) that **maximizes a notion of distance** between the two sets of samples S_1 and S_2. This discriminator effectively identifies differences between the two distributions.

3 Discrimination as a Signal

We aim to generate new samples $\{x_j^*\}$ from the same distribution as a set of real training data $\{x_i\}$. Each sample x_j^* is generated by:

$$x_j^* = g(z_j, \theta)$$

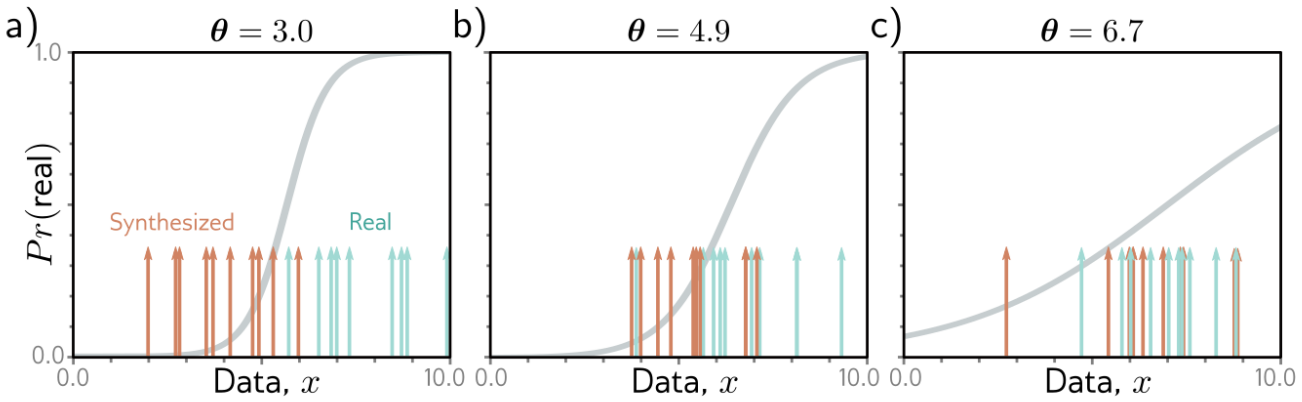
where z_j is drawn from a standard normal distribution, and θ parameterizes the generator.

3.1 Learning Process

- The goal is to optimize θ so that the generated samples $\{x_j^*\}$ become statistically indistinguishable from the real data $\{x_i\}$.
- A **discriminator network** $f(x, \phi)$, parameterized by ϕ, is introduced to classify whether a sample is real or generated.
- If the discriminator cannot distinguish between the real and generated samples, the generator has succeeded.

Example: We start with a training set $\{x_i\}$ of real 1D examples. A different batch of ten of these examples $\{x_i\}_{i=1}^{10}$ is shown in each panel (cyan arrows). To create a batch of samples $\{x_j^*\}$, we use the simple generator: $x_j^* = g(z_j, \theta)$

where latent variables zj are drawn from a standard normal distribution, and the parameter θ translates the generated samples along the x-axis.



At initialization, $\theta = 3.0$, and the generated samples (orange arrows) lie to the left of the real examples (cyan arrows). The discriminator is trained to distinguish the generated samples from the real examples (the sigmoid curve indicates the probability that a data point is real). During training, the generator parameters θ are manipulated to increase the probability that its samples are classified as real. Here, this means increasing θ so that the samples move rightwards where the sigmoid curve is higher.

We alternate between updating the discriminator and the generator. Figures **c** show two iterations of this process. It gradually becomes harder to classify the data, so the impetus to change θ becomes weaker (i.e., the sigmoid becomes flatter). At the end of the process, there is no way to distinguish the two sets of data; the discriminator, which now has chance performance, is discarded, and we are left with a generator that makes plausible samples.

4 Generative Adversarial Networks (GANs)

4.1 Overview

- A **two-player minimax game** between a generator and a discriminator:

$$\min_G \max_D V(G, D)$$

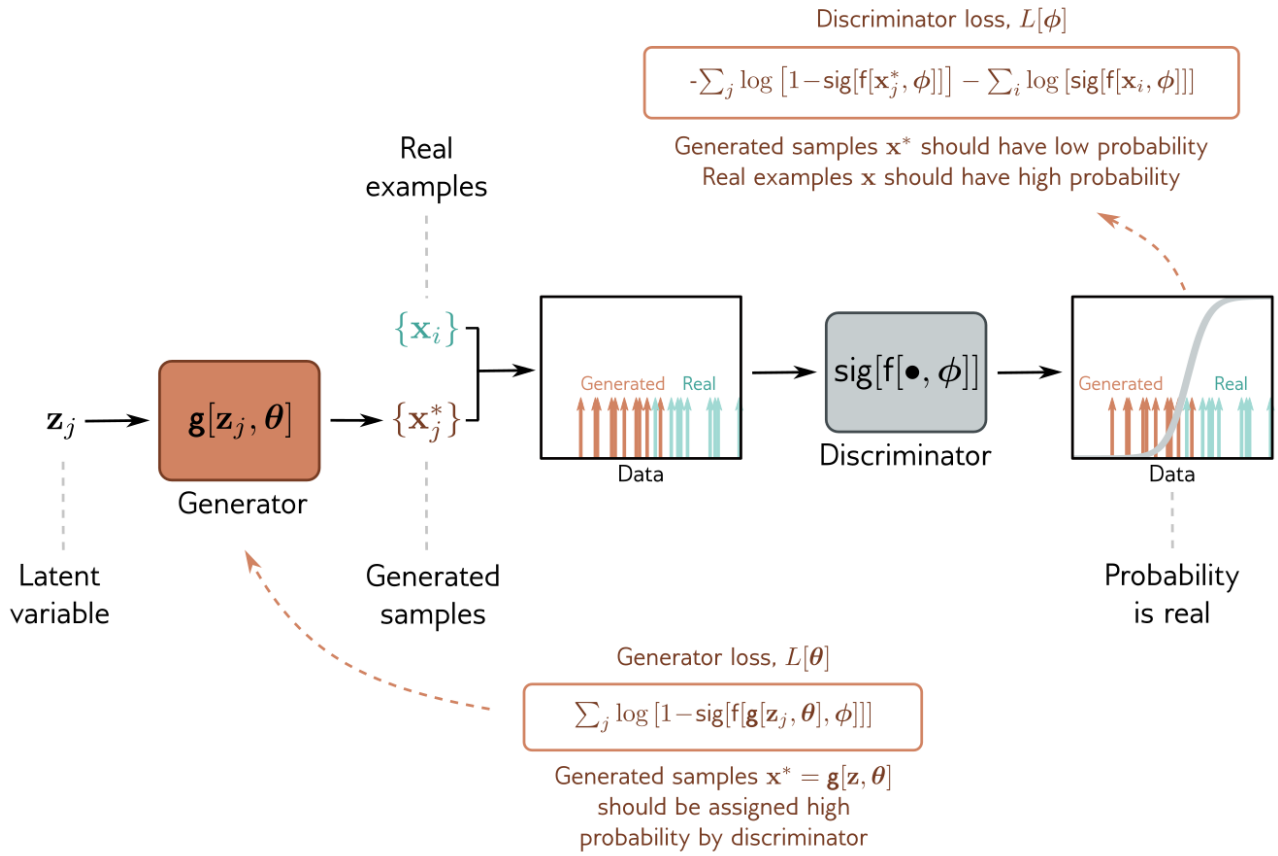
4.1.1 Generator

- Directed, latent variable model with a deterministic mapping between $\mathbf{z}$ and $\mathbf{x}$ given by G_θ:
  - Sample $\mathbf{z} \sim p(\mathbf{z})$, where $p(\mathbf{z})$ is a simple prior (e.g., Gaussian).
  - Set $\mathbf{x} = G_\theta(\mathbf{z})$.
- Similar to a flow model, but the mapping G_θ need not be invertible.
- Distribution over $p_G(\mathbf{x})$ is implicitly defined (no likelihood!).
- Minimizes a two-sample test objective (in support of the null hypothesis $p_{\text{data}} = p_G$).

4.1.2 Discriminator

- Any binary classifier D_ϕ (e.g., neural network) which tries to distinguish **"real"** ($y = 1$) samples from the dataset and **"fake"** ($y = 0$) samples generated from the model.
- **Test statistic:** $-$loss of the classifier.
  - **Low loss:** Real and fake samples are easy to distinguish (different).
  - **High loss:** Real and fake samples are hard to distinguish (similar).
- **Goal:** Maximize the two-sample test statistic (in support of the alternative hypothesis $p_{\text{data}} \neq p_G$), or equivalently minimize classification loss.

4.2 GAN Loss Function and Training Procedure



4.2.1 Training Objective for Discriminator

- **Objective:**

$$\max_{\phi} V(G,D) = \mathbb{E}_{x\sim p_{\text{data}}}[\log[\text{sig}(f[x,\phi])]] + \mathbb{E}_{z\sim p_G}[\log[1-\text{sig}(f[g[z,\theta],\phi])]].$$

Batch Approximation

$$\approx \sum_{i\in S_1} \log[\text{sig}(f[x_i,\phi])] + \sum_{j\in S_2} \log[1-\text{sig}(f[g[z_j,\theta],\phi])]\,.$$

$$\equiv$$

$$\min L[\phi] = -\sum_j \log[1-\text{sig}(f[g[z_j,\theta],\phi])] - \sum_i \log[\text{sig}(f[x_i,\phi])]$$

- For a fixed generative model p_G, the discriminator performs binary classification using the cross-entropy objective:
  - Assign probability 1 to true data points $x \sim p_{\text{data}}$ (in set S_1).
  - Assign probability 0 to fake samples $z \sim p_G$ (in set S_2).
- **Optimal Discriminator:**

$$D_\phi^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_G(x)}.$$

- **Sanity Check:** If $p_G = p_{\text{data}}$, the classifier cannot do better than chance:

$$D_\theta^*(x) = \frac{1}{2}.$$

- For the optimal discriminator D^*, we have:

$$V(G, D^*) = \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \left(\frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)} \right) \right] + \mathbb{E}_{z \sim p_G} \left[\log \left(\frac{p_G(z)}{p_{\text{data}}(z) + p_G(z)} \right) \right].$$

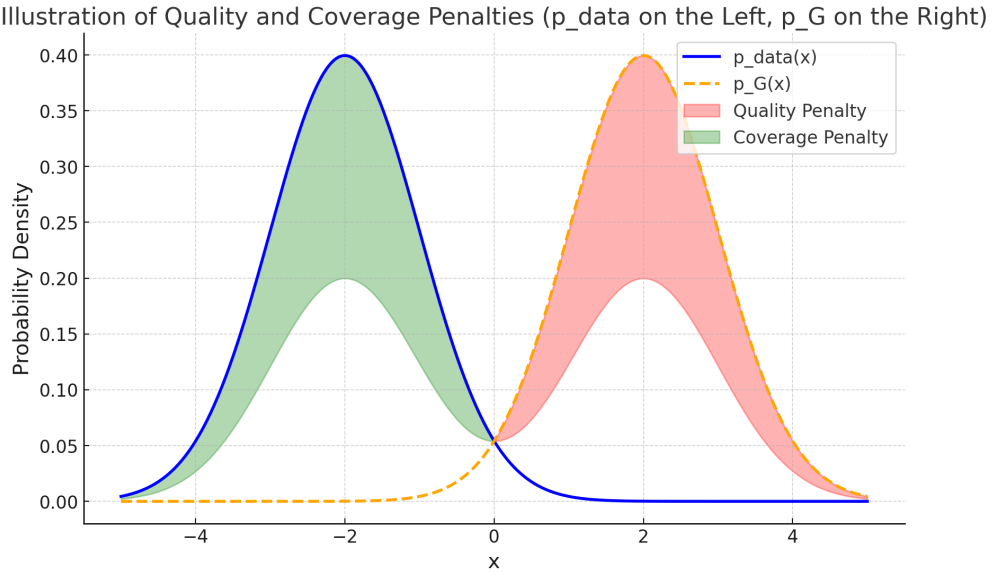
- Using simplifications, the final objective involves the Jensen-Shannon Divergence (JSD):

$$= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{\frac{p_{\text{data}}(x) + p_G(x)}{2}} \right] + \mathbb{E}_{x \sim p_G} \left[\log \frac{p_G(x)}{\frac{p_{\text{data}}(x) + p_G(x)}{2}} \right] - \log 4 \rightarrow \text{Normalized}$$

$$\underbrace{D_{\text{KL}} \left(p_{\text{data}}, \frac{p_{\text{data}} + p_G}{2} \right) + D_{\text{KL}} \left(p_G, \frac{p_{\text{data}} + p_G}{2} \right)}_{2 \times \text{ Jensen-Shannon Divergence (JSD)}} - \log(4)$$

$$= 2D_{\text{JSD}}[p_{\text{data}}, p_G] - \log 4$$

Interpretation:



- The first term (**coverage**) penalizes regions on the left where the true data distribution $p_{\text{data}}(x)$ is high, but the model fails to generate samples x^* in these regions. The penalty arises because $p_{\text{data}}(x)$ is large while $p_G(x)$ is small, resulting in insufficient coverage by the model. The penalty is reduced when $p_{\text{data}}(x)$ is high and $\frac{p_G(x) + p_{\text{data}}(x)}{2}$ is also high.
- The second term (**quality**) penalizes regions on the right where the model generates samples x^* with high probability $p_G(x^*)$ but where the true data distribution $p_{\text{data}}(x)$ is low. This happens when $p_G(x^*)$ is large, but $p_{\text{data}}(x)$ is small, causing the model to generate samples in regions not supported by the data. The penalty is reduced when $p_G(x^*)$ is high and $\frac{p_G(x) + p_{\text{data}}(x)}{2}$ is also high.

4.2.2 Jensen-Shannon Divergence

- Also called the **symmetric KL divergence**:

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

Properties

- $D_{\text{JSD}}[p, q] \geq 0$

- $D_{\text{JSD}}[p, q] = 0$ if and only if $p = q$
- $D_{\text{JSD}}[p, q] = D_{\text{JSD}}[q, p]$
- $\sqrt{D_{\text{JSD}}[p, q]}$ satisfies the triangle inequality → Jensen-Shannon Distance

Optimal Generator for JSD/Negative Cross-Entropy GAN

$$p_G = p_{\text{data}}$$

Optimal Discriminator and Generator

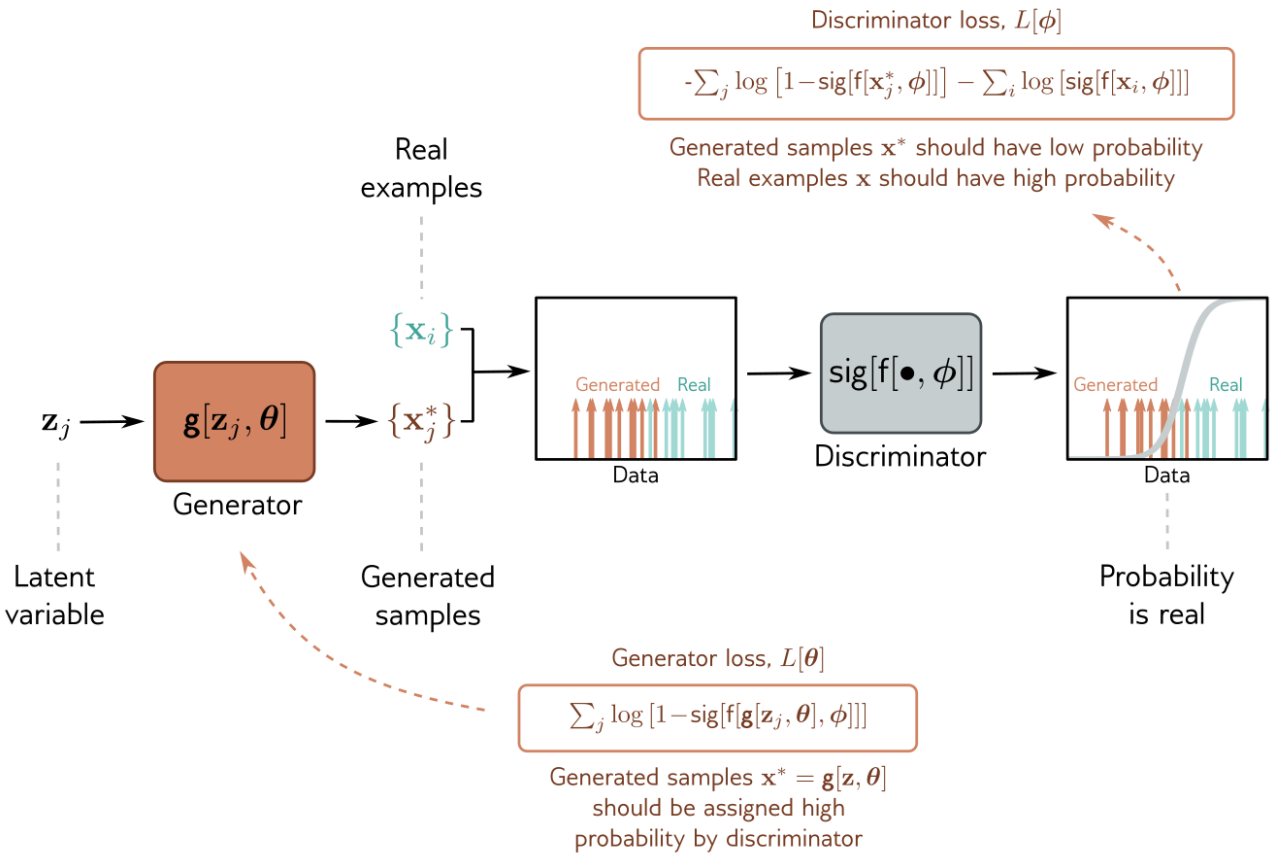
- For the optimal discriminator $D^*_{G^*}(\cdot)$ and generator $G^*(\cdot)$, we have:

$$V(G^*, D^*_{G^*}(x)) = -\log 4$$

4.2.3 Training Objective for Generator

$$\min_{\theta} V(G, D) = \mathbb{E}_{z \sim p_G}[\log\left[1 - \text{sig}(f[g[z, \theta], \phi])\right]$$

5 GAN Loss Function and Training Procedure



5.1 Loss Functions

The total loss function for the GAN is expressed as a minimax game:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))].$$

To simplify training, we divide the loss into two separate objectives:

$$L[\phi] = -\sum_j \log\left[1 - \text{sig}(f[g[z_j, \theta], \phi])\right] - \sum_i \log\left[\text{sig}(f[x_i, \phi])\right],$$

$$L[\theta] = \sum_j \log\left[1 - \text{sig}(f[g[z_j, \theta], \phi])\right].$$

5.2 Training Algorithm

At each step:

- Draw a minibatch of m real data samples $\{x^{(i)}\}_{i=1}^m \sim \mathcal{D}$.
- Draw a minibatch of m noise samples $\{z^{(i)}\}_{i=1}^m \sim p_z$ and generate fake samples:

$$x_j^* = g[z_j, \theta].$$

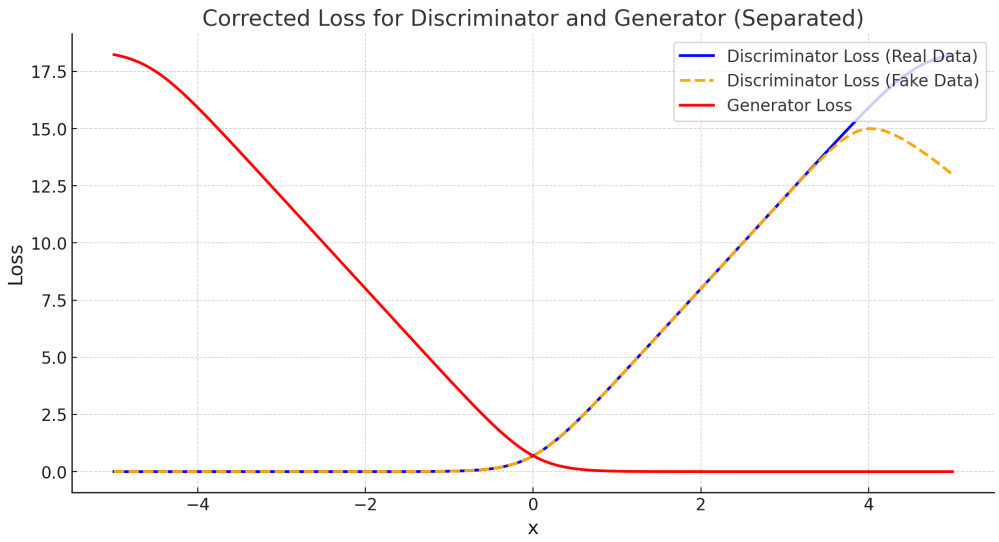
- Perform gradient updates:
  - Update discriminator parameters ϕ by minimizing $L[\phi]$ (stochastic gradient ascent):

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

  - Update generator parameters θ by minimizing $L[\theta]$ (stochastic gradient descent):

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

Repeat for a fixed number of epochs.



Loss Components:

- **Discriminator Loss (Real Data)**:

$$\mathcal{L}_{D,\text{real}} = -\log(D(x))$$

  This term measures how well the discriminator classifies real samples $x \sim p_{\text{data}}(x)$ as real. The loss decreases as $D(x) \to 1$, meaning the discriminator is confident that the samples are real.

- **Discriminator Loss (Fake Data)**:
$$\mathcal{L}_{D,\text{fake}} = -\log(1 - D(x))$$
This term measures how well the discriminator classifies fake samples $x \sim p_G(x)$ as fake. The loss decreases as $D(x) \to 0$, meaning the discriminator is confident that the samples are fake.

- **Generator Loss**:
$$\mathcal{L}_G = -\log(D(x))$$
This term measures how well the generator fools the discriminator into classifying fake samples $x \sim p_G(x)$ as real. The generator minimizes this loss, which decreases as $D(x) \to 1$, meaning the discriminator is less confident in distinguishing fake samples from real samples.

Key Observations:

- The **discriminator loss for real data** ($\mathcal{L}_{D,\text{real}}$) is small where $p_{\text{data}}(x)$ is high because the discriminator correctly identifies real samples.
- The **discriminator loss for fake data** ($\mathcal{L}_{D,\text{fake}}$) is small where $p_G(x)$ is low because the discriminator correctly identifies fake samples.
- The **generator loss** ($\mathcal{L}_G$) is small where $p_G(x)$ aligns with $p_{\text{data}}(x)$, meaning the generator successfully fools the discriminator.

5.3 Key Insights

- The discriminator learns to distinguish real from fake samples by minimizing $L[\phi]$.
- The generator learns to "fool" the discriminator by minimizing $L[\theta]$.
- At convergence:
  - The generator distribution $p_G(x)$ aligns with the data distribution $p_{\text{data}}(x)$.
  - The discriminator output becomes 0.5 (chance performance), indicating indistinguishable distributions.
- This corresponds to a **Nash equilibrium**, where neither the generator nor the discriminator can improve unilaterally.

5.4 Alternating Optimization in GANs

The optimization alternates between:

- Maximizing $V(G, D)$ with respect to the discriminator (D).
- Minimizing $V(G, D)$ with respect to the generator (G).

Over time, the generator adapts to produce samples that align with the real data distribution $p_{\text{data}}(x)$, and the discriminator becomes less effective at distinguishing real from fake samples.

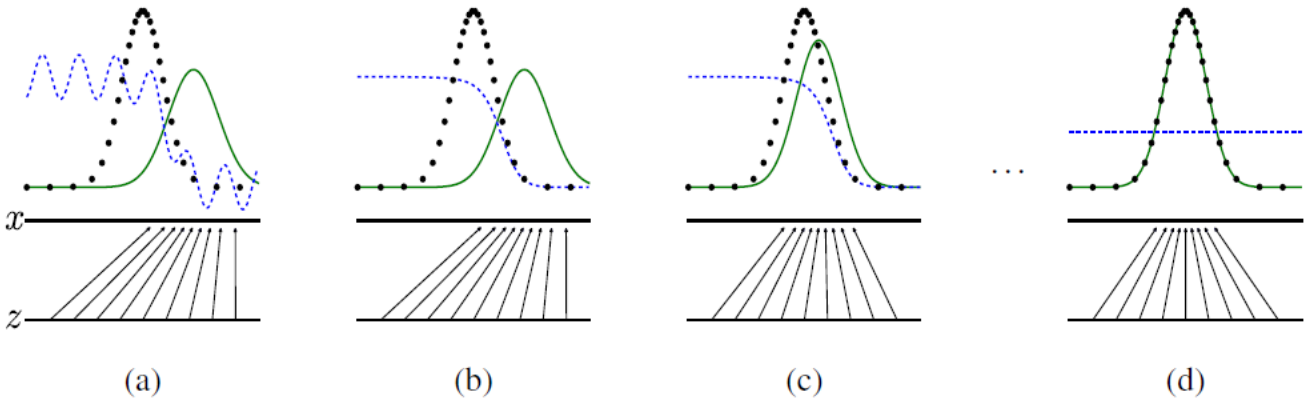


Figure Summary: The figure illustrates the training process of GANs, where the discriminator D (blue dashed line) distinguishes between the data distribution p_x (black dotted line) and the generator distribution p_g (green solid line). The training alternates between updating D to improve classification and G to align p_g with p_x. At convergence, $p_g = p_x$ and $D(x) = 0.5$ across the domain, indicating the discriminator is unable to distinguish between real and generated samples.

6 GAN's Examples

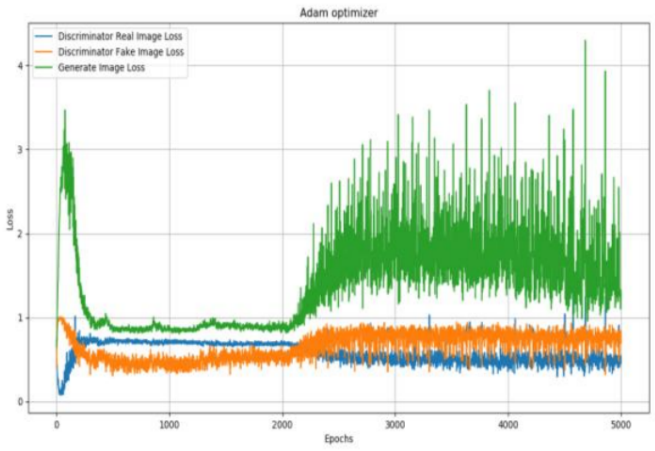
7 Challenges in Training GANs

Generative Adversarial Networks (GANs) have been successfully applied to a variety of domains and tasks. However, training GANs in practice poses several significant challenges:

- **Unstable optimization:** The adversarial nature of GAN training often leads to instability in convergence.
- **Mode collapse:** The generator may produce a limited variety of outputs, failing to capture the diversity of the data distribution.
- **Evaluation:** Quantitatively evaluating GAN performance remains a challenging task due to the lack of standardized metrics.

To address these challenges, numerous "bag of tricks" have been developed to stabilize and improve GAN training.

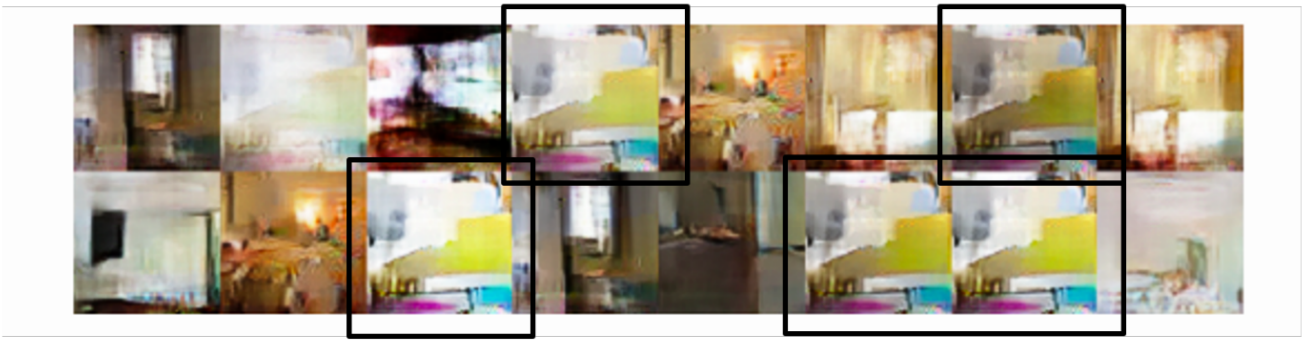
7.1 Optimization Challenges in GANs



- **Theorem (informal):** If the generator updates are made in function space and the discriminator is optimal at every step, then the generator is guaranteed to converge to the data distribution.
- **Unrealistic assumptions:** This theorem assumes ideal conditions that are rarely achievable in practice.
- **Practical observations:** In reality, the generator and discriminator losses often exhibit oscillatory behavior, making GAN training unstable and challenging.
- No robust stopping criteria in practice (unlike likelihood based learning)

7.2 Mode Collapse in GANs

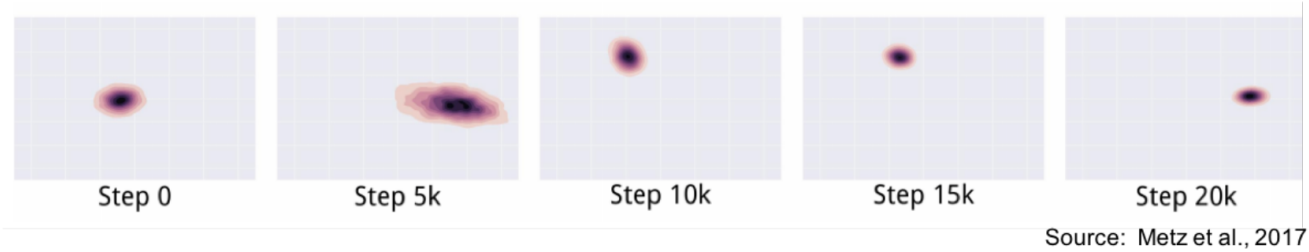
- GANs are notorious for suffering from **mode collapse**.
- Intuitively, mode collapse refers to the phenomenon where the generator collapses to producing only one or a few samples, ignoring the diversity in the data distribution (commonly referred to as "modes").



Arjovsky et al., 2017

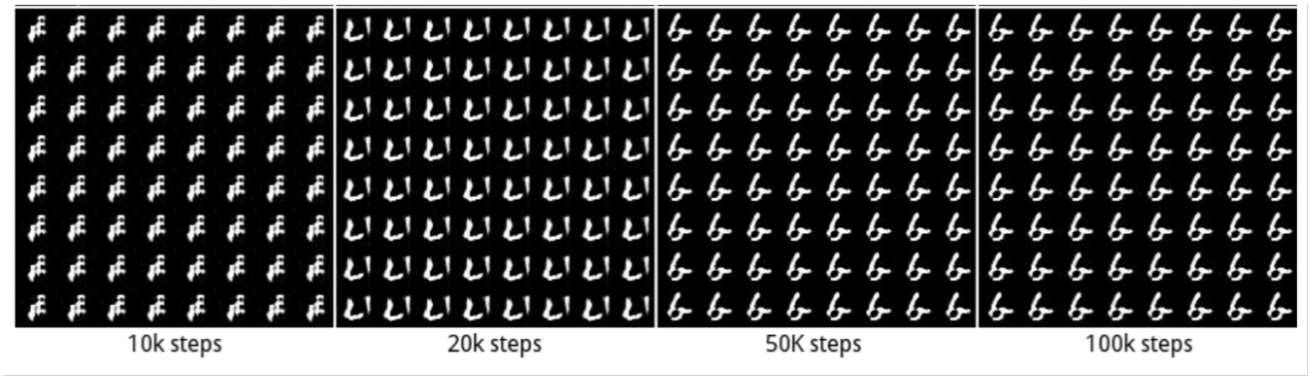
7.2.1 Illustration of Mode Collapse

- The target distribution is often a mixture of Gaussians.
- During training, the generator's distribution keeps oscillating between different modes, failing to capture the full diversity of the target distribution.



Source: Metz et al., 2017

7.2.2 Addressing Mode Collapse

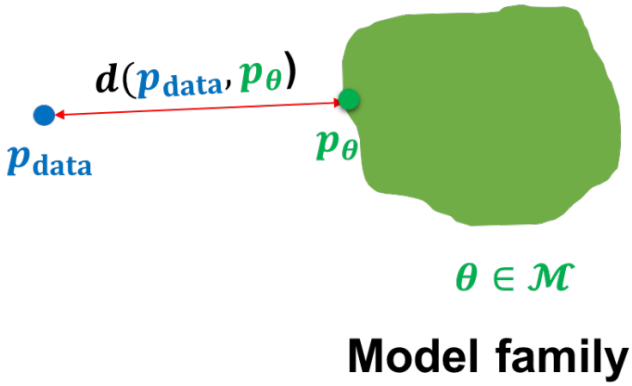


Source: Metz et al., 2017

- Fixes for mode collapse are mostly empirically driven, including:
  - Alternate architectures,
  - Adding regularization terms,
  - Injecting small noise perturbations, etc.
- For practical tips, refer to Soumith Chintala's GAN Hacks.

8 Beyond KL and Jensen-Shannon Divergence

$$\mathbf{x}^{(j)} \sim p_{\text{data}}$$
$$j = 1, 2, \ldots, |\mathcal{D}|$$



- What choices do we have for $d(\cdot)$?
  - **KL divergence:** Used in Autoregressive Models and Flow Models.
  - **Jensen-Shannon divergence:** A scaled and shifted version forms the original GAN objective.
- We will look into two new approaches:
  - **f-divergence** which defines f-GAN.
  - **Wasserstein distance** which defines W-GAN.

9 f-GAN

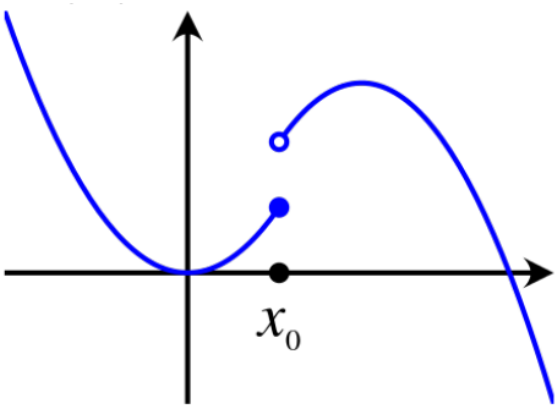
9.1 f-divergences

While traditional GANs rely on the Jensen-Shannon divergence to measure the difference between real and generated distributions, this approach has limitations, particularly when distributions are disjoint, leading to vanishing gradients. To address these challenges, f-divergences offer a more flexible framework, enabling alternative loss functions that can stabilize training and improve performance in diverse scenarios.

Given two densities p and q, the f-divergence is defined as:

$$D_f(p, q) = \mathbb{E}_{x \sim q}\left[f\left(\frac{p(x)}{q(x)}\right)\right],$$

where f is a convex, lower semicontinuous function with $f(1) = 0$.



Jensen's inequality:

$$\mathbb{E}_{x\sim q}\left[f\left(\frac{p(x)}{q(x)}\right)\right] \geq f\left(\mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}\right]\right) = f\left(\int q(x)\frac{p(x)}{q(x)}\right) = f\left(\int p(x)\right) = f(1) = 0.$$

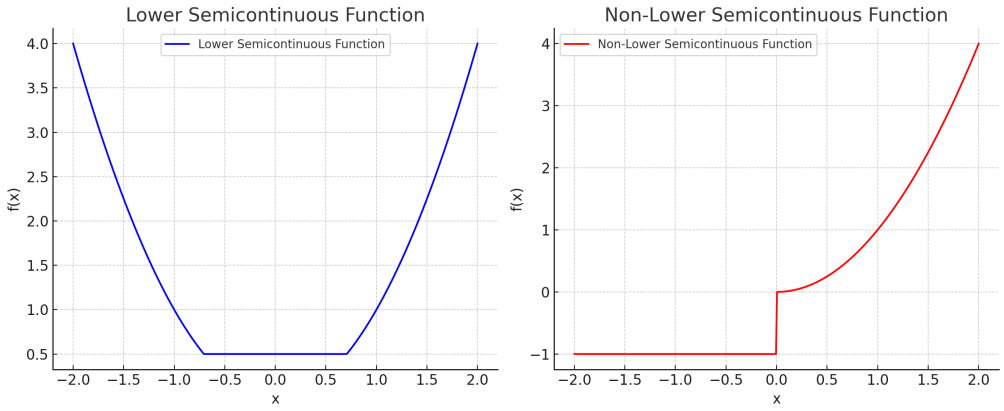
This indicates that f−divergence is none negative for any choice of p and q. This means if p and q are equal then the $D_f(p,q) = 0$. In general is not going to be 0 but has the minimum of 0 which is great for a minimization objective function.

9.1.1 What is Lower Semicontinuity?

Lower semicontinuity is a mathematical property that ensures the function does not jump abruptly downwards. At any given point x_0, the value of the function $f(x_0)$ is at least as low as the limit of $f(x)$ for points approaching x_0. Formally:

$$\liminf_{x\to x_0} f(x) \leq f(x_0).$$

$\liminf x$ captures the smallest "local minimum" that the function approaches (from both left and right) as x gets arbitrarily close to x_0.
Lower semicontinuity is crucial for stability because it ensures that small perturbations in the input (e.g., in the ratio $\frac{p(x)}{q(x)}$) do not cause large, downward discontinuities in f, making the divergence well-defined and robust. This property guarantees that f-divergences remain well-behaved, even in edge cases where the densities $p(x)$ and $q(x)$ have negligible overlap or diverge significantly. By ensuring smooth changes in the divergence values during updates, lower semicontinuity provides a solid foundation for stable optimization in GAN training.



Left Plot: Semicontinuous Function The function $\max(x^2, 0.5)$ is an example of a lower semicontinuous function. It is continuous and does not exhibit abrupt downward jumps. At any point x_0, the function value $f(x_0)$ is at least as large as the limit inferior of $f(x)$ as x approaches x_0:

$$\liminf_{x\to x_0} f(x) \leq f(x_0).$$

This property ensures the function behaves stably under optimization.

Right Plot: Non-Lower Semicontinuous Function The function if $x < 0, -1;$ else x^2 illustrates a violation of lower semicontinuity. It exhibits an abrupt downward jump at $x = 0$, where $f(x_0 = 0) = -1$ is less than the limit of $f(x)$ as x approaches 0 from the right ($0^2 = 0$):

$$\liminf_{x\to x_0} f(x) \leq f(x_0).$$

Such discontinuities can lead to instability during optimization, making the function less robust for applications.

9.1.2 Examples of f-divergences

- Many more f-divergences exist, as shown in the table below:

Name	$\mathbf{D_f(P\|Q)}$	$\mathbf{f(u)}$
Total variation	$\frac{1}{2}\int \|p(x)-q(x)\|dx$	$\frac{1}{2}\|u-1\|$
Kullback-Leibler	$\int p(x)\log\frac{p(x)}{q(x)}dx$	$u\log u$
Reverse KL	$\int q(x)\log\frac{q(x)}{p(x)}dx$	$-\log u$
Pearson χ^2	$\int \frac{(p(x)-q(x))^2}{q(x)}dx$	$(u-1)^2$
Neyman χ^2	$\int \frac{(p(x)-q(x))^2}{p(x)}dx$	$\frac{(1-u)^2}{u}$
Squared Hellinger	$\int \left(\sqrt{p(x)}-\sqrt{q(x)}\right)^2 dx$	$(\sqrt{u}-1)^2$
Jeffrey	$\int (p(x)-q(x))\log\frac{p(x)}{q(x)}dx$	$(u-1)\log u$
Jensen-Shannon	$\frac{1}{2}\int \left[p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}\right]dx$	$-(u+1)\log\frac{1+u}{2}+u\log u$
JS-weighted	$\int \left[\pi p(x)\log\frac{p(x)}{\pi p(x)+(1-\pi)q(x)}+(1-\pi)q(x)\log\frac{q(x)}{\pi p(x)+(1-\pi)q(x)}\right]dx$	$\pi u\log u-(1-\pi+\pi u)\log(1-\pi+\pi u)$
GAN	$\int p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}dx-\log(4)$	$u\log u-(u+1)\log(u+1)$
α-divergence	$\frac{1}{\alpha(\alpha-1)}\int \left(p(x)\left[\left(\frac{q(x)}{p(x)}\right)^\alpha-1\right]-\alpha(q(x)-p(x))\right)dx$	$\frac{1}{\alpha(\alpha-1)}\left(u^\alpha-\alpha(u-1)-1\right)$

Table 1: Examples of f-divergences and their corresponding functions. Source: Nowozin et al., 2016.

9.2 Training with f-divergences

- Given p_{data} and p_θ, we could minimize $D_f(p_\theta, p_{\text{data}})$ or $D_f(p_{\text{data}}, p_\theta)$ as learning objectives.
- These divergences are non-negative and equal to zero if $p_\theta = p_{\text{data}}$.
- However, both depend on the density ratio, which is unknown.

$$D_f(p_\theta, p_{\text{data}}) = \mathbb{E}_{x\sim p_{\text{data}}}\left[f\left(\frac{p_\theta(x)}{p_{\text{data}}(x)}\right)\right] \quad \text{(approx. w. samples, unknown ratio)}$$

$$D_f(p_{\text{data}}, p_\theta) = \mathbb{E}_{x\sim p_\theta}\left[f\left(\frac{p_{\text{data}}(x)}{p_\theta(x)}\right)\right] \quad \text{(approx. w. samples, unknown ratio)}$$

The density ratio (e.g., $\frac{p_\theta(x)}{p_{\text{data}}(x)}$) is unknown because:

- $p_{\text{data}}(x)$: The real data distribution is typically unknown; we only have access to samples from it (empirical p_{data}).
- $p_\theta(x)$: The generator's model density is rarely available explicitly, as it depends on sampling and transformations from the latent space.

So we need to rewrite f-divergences as a two-sample test objective for **likelihood-free learning** (no need for p_{data} or p_θ), we need to be able to estimate the objective using only samples (e.g., training data and samples from the model). Similar to what we seen before and we can use NNs (Generator, Discriminator).

9.3 Towards Variational Divergence Minimization

- **Fenchel conjugate:** For any function $f(\cdot)$, its convex conjugate is given by:

$$f^*(t) = \sup_{u\in \text{dom}_f}\left(ut - f(u)\right),$$

By definition, the supremum is the maximum value that can be achieved by optimizing over a set of functions. where dom_f is the domain of the function f. Let $T^*(t)$ denote the maximizer of the supremum for a given t, i.e.,

$$T^*(t) = \arg\max_{u\in \text{dom}_f}\left(ut - f(u)\right).$$

This allows us to rewrite:

$$f^*(t) = T^*(t)t - f(T^*(t)).$$

- f^* is convex (the pointwise supremum of convex functions is convex) and lower semi-continuous.
- Let f^{**} be the Fenchel conjugate of f^*:

$$f^{**}(u) = \sup_{t \in \text{dom}_{f^*}} \left(tu - f^*(t)\right),$$

where $T^*(u)$ is the maximizer of the supremum for a given u, i.e.,

$$T^*(u) = \arg\max_{t \in \text{dom}_{f^*}} \left(tu - f^*(t)\right).$$

Thus, we can write:

$$f^{**}(u) = T^*(u)u - f^*(T^*(u)).$$

- $f^{**} \leq f$. **Proof:** By definition, for all t, u,

$$f^*(t) \geq ut - f(u) \quad \text{or equivalently} \quad f(u) \geq ut - f^*(t).$$

Taking the supremum over t, we have:

$$f(u) \geq \sup_t (ut - f^*(t)) = f^{**}(u).$$

- **Strong Duality:** $f^{**} = f$ when $f(\cdot)$ is convex and lower semi-continuous.

The reason this is useful, is that similar to ELBO what we try to write our f in terms of conjugate and this way we will get bounds on the values of f−divergence using above definition. This is the main idea behind **f−GAN**:

$$\begin{aligned}
D_f(p,q) &= \mathbb{E}_{x\sim q}\left[f\left(\frac{p(x)}{q(x)}\right)\right] && \text{(Definition of the } f\text{-divergence)} \\
&= \mathbb{E}_{x\sim q}\left[f^{**}\left(\frac{p(x)}{q(x)}\right)\right] && (f^{**} \text{ is the Legendre transform (convex conjugate) of } f) \\
f^{**} &\stackrel{\text{def}}{=} \sup_{t\in \text{dom}_{f^*}} \left(t\frac{p(x)}{q(x)} - f^*(t)\right) && \text{(Definition of the convex conjugate } f^{**}) \\
&= \mathbb{E}_{x\sim q}\left[\sup_{t\in \text{dom}_{f^*}} \left(t\frac{p(x)}{q(x)} - f^*(t)\right)\right] && \text{(Substitute } f^{**} \text{ into the expectation)} \\
&= \mathbb{E}_{x\sim q}\left[T^*(x)\frac{p(x)}{q(x)} - f^*\left(T^*(x)\right)\right] && \text{(Replace sup with the maximizer } T^*(x) \text{ at each } x)
\end{aligned}$$

For every value of x, there is going to be a different value of the density ratio, there is going to be a different value of t that achieves the supremum. This is similar to what the discriminator is doing.

$$\begin{aligned}
&= \int_{\mathcal{X}} q(x)\left[T^*(x)\frac{p(x)}{q(x)} - f^*\left(T^*(x)\right)\right] dx && \text{(Rewrite expectation as an integral over } \mathcal{X}) \\
&= \int_{\mathcal{X}} \left[T^*(x)p(x) - f^*\left(T^*(x)\right)q(x)\right] dx && \text{(Simplify: } q(x)\cdot\frac{p(x)}{q(x)} = p(x))
\end{aligned}$$

This is basically the difference between expectation over $p(x)$ and $q(x)$, which is similar to **Jensen Shannon**. Additionally, these are expectations with respect to $p(x)$ and $q(x)$ which can be calculated via sampling.

$$= \sup_T \int_{\mathcal{X}} \left[T(x)p(x) - f^*\left(T(x)\right) q(x)\right] dx \qquad \text{(Generalize: } T^*(x) \text{ is the maximizer over all } T\text{)}$$

This indicates that there is going to be some function T that gives us the optimal value of T^* for evey value of x. This is equivalent of the previous step, instead of T^* we use $\sup T$.

using fenchel conjugate

$$f(u) = \sup_T \int_{\mathcal{X}} \left[T(x)p(x) - f^*\left(T(x)\right) q(x)\right] dx \qquad \text{(Generalize: } T^*(x) \text{ is the maximizer over all } T\text{)}$$

$$f(u) \geq \sup_{T \in \mathcal{T}} \int_{\mathcal{X}} \left[T(x)p(x) - f^*\left(T(x)\right) q(x)\right] dx \qquad \text{(Restrict the maximization to a subset } \mathcal{T}\text{: inequality arises)}$$

$$= \sup_{T \in \mathcal{T}} \left(\mathbb{E}_{x\sim p}\left[T(x)\right] - \mathbb{E}_{x\sim q}\left[f^*\left(T(x)\right)\right]\right) \qquad \text{(Separate the terms inside the integral into expectations)}$$

The Fenchel conjugate appears in the term $f^*(T(x))$, which arises from the conjugate representation of f. The lower bound emerges as the supremum over test functions T. Intuitively:

- $T(x)p(x)$: Represents the "linear part" $(t \cdot x)$ of the conjugate in the density-weighted space.
- $q(x)f^*(T(x))$: Represents the cost of the conjugate function, weighted by $q(x)$.

Intuition for the Lower Bound

The Fenchel conjugate $f^*(t)$ provides a way to decompose $f(x)$ into a supremum of linear functions minus the conjugate. The lower bound leverages this decomposition:

- It ensures the supremum over T provides a valid approximation of $D_f(p, q)$.
- $f^*(T(x))$ acts as a "penalty" term, ensuring the test function $T(x)$ respects the convex structure imposed by f.

where $\mathcal{T} : \mathcal{X} \to \mathbb{R}$ is an arbitrary class of functions.

Note: Lower bound is likelihood-free w.r.t. p and q.

9.4 f-GAN: Variational Divergence Minimization

- **Variational lower bound:**

$$D_f(p, q) \geq \sup_{T \in \mathcal{T}} \left(\mathbb{E}_{x\sim p}[T(x)] - \mathbb{E}_{x\sim q}[f^*(T(x))]\right)$$

- Choose any f-divergence.
- Let $p = p_{\text{data}}$ and $q = p_G$.
- Parameterize our NNs $T \equiv \textit{Discriminator}$ by ϕ and G by θ.
- Consider the following f-GAN objective:

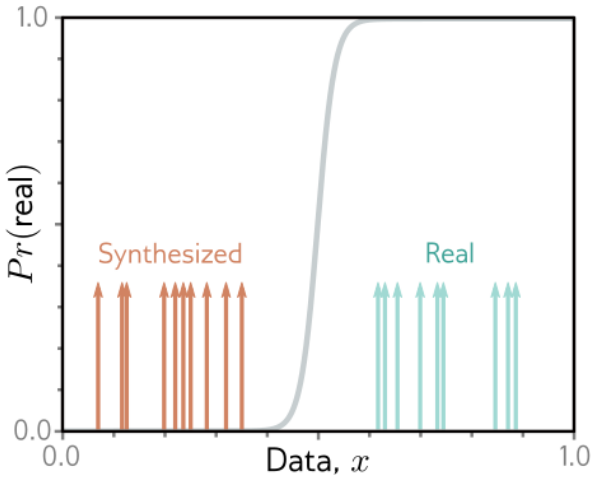
$$\min_\theta \max_\phi F(\theta, \phi) = \mathbb{E}_{x\sim p_{\text{data}}}[T_\phi(x)] - \mathbb{E}_{x\sim p_{G_\theta}}[f^*(T_\phi(x))]$$

- Generator G_θ tries to minimize the divergence estimate, and discriminator T_ϕ tries to tighten the lower bound.

9.5 Vanishing Gradients

When the discriminator is optimal, GAN training minimizes the distance between the generated and real samples. However, if the probability distributions of these samples are disjoint, this distance becomes infinite, and small changes to the generator cannot reduce the loss.

This issue arises because generated samples and real data often reside in distinct low-dimensional subspaces, leading to minimal or no overlap and resulting in vanishing gradients for the generator. As the discriminator becomes more accurate, the generator's gradients decrease, making it difficult to improve the generator further.



In the above image, If the generated samples (orange arrows) are easy to distinguish from the real examples (cyan arrows), then the discriminator (sigmoid) may have a very shallow slope at the positions of the samples; hence, the gradient to update the parameter of the generator may be tiny.

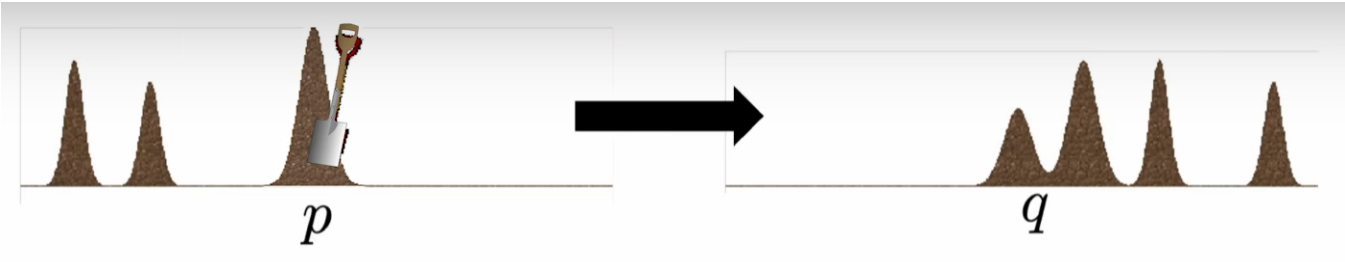
10 WGAN

10.1 Wasserstein (Earth Mover) Distance

We have seen beforet that GAN loss can be interpreted in terms of distances between probability distributions and that the gradient of this distance becomes zero when the generated samples are too easy to distinguish from the real examples. The obvious way forward is to choose a distance metric with better properties.

The Wasserstein or (for discrete distributions) earth mover's distance is the quantity of work required to transport the probability mass from one distribution to create the other. Here, "work" is defined as the mass multiplied by the distance moved. The Wasserstein distance is well-defined even when the distributions are disjoint and decreases smoothly as they become closer to one another.

Analogy - Earth Mover

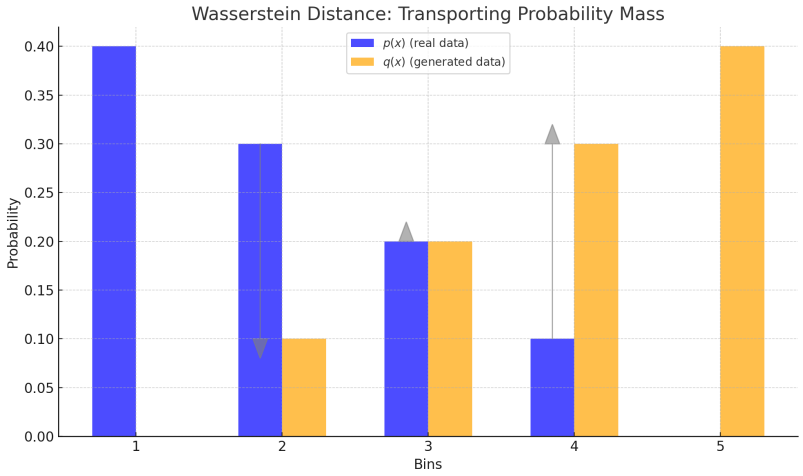


One way of describing the Wasserstein (Earth Mover) Distance is to see how much dirt you need to move to change p to q, the higher the work the larger the distance.

$$D_w(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|_1],$$

where $\Pi(p,q)$ contains all joint distributions of (x,y) where the marginal of x is $p(x) = \int \gamma(x,y)\,dy$, and the marginal of y is $q(y)$.

$\gamma(y \mid x)$: a probabilistic earth moving plan that warps $p(x)$ to $q(y)$.



The gray arrows indicate how the probability mass from $p(x)$ is "moved" to match $q(x)$, with the "work" being proportional to both the mass moved and the distance moved between bins.

Example:

Let

$$p(x) = \begin{cases} 1, & x = 0, \\ 0, & x \neq 0, \end{cases} \quad \text{and} \quad q_\theta(x) = \begin{cases} 1, & x = \theta, \\ 0, & x \neq \theta. \end{cases}$$

- $D_{\mathrm{KL}}(p, q_\theta) = \begin{cases} 0, & \theta = 0, \\ \infty, & \theta \neq 0. \end{cases}$
- $D_{\mathrm{JS}}(p, q_\theta) = \begin{cases} 0, & \theta = 0, \\ \log 2, & \theta \neq 0. \end{cases}$
- $D_w(p, q_\theta) = \left|\theta\right|$

10.2 Discrete Distributions

Consider distributions $\Pr(x = i)$ and $q(x = j)$ defined over K bins. Assume there is a cost C_{ij} associated with moving one unit of mass from bin i in the first distribution to bin j in the second; this cost might be the absolute difference $|i - j|$ between the indices. The amounts that are moved form the *transport plan* and are stored in a matrix $\mathbf{P}$.

The Wasserstein distance is defined as:

$$D_w\left[\Pr(x) \| q(x)\right] = \min_{\mathbf{P}} \left[\sum_{i,j} P_{ij} \cdot |i-j| \right],$$

subject to the constraints that:

$$\sum_j P_{ij} = \Pr(x = i) \quad \text{(initial distribution of } \Pr(x)\text{)},$$

$$\sum_i P_{ij} = q(x = j) \quad \text{(initial distribution of } q(x)\text{)},$$

$$P_{ij} \geq 0 \quad \text{(non-negative masses)}.$$

In other words, the Wasserstein distance is the solution to a constrained minimization problem that maps the mass of one distribution to the other. This is inconvenient as we must solve this minimization problem over the elements P_{ij} every time we want to compute the distance. Fortunately, this is a standard problem that is easily solved for small systems of equations. It is a *linear programming problem* in its primal form:

Primal Form	**Dual Form**
minimize $\mathbf{c}^T\mathbf{p}$, such that $\mathbf{Ap} = \mathbf{b}$, $\mathbf{p} \geq 0$	maximize $\mathbf{b}^T\mathbf{f}$, such that $\mathbf{A}^T\mathbf{f} \leq \mathbf{c}$

where $\mathbf{p}$ contains the vectorized elements P_{ij} that determine the amount of mass moved, $\mathbf{c}$ contains the distances, $\mathbf{Ap} = \mathbf{b}$ contains the initial distribution constraints, and $\mathbf{p} \geq 0$ ensures the masses moved are non-negative.

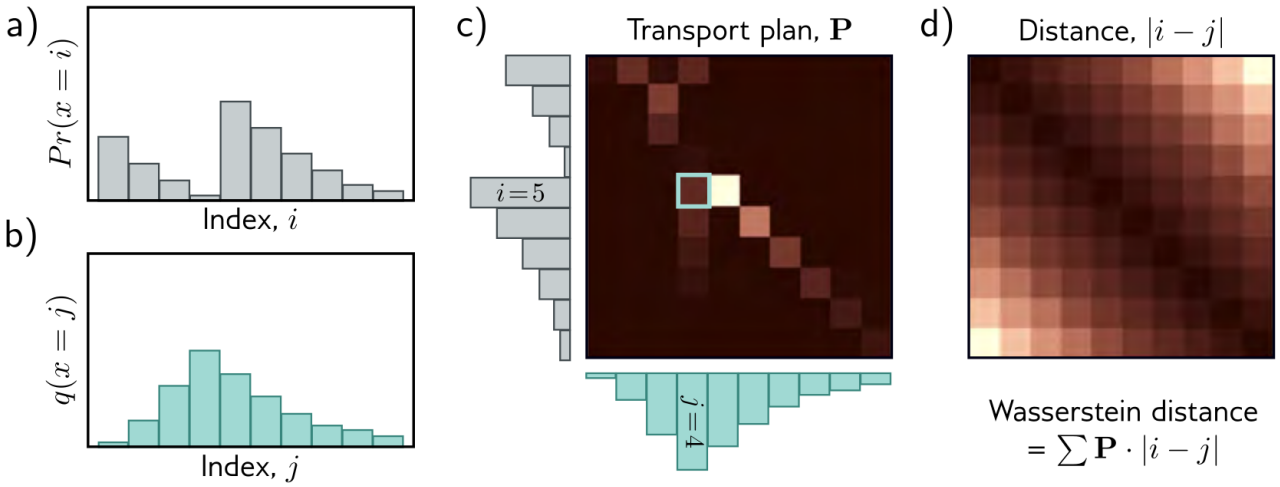
As for all linear programming problems, there is an equivalent *dual problem* with the same solution. Here, we maximize with respect to a variable $\mathbf{f}$ that is applied to the initial distributions, subject to constraints that depend on the distances $\mathbf{c}$. The solution to this dual problem is (Kantorovich-Rubinstein duality):

$$\begin{aligned} D_w\left[\Pr(x) \| q(x)\right] &= \sup_f E_{x\sim p}[f(x)] - E_{x\sim q}[f(x)] \\ &= \max_{\mathbf{f}} \left[\sum_i \Pr(x = i) f_i - \sum_j q(x = j) f_j \right] \end{aligned}$$

subject to: (Lipschitz constant of 1)

$$|f_i - f_j| < 1.$$

10.2.1 Example:



1. **Discrete Distribution:** Consider a discrete distribution $p(x = i)$. This distribution defines the probability mass at each point i in the space.

2. **Target Distribution:** We aim to transform the probability mass of $p(x = i)$ to create a target distribution $q(x = j)$.

3. **Transport Plan:** The transport plan P determines how much mass will be moved from point i to point j. Each entry p_{ij} of the matrix P specifies the amount of mass shifted from i to j. For example, the highlighted entry p_{54} indicates how much mass will be moved from $i = 5$ to $j = 4$. The transport plan P must satisfy the following constraints:

   - $p_{ij} \geq 0 \quad \forall\, i, j$ (non-negativity constraint)
   - $\sum_j p_{ij} = p(x = i) \quad \forall\, i$ (row sums match source distribution)
   - $\sum_i p_{ij} = q(x = j) \quad \forall\, j$ (column sums match target distribution)

   These constraints ensure that P forms a valid joint probability distribution between i and j.

4. **Distance Matrix:** A distance matrix D defines the distance d_{ij} between each pair of points i and j. The distance could be based on Euclidean distance or any other distance metric, depending on the context.

5. **Optimal Transport Plan:** The optimal transport plan P minimizes the total cost of moving mass according to the following objective function:

$$\min_P \sum_{i,j} p_{ij}\, d_{ij}$$

   Here, p_{ij} represents the amount of mass moved from i to j, and d_{ij} represents the distance between these two points. The Wasserstein distance is the minimum value of this cost function. The elements of the optimal transport plan P tend to lie close to the diagonal of the distance matrix D, where the cost d_{ij} is typically smallest.

10.3 Wasserstein distance for continuous distributions

Translating these results back to the continuous multi-dimensional domain, the equivalent of the primal form is:

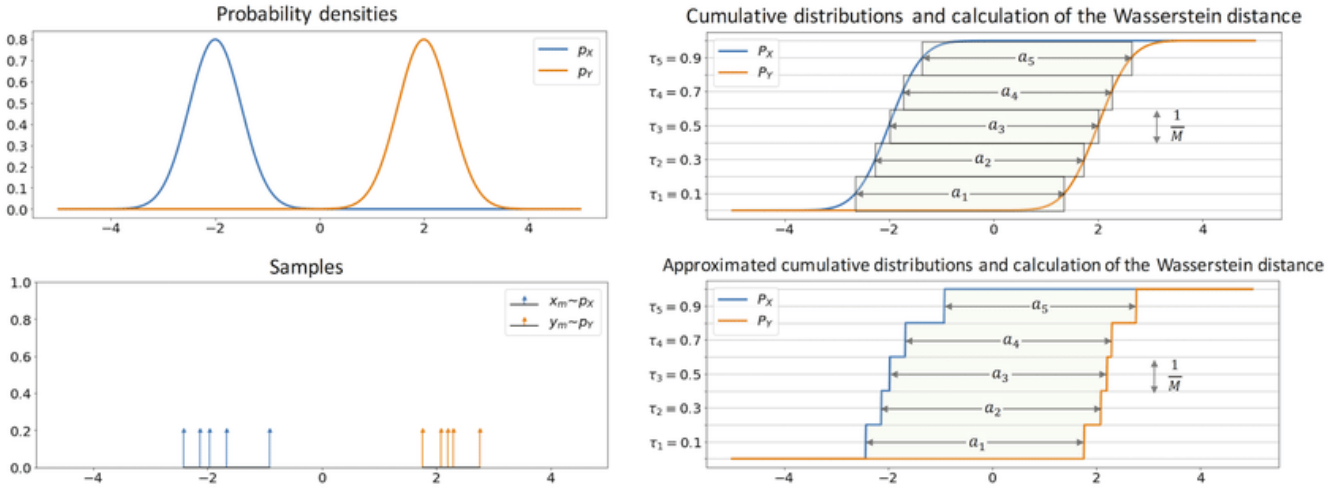
$$D_w\left[Pr(x), q(x)\right] = \min_{\pi(\cdot,\cdot)} \left[\int \int \pi(\mathbf{x_1}, \mathbf{x_2}) \cdot \|\mathbf{x_1} - \mathbf{x_2}\| d\mathbf{x_1} d\mathbf{x_2} \right],$$

subject to constraints similar to the transport plan $\pi(\mathbf{x_1}, \mathbf{x_2})$ representing the mass moved from position $\mathbf{x_1}$ to $\mathbf{x_2}$.

The equivalent of the dual form is:

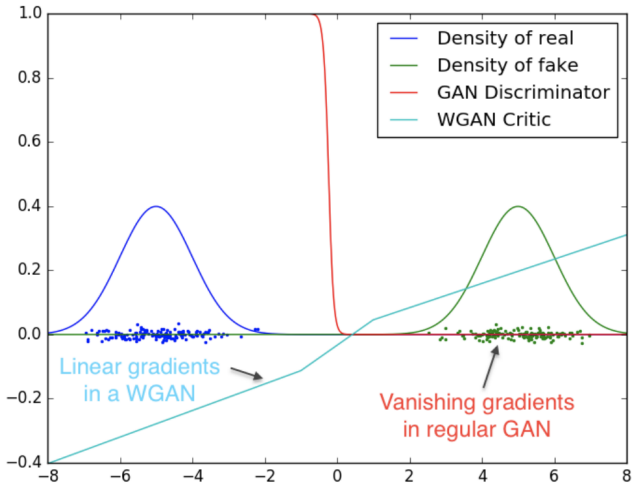
$$D_w\left[Pr(x), q(x)\right] = \max_{f[x]} \left[\int Pr(x) f[x] dx - \int q(x) f[x] dx \right],$$

subject to the constraint that the Lipschitz constant of the function $f[x]$ is less than one (i.e., the absolute gradient of the function is less than one).



- **Top-left:** Displays the probability densities P_X (blue) and P_Y (orange).
- **Top-right:** Shows the cumulative distributions P_X and P_Y, along with dotted lines indicating the quantile levels (τ).
- **Bottom-left:** Illustrates samples drawn from P_X (blue stems) and P_Y (orange stems).
- **Bottom-right:** Approximates cumulative distributions using empirical samples from P_X and P_Y, showing step functions.

10.4 Wasserstein GAN loss function



Wasserstein GAN combines the concepts of a discriminator $D_\phi(x)$ and a generator $G_\theta(z)$, formulated as:

$$\min_\theta \max_\phi \mathbb{E}_{x\sim p_{\text{data}}} \left[D_\phi(x)\right] - \mathbb{E}_{z\sim p(z)} \left[D_\phi(G_\theta(z))\right].$$

In the context of neural networks, the discriminator $D_\phi(x)$ corresponds to a function $f[x, \phi]$, where ϕ represents the parameters of the neural network. The optimization over the discriminator can be expressed as:

$$L[\phi] = \sum_j f[\mathbf{x}_j^*, \phi] - \sum_i f[\mathbf{x}_i, \phi]$$

$$= \sum_j f[g[\mathbf{z}_j, \theta], \phi] - \sum_i f[\mathbf{x}_i, \phi].$$

Here, $\mathbf{x}_j^*$ are generated samples, and $\mathbf{x}_i$ are real examples. To ensure the Lipschitz constraint on $D_\phi(x)$, we require that the absolute gradient norm of the discriminator satisfies:

$$\left| \frac{\partial f[\mathbf{x}, \phi]}{\partial \mathbf{x}} \right| < 1,$$

Enforcing the Lipschitzness is difficult in reality but can be achieved through weight clipping (e.g., ± 0.01) or by applying a gradient penalty on $\nabla_x D_\phi(x)$.

An alternative to weight clipping is the gradient penalty Wasserstein GAN (WGAN-GP), which introduces a regularization term to better enforce the gradient norm constraint. This enhances the stability of training and ensures adherence to the Lipschitz condition.

11 Inferring latent representations in GANs

The generator of a GAN is typically a directed, latent variable model with:

- Latent variables $\mathbf{z}$, and
- Observed variables $\mathbf{x}$.

Key Question: How can we infer the latent feature representations in a GAN?

11.1 Challenges

- **Non-invertibility:** Unlike a normalizing flow model, the mapping $G : \mathbf{z} \mapsto \mathbf{x}$ is not guaranteed to be invertible.
- **No Variational Inference:** Unlike a variational autoencoder, GANs lack an inference network $q(\cdot)$ that can learn a variational posterior over latent variables.

11.2 Solution 1: Using the Discriminator

- For any given point $\mathbf{x}$, use the activations of the prefinal layer of the discriminator as the feature representation.
- **Intuition:** Similar to supervised deep neural networks, the discriminator learns meaningful representations for $\mathbf{x}$ while distinguishing between real and fake $\mathbf{x}$.

11.3 Directly Inferring Latent Variables

If we want to directly infer the latent variables $\mathbf{z}$ of the generator, we require a different learning algorithm.

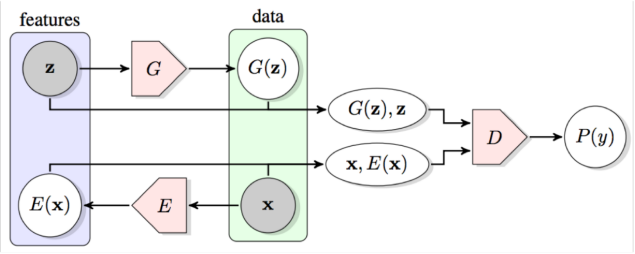
11.4 Solution 2: Comparing Joint Distributions

- A regular GAN optimizes a two-sample test objective that compares:
  - Samples of $\mathbf{z}$ from the generator, and

  - Samples of $\mathbf{x}$ from the data distribution.

- To infer latent representations, we compare samples of $(\mathbf{x}, \mathbf{z})$ from the joint distributions of observed and latent variables as per the model and the data distribution. - **BiGAN**

11.4.1 Bidirectional Generative Adversarial Networks (BiGAN)



Source: Donahue et al., 2016

- In a BiGAN, we have an encoder network E in addition to the generator network G.
- The encoder network only observes $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ during training to learn a mapping $E : \mathbf{x} \mapsto \mathbf{z}$.
- As before, the generator network only observes the samples from the prior $\mathbf{z} \sim p(\mathbf{z})$ during training to learn a mapping $G : \mathbf{z} \mapsto \mathbf{x}$.
- The discriminator D observes samples from:
  - The generative model $\mathbf{z}, G(\mathbf{z})$, and
  - The encoding distribution $E(\mathbf{x}), \mathbf{x}$.
- The goal of the discriminator is to maximize the two-sample test objective between $\mathbf{z}, G(\mathbf{z})$ and $E(\mathbf{x}), \mathbf{x}$.
- After training is complete:
  - New samples are generated via G,
  - Latent representations are inferred via E.

References

[1] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[2] Stefano Ermon. *CS236*.

[3] Martin Arjovsky e.t all. *Wasserstein GAN*.

[4] Soheil Kolouri e.t all. *Sliced-Wasserstein Autoencoder*.