

GENERATIVE MODELS

FROM GAUSSIAN SAMPLING TO DIFFUSION MODELS

**Constant Bourdrez,
PhD Student, ENS-PSL
Université PSL**

IASD
Université Dauphine-PSL

September 24, 2026

WHO AM I?

- ▶ Constant Bourdrez
- ▶ PhD Student, Ecole Normale Supérieure, Université PSL
- ▶ Research group: Centre de Science des Données, ENS-PSL
- ▶ Research interests: generative models, reinforcement learning, differential privacy.
- ▶ All my slides and code are available on www.constantbourdrez.github.io

ABOUT THIS COURSE

Outline of the course:

- ▶ Lecture 1 (24/09, 14h15–17h30): From sampling to the first generative model
- ▶ Practical 1 (25/09, 9h–12h15): Building and training a VAE
- ▶ Lecture 2 (28/09, 9h–12h15): GANs (from the first GAN to f-GAN, WGAN, discriminator rejection sampling)
- ▶ Practical 2 (02/11, 9h–12h15): Training a GAN
- ▶ Lecture 3 (03/11, 9h–12h15): Diffusion Models (DDPM to EDM, score-based and classifier guidance)
- ▶ Practical 3 (05/11, 9h–12h15): Comparing ODE and SDE in small dimensions
- ▶ Lecture 4 + Practical 4 (29/01, 14h15–17h30): Flow Matching Models
- ▶ Lecture 5 (01/02, 9h–12h15): Evaluating generative models (IS, Precision and Recall) + Project presentation
- ▶ Last session (08/03, 9h–12h15): Student Presentations

ABOUT THIS COURSE

Project Presentation:

- ▶ Depending on the number of students, you will have to work in pairs or groups of 3.
- ▶ Same model architecture for all groups.
- ▶ Each group will have to choose a paper for training, regularizing, sampling the model.
- ▶ Each project will be tested every morning for a month.
- ▶ Final presentation on the last session (08/03/2024). Depending on the number of students, each group will have 5 to 7 minutes to present their work.
- ▶ Report to be handed the day before the project at 23:59.
- ▶ Report will be graded on the clarity of the presentation, the quality of the writing, the quality of the intuition behind experiments but not the results.
- ▶ Grade: 40% presentation, 60% report.

INTRODUCTION TO GENERATIVE MODELS

- 1 What is sampling? 10**
 - 1.1 Definition 10
 - 1.2 Random and Pseudo-random sampling 22
 - 1.3 From uniform to Gaussian / Gaussian mixtures 26
- 2 Generative Models 42**
 - 2.1 Definition 42
 - 2.2 Implicit vs Explicit Models 42
 - 2.3 Divergences 47
 - 2.4 Examples of Generative Models 56
- 3 Types of Generative Models 58**
 - 3.1 Autoregressive Models 58
 - 3.2 Normalizing Flows 63
 - 3.3 Energy-Based Models 73
- 4 From an Autoencoder to a Generative Model 77**
 - 4.1 Autoencoder 77
 - 4.2 Variational Autoencoder 79
 - 4.3 Variants of VAE 84

GENERATIVE ADVERSARIAL NETWORKS

- 1 From the first GAN to f -GAN and WGAN 87
 - 1.1 A generator and a discriminator 87
 - 1.2 In practice 93
 - 1.3 Generalizing GANs with f -divergences 96

- 2 Evolution of Neural Architectures for GANs 114
 - 2.1 Architectural Milestones 114
 - 2.2 Applications 125
 - 2.3 Improving GANs using the Discriminator 135

DIFFUSION MODELS

- 1 Introduction to Diffusion Models 139**
 - 1.1 Denoising Diffusion Probabilistic Models (DDPM) 139
 - 1.2 DDPM: Sampling and Results 147

- 2 SDE and ODE Formulations 153**
 - 2.1 Score-Based Models and SDEs 153
 - 2.2 SDE Formulation 155
 - 2.3 Probability-Flow ODE 162

- 3 From Research to Industrial Applications 165**
 - 3.1 Guidance Techniques 165
 - 3.2 Increasing the dimensionality Efficiency 173

FLOW MATCHING MODELS

- 1 Flow Matching Models 178**
 - 1.1 Optimal Transport and Flow Matching: Reminders 178
 - 1.2 From Diffusion to Flow Matching 184
 - 1.3 Conditional Probability Paths 186
 - 1.4 The Flow Matching Objective 188
 - 1.5 Sampling and the Choice of Coupling 190
 - 1.6 Wrap-up 192

EVALUATION OF GENERATIVE MODELS & PROJECT PRESENTATIONS

1	Evaluation of Generative Models	194
1.1	What are we trying to measure?	194
1.2	Image generation metrics	199
1.3	Precision and Recall for Generative Models	203
1.4	Conditional generation and alignment	214
1.5	Text generation metrics	217
2	Project Presentations	221
2.1	Trading-off Quality and Diversity in GANs	221
2.2	Final Project: Presentation, Code and Report	225
3	Using computation resources for your project	230
3.1	What is SLURM ?	230
3.2	How to use it ?	233
3.3	How to connect to Mesonet ?	235

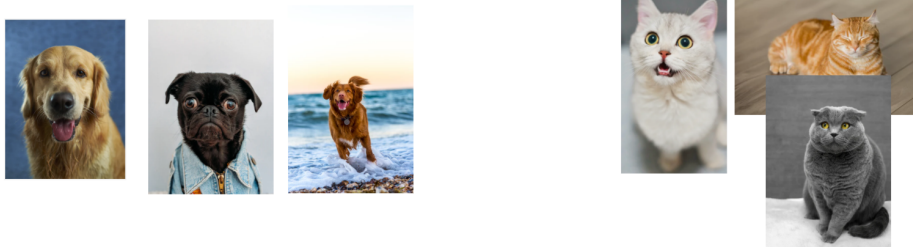
Part I

INTRODUCTION TO GENERATIVE MODELS

WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :

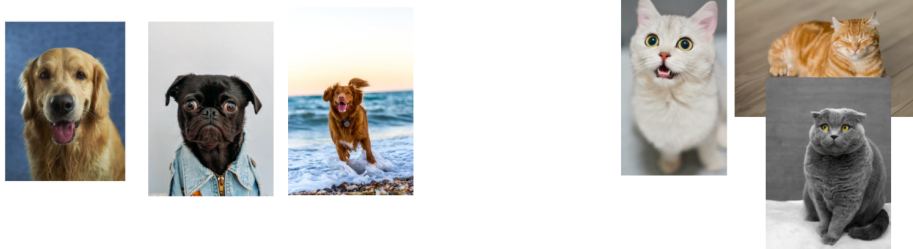


Data

WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

What constraints should a generated image satisfy?

WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

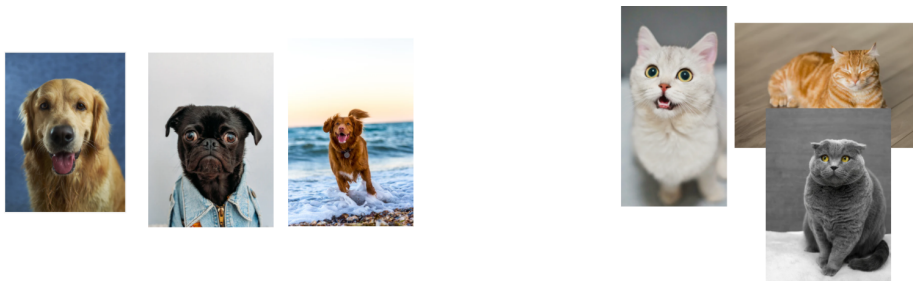


What constraints should a generated image satisfy?

WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

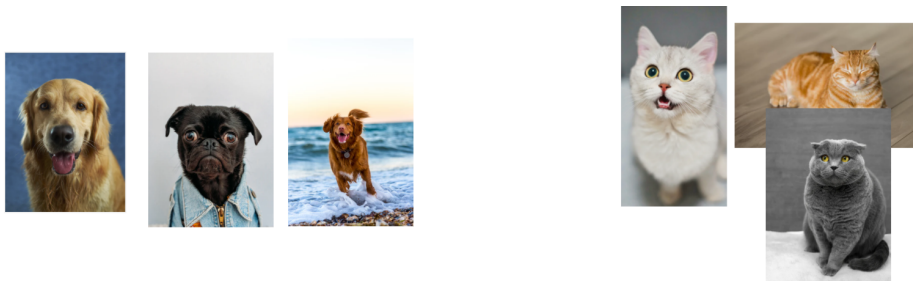


How do we ensure resemblance to real images?

WHAT IS SAMPLING?

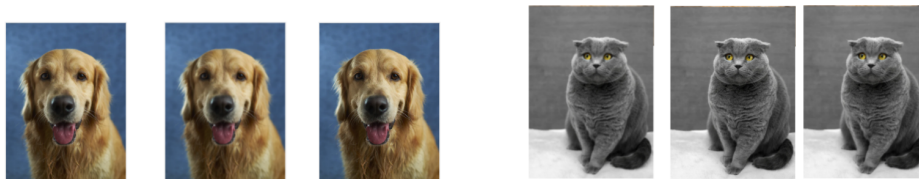
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model

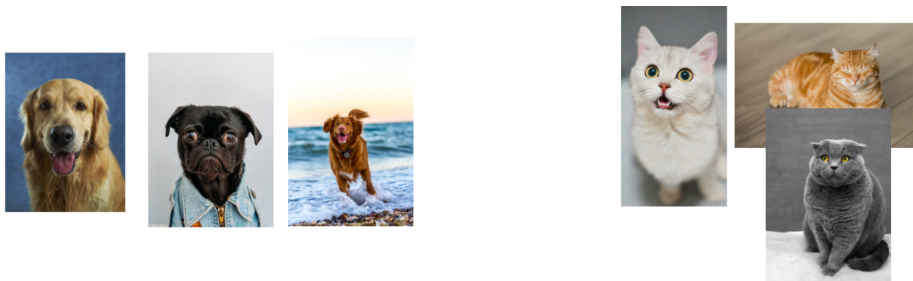


How do we ensure resemblance to real images?

WHAT IS SAMPLING?

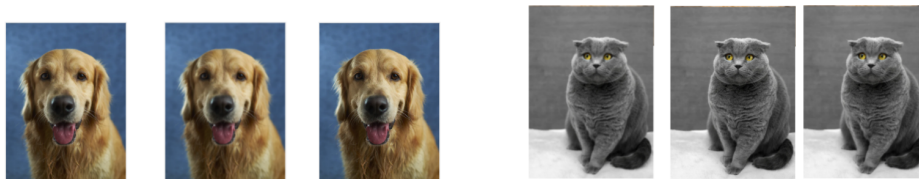
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model

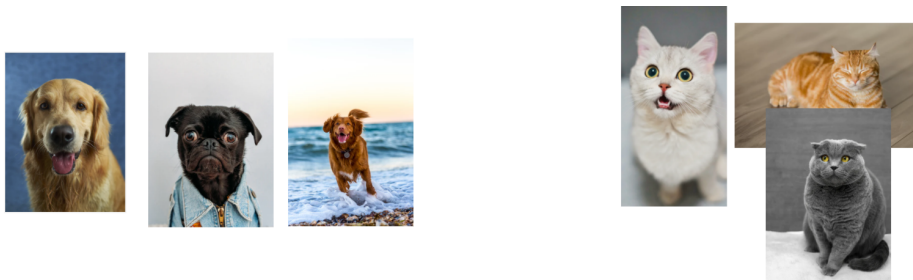


How to cover the diversity of the dataset?

WHAT IS SAMPLING?

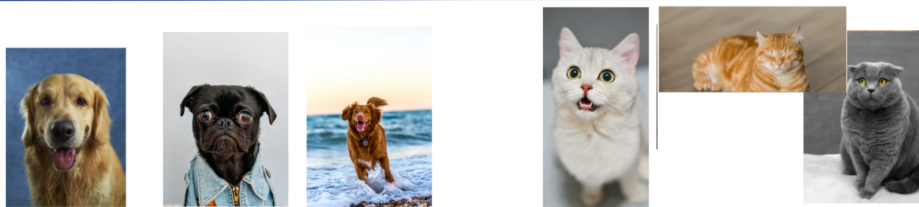
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model

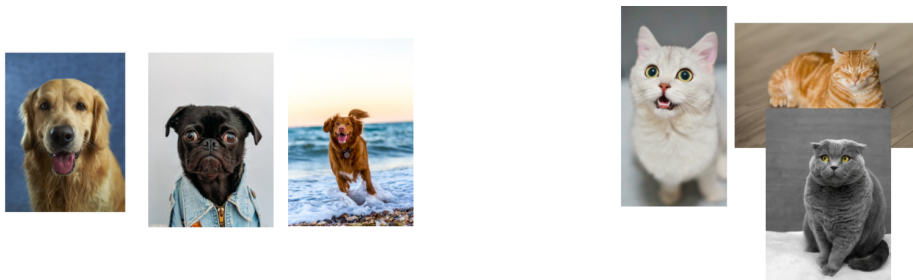


How to cover the diversity of the dataset?

WHAT IS SAMPLING?

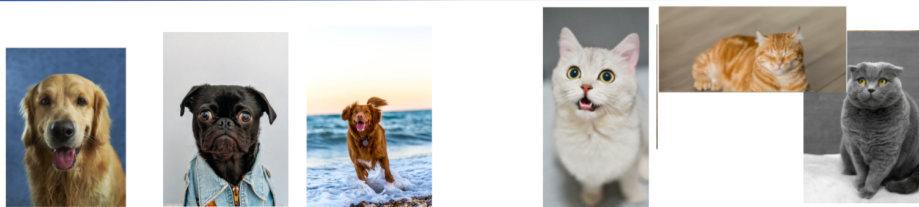
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model

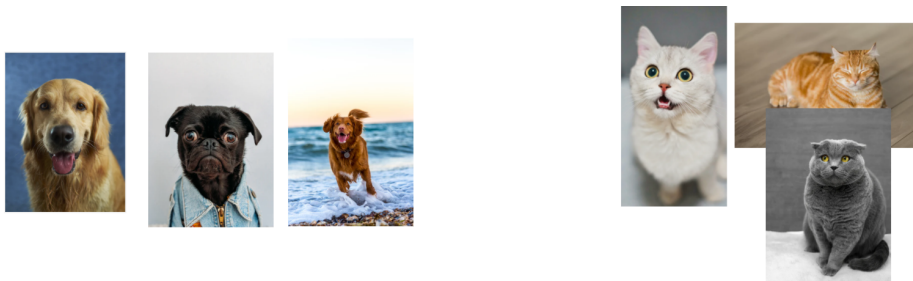


What about likelihood and novelty?

WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model

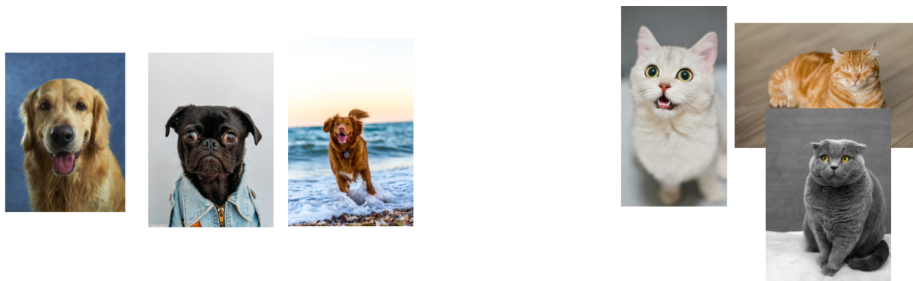


What about likelihood and novelty?

WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model

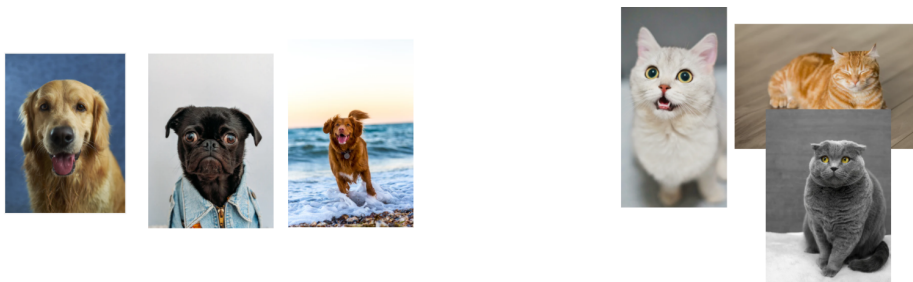


How to incorporate stochasticity in generation?

WHAT IS SAMPLING?

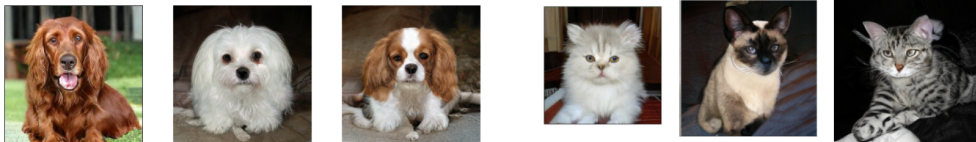
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P :



Data

Model



WHAT IS SAMPLING?

OBJECTIVE OF SAMPLING IMAGES

Constraints for sampling:

- ▶ *Resemblance*: Generated images should look like real ones.
- ▶ *Coverage*: Samples should represent the full data distribution.
- ▶ *Likelihood*: Samples should have high probability under the model.
- ▶ *Novelty*: Samples should not simply replicate training data.
- ▶ *Stochasticity*: Sampling should reflect inherent randomness.

WHAT IS SAMPLING?

WHY SAMPLING MATTERS

Definition:

Let P be a distribution on the sample space $\mathcal{X}$. The goal is to sample points under the distribution $p(x)$. Often, we consider conditional distributions $p(x|y)$ where y is some conditioning variable.

- ▶ Image generation : e.g., sampling new human faces (FFHQ, CelebA).
- ▶ Text generation : e.g., language models generating sentences.
- ▶ Image-to-image translation : e.g., translating day-to-night photos.
- ▶ Image-to-text generation : e.g., automatic image captioning.
- ▶ Text-to-image synthesis : e.g., generating images from text prompts.
- ▶ Text-to-text generation : e.g., machine translation.
- ▶ Speech-to-text transcription : e.g., transcribing audio to subtitles.
- ▶ Many other conditional generative tasks...

WHAT IS SAMPLING?

EXAMPLES: IMAGES-TO-IMAGES COLORIZATION

Example: Image Colorization

Original Image



Grayscale Image



Colorized Image



Task: Given a grayscale image, sample a plausible colored version.

WHAT IS SAMPLING?

EXAMPLES: IMAGES-TO-IMAGES INPAINTING

Example: Image Inpainting

Original Image



Masked Image



Filled Image



Task: Given an image with missing regions, sample a plausible completion.

WHAT IS SAMPLING?

EXAMPLES: IMAGES-TO-IMAGES UNCROPPING

Example: Image Uncropping

Original Image



Cropped Image



Filled Image



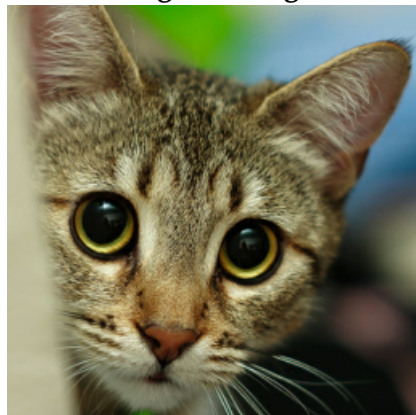
Task: Given an image with missing regions, sample a plausible completion.

WHAT IS SAMPLING?

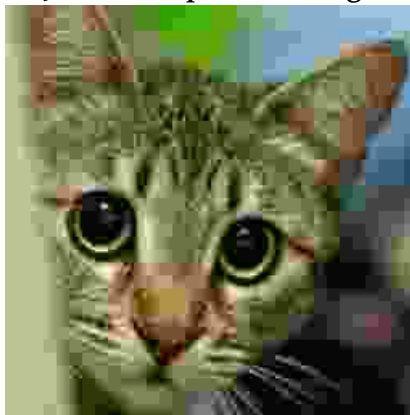
EXAMPLES: IMAGES-TO-IMAGES JPEG ARTIFACT REMOVAL

Example: Image JPEG Artifact Removal

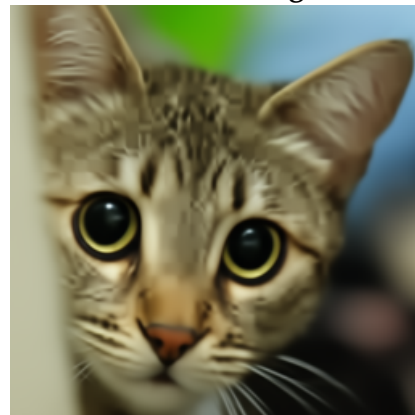
Original Image



JPEG Compressed Image



Restored Image



Task: Given a JPEG compressed image, sample a high-quality restoration.

WHAT IS SAMPLING?

EXAMPLES: TEXT-TO-IMAGE SYNTHESIS

Example: Text-to-Image Synthesis

Parti-350M



Parti-750M



Parti-3B



Parti-20B



A portrait photo of a kangaroo wearing an orange hoodie and blue sunglasses standing on the grass in front of the Sydney Opera House holding a sign on the chest that says Welcome Friends!

Task: Given a text description, sample a corresponding image.

WHAT IS SAMPLING?

EXAMPLES: TEXT-TO-TEXT TRANSLATION

Example: Text-to-Text Translation

Input (English):

A scenic view of a mountain during sunset.

A bustling city street at night.

A serene beach with palm trees.

Output (French):

Une vue pittoresque d'une montagne au coucher du soleil.

Une rue animée de la ville la nuit.

Une plage sereine avec des palmiers.

Task: Given an English text description, sample a French translation.

WHAT IS SAMPLING?

EXAMPLES: TEXT-TO-TEXT MATH PROBLEM SOLVING

Example: Text-to-Text Math Problem Solving

Input (Problem):

If a car travels at 60 mph for 2 hours, how far does it go?

What is the derivative of $x^2 + 3x + 2$?

Solve for x : $2x + 5 = 15$.

Output (Solution):

The car travels 120 miles.

The derivative is $2x + 3$.

The solution is $x = 5$.

Task: Given a text description, sample a corresponding solutions.

WHAT IS SAMPLING?

HOW DO WE ACTUALLY GENERATE AN IMAGE?

Prompt:

"Emmanuel Macron wearing a puffy jacket and beanie, looking intimidating."

?



How should we generate something random corresponding to this?

WHAT IS SAMPLING?

HOW DO WE ACTUALLY GENERATE AN IMAGE?

Prompt:

"Emmanuel Macron wearing a puffy jacket and beanie, looking intimidating."

?

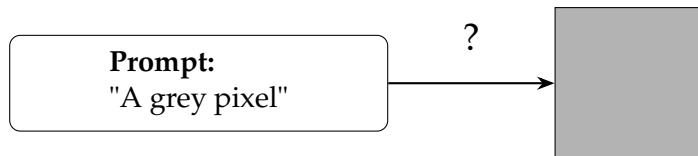


How should we generate something random corresponding to this?

It's very complex.

FROM COMPLEX TO SIMPLE

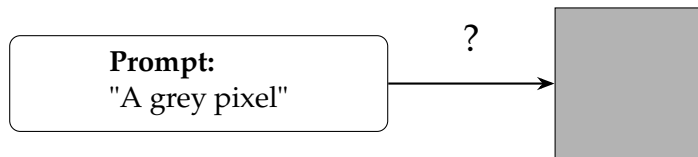
SIMPLIFYING THE PROBLEM



The simplest generation we can do: how can we do this?

FROM COMPLEX TO SIMPLE

SIMPLIFYING THE PROBLEM



The simplest generation we can do: how can we do this?

Even this is not trivial!

RANDOM VS PSEUDO-RANDOM SAMPLING

DEFINITIONS AND DIFFERENCES

Random (true randomness). Numbers generated from nondeterministic physical processes.

- ▶ Examples: quantum effects (photon arrival, electron tunneling), radioactive decay, thermal/shot noise.
- ▶ Properties: unpredictable, not reproducible from a finite state; entropy comes from physics.
- ▶ Uses: cryptography key generation, lotteries, high-stakes simulations.

Pseudo-random (PRNG). Numbers generated by deterministic algorithms.

- ▶ Defined by a recurrence/state update; *reproducible* given a seed.
- ▶ Aim for long period, good statistical tests, fast generation; not inherently cryptographically secure.
- ▶ Uses: ML training, Monte Carlo, graphics, games; seeding controls reproducibility.

RANDOM AND PSEUDO-RANDOM SAMPLING

SOURCES AND ALGORITHMS

True random sources (hardware/physical):

- ▶ Quantum RNGs (beam-splitter photon paths, vacuum fluctuations), radioactive decay counters.
- ▶ Electronic noise (thermal/Johnson–Nyquist, avalanche diode shot noise).
- ▶ External entropy (network jitter, disk timings) — lower quality, needs whitening.

Popular PRNG algorithms (software):

- ▶ Linear Congruential Generators (LCG), Xorshift / xoroshiro, PCG (permuted congruential generator).
- ▶ Mersenne Twister (MT19937): long period, equidistribution; standard in many libraries.
- ▶ Cryptographic PRNGs (ChaCha20-CTR, AES-CTR-DRBG) for security-sensitive use.

PSEUDO-RANDOM GENERATION

LINEAR CONGRUENTIAL GENERATOR (LCG)

Statement. We can generate *uniform* pseudo-random numbers efficiently with a simple PRNG (e.g., LCG) using a seed for reproducibility.

Algorithm 1: LCG (one-step)

Input: m, a, c, X_0

Output: $U_t \in (0, 1)$

$X_{t+1} \leftarrow (aX_t + c) \bmod m;$

$U_{t+1} \leftarrow X_{t+1}/m$

FROM UNIFORM TO MORE COMPLEX?

A NATURAL QUESTION

- ▶ OK, we can sample **uniform** variables quickly.
- ▶ **What about more complex distributions?** e.g., a Gaussian or a Gaussian mixture.
- ▶ Idea: *transform* uniforms into the target distribution. (Next: Box–Muller, Inverse CDF)

$$U \sim \text{Unif}(0, 1) \xRightarrow{\text{Box-Muller}} Z \sim \mathcal{N}(0, 1)$$

SAMPLING TRANSFORMATIONS

BOX-MULLER (UNIFORM $\rightarrow$ GAUSSIAN)

Idea: map two i.i.d. uniforms (U_1, U_2) to two i.i.d. Gaussians (Z_1, Z_2).

- Polar transform: $R = \sqrt{-2 \ln U_1}$, $\Theta = 2\pi U_2$; then $Z_1 = R \cos \Theta$, $Z_2 = R \sin \Theta$.
- Produces *pairs* of normals; efficient reuse in vectorized code.
- Alternative: Marsaglia polar method avoids costly trig with rejection.

Algorithm 2: Box-Muller

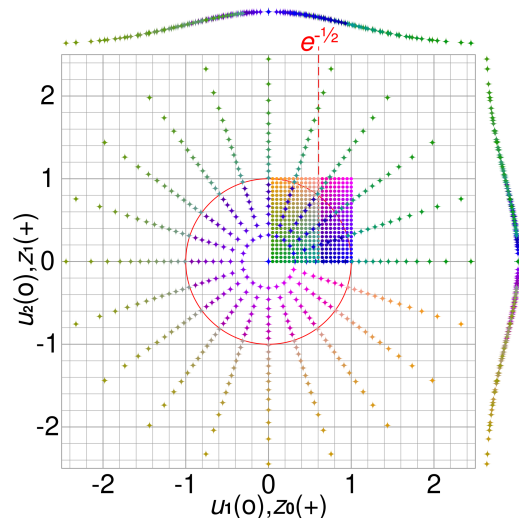
Output: $Z_1, Z_2 \sim \mathcal{N}(0, 1)$ i.i.d.

Draw $U_1, U_2 \sim \text{Unif}(0, 1)$ i.i.d.

$R \leftarrow \sqrt{-2 \log U_1}$

$\Theta \leftarrow 2\pi U_2$

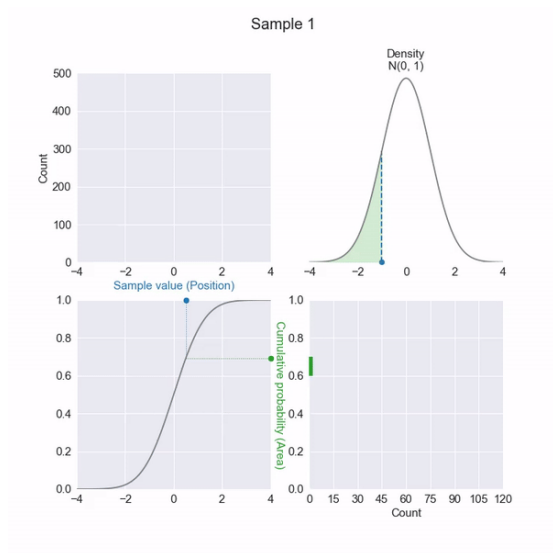
$Z_1 \leftarrow R \cos \Theta$; $Z_2 \leftarrow R \sin \Theta$



INVERSE CDF

A.K.A. INVERSE TRANSFORM SAMPLING

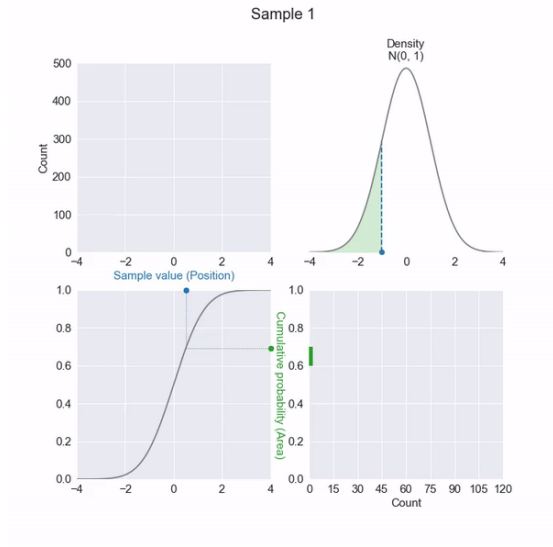
- ▶ If $U \sim \text{Unif}(0, 1)$ and F is a CDF, then $X = F^{-1}(U)$ has CDF F .
- ▶ Requires F to be strictly increasing / invertible; otherwise use generalized inverse.
- ▶ In practice: precompute/discretize F , interpolate F^{-1} , or use spline/numerical root-finding.



INVERSE CDF

A.K.A. INVERSE TRANSFORM SAMPLING

- ▶ If $U \sim \text{Unif}(0, 1)$ and F is a CDF, then $X = F^{-1}(U)$ has CDF F .
- ▶ Requires F to be strictly increasing / invertible; otherwise use generalized inverse.
- ▶ In practice: precompute/discretize F , interpolate F^{-1} , or use spline/numerical root-finding.



Question: What is the issue with this method in high dimensions?

HIGH DIMENSIONAL SAMPLING

WHEN IS IT (RELATIVELY) EASY?

Fact: Sampling in high dimensions is generally hard, even if we know $p(x_1, \dots, x_N)$.

Special cases where it becomes easy(er):

- ▶ **Independence:** $p(x_1, \dots, x_N) = \prod_{i=1}^N p_i(x_i)$ — sample each coordinate independently.
- ▶ **Markov (autoregressive) chain:** $p(x_{1:N}) = p(x_1) \prod_{i=2}^N p(x_i | x_{i-1})$ — ancestral sampling along the chain.
- ▶ **Tree-structured Bayesian networks:** order variables topologically and sample parents $\rightarrow$ children.

FROM UNIVARIATE TO MULTIVARIATE GAUSSIAN

DENSITIES AND PARAMETERS

Univariate (1D) Gaussian. For mean $\mu \in \mathbb{R}$ and standard deviation $\sigma > 0$,

$$X \sim \mathcal{N}(\mu, \sigma^2), \quad p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

Parameters:

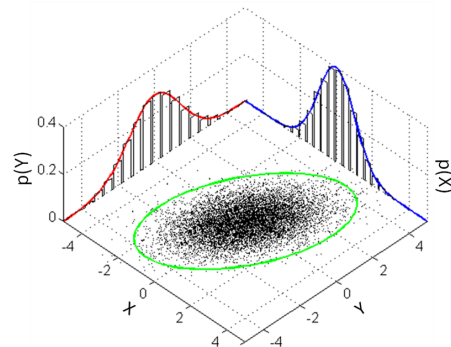
- ▶ μ (location): shifts the center of mass of the density.
- ▶ σ (scale): controls spread (variance σ^2).

Multivariate (d D) Gaussian. For mean $\mu \in \mathbb{R}^d$ and covariance $\Sigma \in \mathbb{R}^{d \times d}$ (symmetric PD),

$$X \sim \mathcal{N}(\mu, \Sigma), \quad p(x) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right).$$

Parameters:

- ▶ μ (location vector): shifts the center in $\mathbb{R}^d$.
- ▶ Σ (covariance): encodes scale/shape/orientation; $\Sigma \succ 0$.



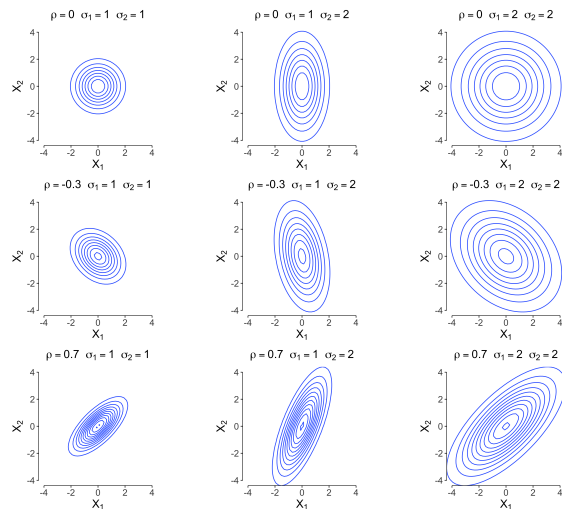
Elliptical contours illustrating Σ in $\mathbb{R}^d$.

FROM UNIVARIATE TO MULTIVARIATE GAUSSIAN

COVARIANCE TYPES

Special cases.

- Isotropic: $\Sigma = \sigma^2 \mathbf{I} \Rightarrow$ spherical contours.
- Diagonal: $\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_d^2) \Rightarrow$ independent coordinates.



Elliptical contours illustrating Σ in $\mathbb{R}^d$.

FROM $\mathcal{N}(0, \mathbf{I})$ TO $\mathcal{N}(\mu, \Sigma)$

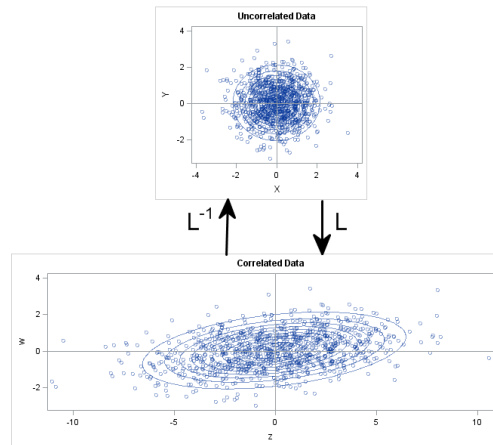
CHOLSKY FACTORIZATION AND SAMPLING

Cholesky factorization. For a symmetric positive-definite Σ , there exists a unique lower-triangular L with positive diagonal such that $\Sigma = LL^\top$.

Sampling recipe.

1. Draw $z \sim \mathcal{N}(0, \mathbf{I}_d)$ (independent standard normals).
2. Compute L such that $LL^\top = \Sigma$ (Cholesky).
3. Set $x \leftarrow \mu + Lz \Rightarrow$ then $x \sim \mathcal{N}(\mu, \Sigma)$.

Why it works. $\mathbb{E}[z] = 0$, $\text{Cov}(z) = \mathbf{I}$ and $\text{Cov}(Lz) = LIL^\top = \Sigma$.



Map unit sphere (standard normal) to ellipse via L , then translate by μ .

GAUSSIAN MIXTURES

DEFINITION AND INTUITION

Motivation. To increase the model complexity beyond a single Gaussian, we can *mix* several Gaussians; this yields a flexible, multi-modal density.

Definition. A Gaussian Mixture Model (GMM) with K components on $\mathbb{R}^d$ is

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k), \quad \pi_k \geq 0, \quad \sum_k \pi_k = 1.$$

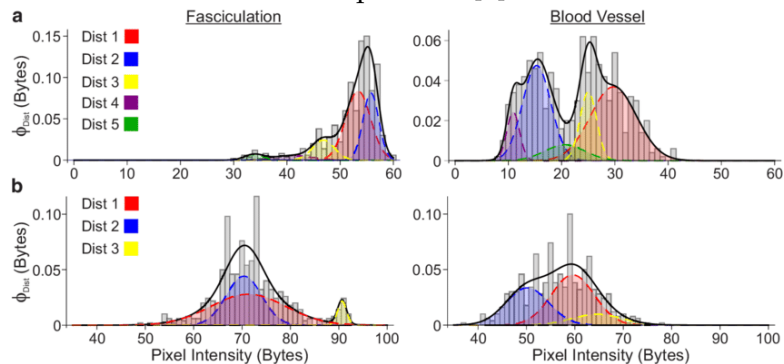
Parameters: weights $\{\pi_k\}$, means $\{\mu_k\}$, covariances $\{\Sigma_k\}$.

GAUSSIAN MIXTURES

WHY GAUSSIAN MIXTURES?

- *Expressivity*: universal approximator of smooth densities as K increases.

Example from [3]:



GAUSSIAN MIXTURES

SAMPLING FROM MIXTURE MODELS

Goal: draw $X \sim p(x) = \sum_k \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$.

Algorithm 3: Sampling from a GMM

Input: weights $\{\pi_k\}$, means $\{\mu_k\}$, covariances $\{\Sigma_k\}$

Output: $X \in \mathbb{R}^d$

Draw component index $K \sim \text{Categorical}(\pi_1, \dots, \pi_K)$

Draw $X \sim \mathcal{N}(\mu_K, \Sigma_K)$ (e.g., via Box–Muller/Cholesky)

Notes:

- ▶ Use prefix-sum table for the categorical draw; vectorize for batches.
- ▶ For Σ_K : diagonal for speed; Cholesky factor L with $LL^\top = \Sigma_K$ for full-cov.

TRAINING GAUSSIAN MIXTURES

MAXIMUM LIKELIHOOD — PRINCIPLE

Hypothesis: we observe i.i.d. data $x_1, \dots, x_N \sim P$ and posit a parametric model $p_\theta(x)$ (here, a GMM).

► **Population objective:**

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{X \sim P} [\log p_\theta(X)]$$

► **Empirical log-likelihood (practice):**

$$\hat{\theta} = \arg \max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_\theta(x_n) = \arg \max_{\theta} \log \prod_{n=1}^N p_\theta(x_n).$$

► **Why the log?** Turns products into sums; numerically stable.

Notes: LLN links empirical and population objectives; for GMMs, regularize Σ_k with a small $\epsilon \mathbf{I}$ to avoid degeneracy.

TRAINING GAUSSIAN MIXTURES

MAXIMUM LIKELIHOOD ESTIMATION (MLE)

Objective (given data $\{x_n\}_{n=1}^N$):

$$\mathcal{L}(\Theta) = \sum_{n=1}^N \log \left(\sum_{k=1}^K \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \right) = \sum_{n=1}^N \underbrace{\text{LSE}_k \left(\log \pi_k + \log \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \right)}_{\text{log-sum-exp}}.$$

where $\text{LSE}(z_1, \dots, z_K) := \log \sum_{k=1}^K e^{z_k}$.

- ▶ Constraints: $\pi \in \Delta^{K-1}$ (simplex), $\Sigma_k \succ 0$ (PD).
- ▶ Numerics: compute with the *log-sum-exp trick* (subtract $\max_k$) for stability.
- ▶ EM connection: LSE is a smooth max; E-step computes responsibilities as a softmax:

$$\gamma_{nk} = \text{softmax}_k \left(\log \pi_k + \log \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \right).$$

TRAINING GAUSSIAN MIXTURES

GRADIENT DESCENT APPROACH

Parameterization for constraints:

- ▶ Weights: raw logits w_k with $\pi_k = \text{softmax}(w)_k$.
- ▶ Covariance: diagonal with $\sigma_{k,j}^2 = \exp(\alpha_{k,j})$; or full-cov via Cholesky $\Sigma_k = L_k L_k^\top$.

Stabilization:

- ▶ Compute $\log \sum_k$ via *log-sum-exp*; clip α ; add ϵI to Σ_k .
- ▶ Mini-batch SGD/Adam; early stopping; multiple restarts.

Pseudocode (one step):

Algorithm 4: GD step for GMM MLE

Input: minibatch $\{x_b\}$, current Θ

Compute $\ell_b = \log \sum_k \pi_k \mathcal{N}(x_b \mid \mu_k, \Sigma_k)$ for all b (log-sum-exp)

$J \leftarrow -\frac{1}{|B|} \sum_b \ell_b$

// negative log-likelihood

Backprop to get $\nabla_{\Theta} J$; update $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} J$ (Adam)

TRAINING GAUSSIAN MIXTURES

EM ALGORITHM — INTUITION

From LSE to EM. Using the LSE view, the softmax acts like a *soft argmax*: in the E-step we softly pick the near-maximum component; in the M-step we optimize parameters using those soft weights (points near the max matter more).

$$\gamma_{nk} = \text{softmax}_k \left(\log \pi_k + \log \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \right).$$

Latent variables: introduce $z_{nk} \in \{0, 1\}$ (one-hot) with $\mathbb{P}(z_{nk} = 1) = \pi_k$.

- ▶ **E-step:** responsibilities $\gamma_{nk} = \mathbb{P}(z_{nk} = 1 \mid x_n, \Theta^{(t)})$ (soft assignments).
- ▶ **M-step:** maximize the expected complete-data log-likelihood

$$Q(\Theta, \Theta^{(t)}) = \sum_{n,k} \gamma_{nk} \left[\log \pi_k + \log \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \right].$$

- ▶ **Guarantees:** each EM iteration non-decreasing in data log-likelihood; converges to a stationary point.

When to prefer EM vs GD: closed-form M-steps (fast, stable) vs flexible constraints/priors with GD.

TRAINING GAUSSIAN MIXTURES

EXPECTATION-MAXIMIZATION (EM) ALGORITHM (PSEUDO-CODE)

Algorithm 5: EM for Gaussian Mixture Models

Input: data $\{x_n\}_{n=1}^N$, number of components K

Output: parameters $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$

Initialize π_k, μ_k, Σ_k (e.g., k-means)

repeat

 // E-step: responsibilities

for $n = 1$ **to** N **do**

for $k = 1$ **to** K **do**

$r_{nk} \leftarrow \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$

end

$\gamma_{nk} \leftarrow r_{nk} / \sum_{j=1}^K r_{nj}$

end

 // M-step: parameter updates

$N_k \leftarrow \sum_{n=1}^N \gamma_{nk}$ **for** $k = 1, \dots, K$

$\pi_k \leftarrow N_k / N$

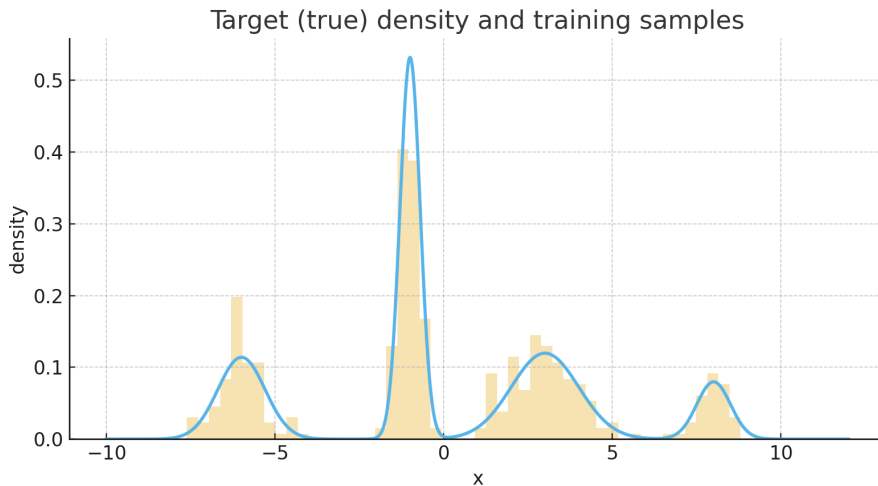
$\mu_k \leftarrow \frac{1}{N_k} \sum_{n=1}^N \gamma_{nk} x_n$

$\Sigma_k \leftarrow \frac{1}{N_k} \sum_{n=1}^N \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^\top + \epsilon \mathbf{I}$

until convergence

GAUSSIAN MIXTURES

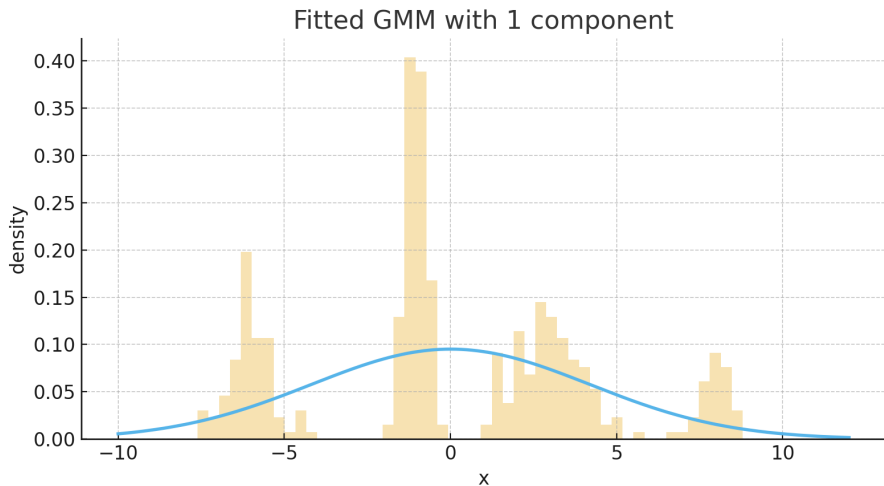
EXAMPLE



Fitting a GMM to 1D data points. In blue the density, in orange the histogram of data points.

GAUSSIAN MIXTURES

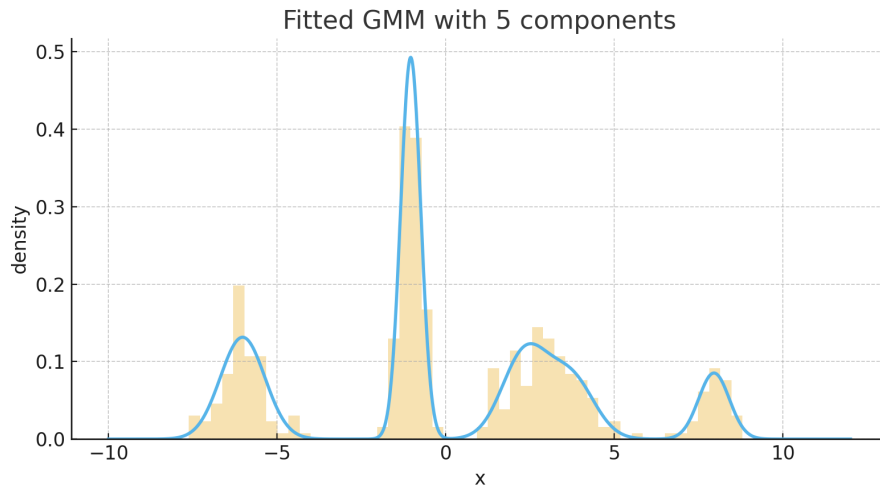
EXAMPLE



Fitting a GMM to 1D data points. $K = 1$.

GAUSSIAN MIXTURES

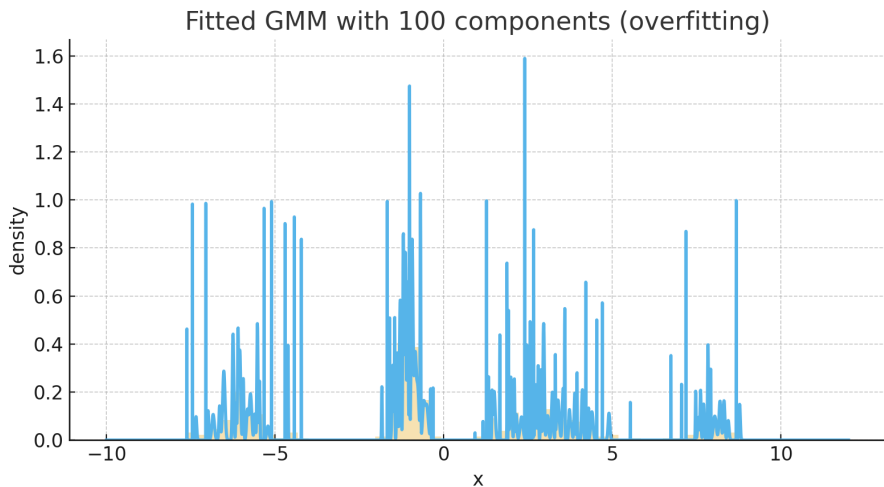
EXAMPLE



Fitting a GMM to 1D data points. $K = 5$.

GAUSSIAN MIXTURES

EXAMPLE



Fitting a GMM to 1D data points. $K = 150$.

GAUSSIAN MIXTURES

IN TERMS OF GENERATIVE MODELS

- ▶ *Resemblance*: If the modes are on the data, samples will look realistic.
- ▶ *Coverage*: More components $\rightarrow$ better coverage of data distribution.
- ▶ *Likelihood*: GMMs provide explicit likelihoods for samples.
- ▶ *Novelty*: If not overfitting, samples can be novel.
- ▶ *Stochasticity*: Inherent randomness from component and Gaussian sampling.

GAUSSIAN MIXTURES

IN TERMS OF GENERATIVE MODELS

- ▶ *Resemblance*: If the modes are on the data, samples will look realistic.
- ▶ *Coverage*: More components → better coverage of data distribution.
- ▶ *Likelihood*: GMMs provide explicit likelihoods for samples.
- ▶ *Novelty*: If not overfitting, samples can be novel.
- ▶ *Stochasticity*: Inherent randomness from component and Gaussian sampling.

The GMM is a simple generative model! However, it has limitations in high dimensions and complex data (e.g., images).

GENERATIVE MODELS

TARGET DISTRIBUTION P AND MODEL $\hat{P}_\theta$

Goal. Approximate the (unknown) data distribution P on $\mathcal{X}$ with a *model family* $\{\hat{P}_\theta : \theta \in \Theta\}$.

Definition. A *generative model* is a probability distribution $\hat{P}_\theta$ over $\mathcal{X}$, parameterized by θ , together with a sampling procedure $x \sim \hat{P}_\theta$.

Types of generative models.

- ▶ **Explicit models:** provide a tractable density $\hat{p}_\theta(x)$ (e.g., GMMs, autoregressive, flows).
- ▶ **Implicit models:** define only a sampler $x = T_\theta(\varepsilon)$ with $\varepsilon \sim p(\varepsilon)$ (e.g., GANs, simulators).

IMPLICIT VS EXPLICIT MODELS

DEFINITIONS

Explicit (likelihood-based).

- ▶ Provide tractable density $\hat{p}_\theta(x)$ (or exact likelihood via change of variables).
- ▶ Examples: autoregressive (exact), flows (exact via Jacobian), GMMs (explicit), diffusion (via surrogate bounds).

What do explicit likelihood-based models allow us to do?

- ▶ *Out-of-distribution (OOD) detection*: Compute likelihoods for new samples; flag samples with low likelihood as OOD or anomalous.
- ▶ *Uncertainty quantification*: Assign probabilities to possible outcomes, enabling principled ways to measure uncertainty.
- ▶ *Anomaly detection*: Identify rare or unexpected events by their low likelihood under the model.
- ▶ *Model comparison*: Compare different generative models quantitatively using likelihoods or information criteria.
- ▶ *Principled training*: Enable maximum likelihood estimation and evaluation on held-out data.

IMPLICIT VS EXPLICIT MODELS

CHANGE OF VARIABLES FORMULA

Change of variables formula for densities:

Suppose $F : \mathcal{X} \rightarrow \mathcal{Z}$ is an *invertible* and *differentiable* function between open subsets of $\mathbb{R}^d$. Let $q(z)$ be a density on $\mathcal{Z}$. Then the induced density on $\mathcal{X}$ is:

$$p(x) = q(F(x)) \cdot |\det \text{Jac}_F(x)|$$

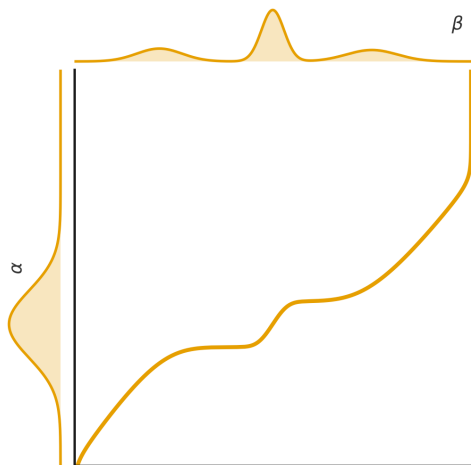
where $\text{Jac}_F(x)$ is the Jacobian matrix of F at x .

Conditions:

- ▶ F must be invertible (bijection) and differentiable, with differentiable inverse.
- ▶ The determinant of the Jacobian must be nonzero everywhere in the domain.

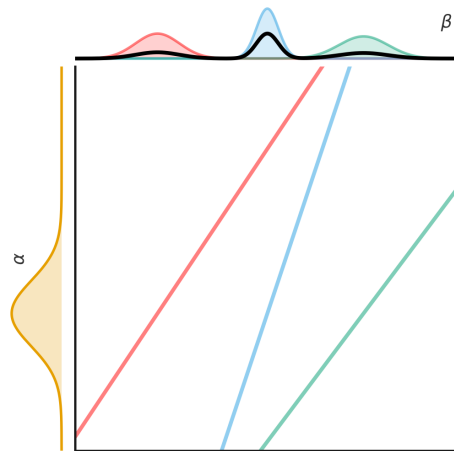
IMPLICIT VS EXPLICIT MODELS

CHANGE OF VARIABLES FORMULA



Deterministic Monge transport ($T = F\beta^{-1} \circ F\alpha$)

Illustration of the change-of-variables formula: mapping a simple distribution (e.g., gaussian) through a deterministic function F to obtain a more complex distribution.



Stochastic transport: three linear maps (left map in red)

Illustration of the change-of-variables formula: mapping a simple distribution (e.g., gaussian) through a stochastic function to obtain a more complex distribution.

IMPLICIT VS EXPLICIT MODELS

PUSH-FORWARD AND LIMITATIONS OF CHANGE OF VARIABLES

Invertibility and Differentiability: Not always possible

Most functions F from $\mathcal{X}$ to $\mathcal{Z}$ are *not* invertible and differentiable everywhere, especially when mapping between spaces of different dimensions or when F is not one-to-one.

- ▶ The change-of-variables formula applies only when F is a bijection between open subsets of $\mathbb{R}^d$ and both F and F^{-1} are differentiable.
- ▶ In practice, many interesting mappings (e.g., neural networks with bottlenecks, dimensionality reduction) do not satisfy these conditions.

Push-forward distribution (formal definition)

Given a measurable function $F : \mathcal{Z} \rightarrow \mathcal{X}$ and a probability distribution Q on $\mathcal{Z}$, the **push-forward** distribution $P = F_{\#}Q$ on $\mathcal{X}$ is defined by:

$$P(B) = Q(F^{-1}(B)) \quad \text{for any measurable set } B \subseteq \mathcal{X}.$$

That is, P is the distribution of $x = F(z)$ when $z \sim Q$.

Summary: The push-forward framework generalizes change-of-variables to cases where F may not be invertible or differentiable, but in those cases, we cannot write a simple density formula for $p(x)$.

DIVERGENCES

DEFINITION AND ESTIMATION TARGET

Divergence $\mathcal{D}(P\|Q)$. A non-negative functional with $\mathcal{D}(P\|Q) \geq 0$ and $\mathcal{D}(P\|Q) = 0$ iff $P = Q$. Not necessarily symmetric; no triangle inequality (*not* a distance).

Learning objective.

$$\theta^* = \arg \min_{\theta \in \Theta} \mathcal{D}(P \parallel \hat{P}_\theta).$$

Examples next: KL, Total Variation, Wasserstein.

Goal: Find θ by minimizing a divergence $\mathcal{D}(P \parallel \hat{P}_\theta)$ or a surrogate (e.g., ELBO).

DIVERGENCES

KULLBACK–LEIBLER (FORWARD KL)

For $P \ll Q$ with densities p, q ,

$$\mathcal{D}_{\text{KL}}(P\|Q) = \int p(x) \log \frac{p(x)}{q(x)} dx = \mathbb{E}_{X \sim P} [\log p(X) - \log q(X)].$$

- ▶ Asymmetric; mode-covering when used as $\mathcal{D}_{\text{KL}}(P\|Q)$ in many settings.
- ▶ **MLE link:** $\mathbb{E}_P[\log q_\theta(X)] = -\mathcal{D}_{\text{KL}}(P\|Q_\theta) - H(P)$.
- ▶ Requires $q > 0$ wherever $p > 0$ (absolute continuity).

DIVERGENCES

TOTAL VARIATION (TV)

$$\text{TV}(P, Q) = \sup_{A \subseteq \mathcal{X}} |P(A) - Q(A)| = \frac{1}{2} \int |p(x) - q(x)| dx.$$

- ▶ Metric on probability measures; bounded in $[0, 1]$.
- ▶ Interpretable as maximum test error gap over events.
- ▶ Hard to estimate directly in high dimension.

DIVERGENCES

WASSERSTEIN (OPTIMAL TRANSPORT)

For cost $c(x, y) = \|x - y\|$ and couplings $\Pi(P, Q)$,

$$\mathcal{W}_1(P, Q) = \inf_{\pi \in \Pi(P, Q)} \mathbb{E}_{(X, Y) \sim \pi} [\|X - Y\|].$$

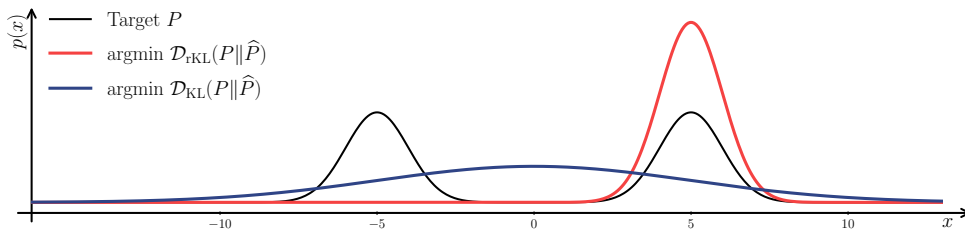
- ▶ Sensitive to the geometry of $\mathcal{X}$; finite even with disjoint supports.
- ▶ Dual (Kantorovich–Rubinstein): $\mathcal{W}_1 = \sup_{\|f\|_{\text{Lip}} \leq 1} (\mathbb{E}_P[f] - \mathbb{E}_Q[f])$.
- ▶ Basis for WGAN objectives.

DIVERGENCES

KL VS REVERSE KL

Forward KL $\mathcal{D}_{\text{KL}}(P\|Q)$ tends to be *mode-covering*; **Reverse KL** $\mathcal{D}_{\text{KL}}(Q\|P)$ tends to be *mode-seeking*.

- ▶ Forward KL penalizes missing mass where $p > 0$ and $q \approx 0$ (heavy penalty).
- ▶ Reverse KL penalizes placing mass where $q > 0$ but $p \approx 0$; may ignore small modes.
- ▶ Choice impacts behavior of trained models.



HOW TO COMPUTE A DIVERGENCE?

KL AND MAXIMUM LIKELIHOOD

From KL to log-likelihood (derivation).

$$\begin{aligned}\mathcal{D}_{\text{KL}}(P\|Q_\theta) &= \int p(x) \log \frac{p(x)}{q_\theta(x)} dx \\ &= \mathbb{E}_{X \sim P} [\log p(X) - \log q_\theta(X)] \\ &= \underbrace{\mathbb{E}_P[\log p(X)]}_{-H(P)} - \mathbb{E}_P[\log q_\theta(X)] \\ &= -\mathbb{E}_P[\log q_\theta(X)] - H(P).\end{aligned}$$

Therefore,

$$\mathbb{E}_P[\log q_\theta(X)] = -\mathcal{D}_{\text{KL}}(P\|Q_\theta) - H(P),$$

and since $H(P)$ does not depend on θ ,

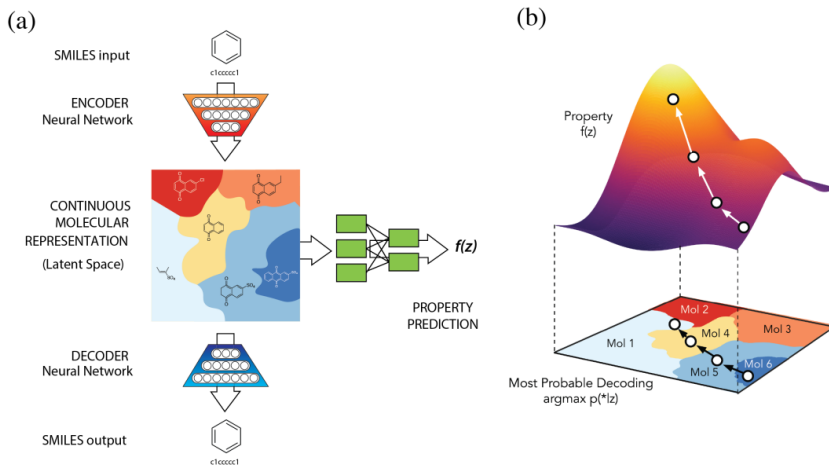
$$\theta^* = \arg \max_{\theta} \mathbb{E}_P[\log q_\theta(X)] \equiv \arg \min_{\theta} \mathcal{D}_{\text{KL}}(P\|Q_\theta).$$

HOW TO COMPUTE A DIVERGENCE?

LATENT VARIABLES AND LOW-DIMENSIONAL STRUCTURE

Idea. Introduce a *latent space* $\mathcal{Z} = \mathbb{R}^k$ with $k \ll d$ (data in $\mathbb{R}^d$) to capture low-dimensional structure.

- ▶ **Generative story:** sample $z \sim q(z)$ (prior), then $x \sim p_\theta(x | z)$ (decoder/model).
- ▶ **Inference:** approximate the posterior with $q_\phi(z | x)$ (encoder) since $p_\theta(z | x)$ is intractable.
- ▶ Benefits: compression, structure, disentanglement, controllable generation.



HOW TO COMPUTE A DIVERGENCE?

DERIVING THE ELBO

Goal: maximize $\log p_\theta(x)$ where $p_\theta(x) = \int p_\theta(x, z) dz$.
For any distribution $q_\phi(z | x)$,

$$\begin{aligned}\log p_\theta(x) &= \log \int q_\phi(z | x) \frac{p_\theta(x, z)}{q_\phi(z | x)} dz \\ &\geq \int q_\phi(z | x) \log \frac{p_\theta(x, z)}{q_\phi(z | x)} dz \\ &= \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(x, z) - \log q_\phi(z | x) \right] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z) + \log p(z) - \log q_\phi(z | x)] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z)] - \mathcal{D}_{\text{KL}}(q_\phi(z | x) \| p(z))\end{aligned}$$

Define the **ELBO**:

$$\text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z)] - \mathcal{D}_{\text{KL}}(q_\phi(z | x) \| p(z)).$$

HOW TO COMPUTE A DIVERGENCE?

APPROXIMATE MLE AND ELBO

ELBO in KL form.

$$\text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x | z)] - \mathcal{D}_{\text{KL}}(q_\phi(z | x) \| p(z)).$$

And the exact evidence decomposes as

$$\log p_\theta(x) \geq \text{ELBO}(\theta, \phi; x) - \mathcal{D}_{\text{KL}}(q_\phi(z | x) \| p_\theta(z | x)),$$

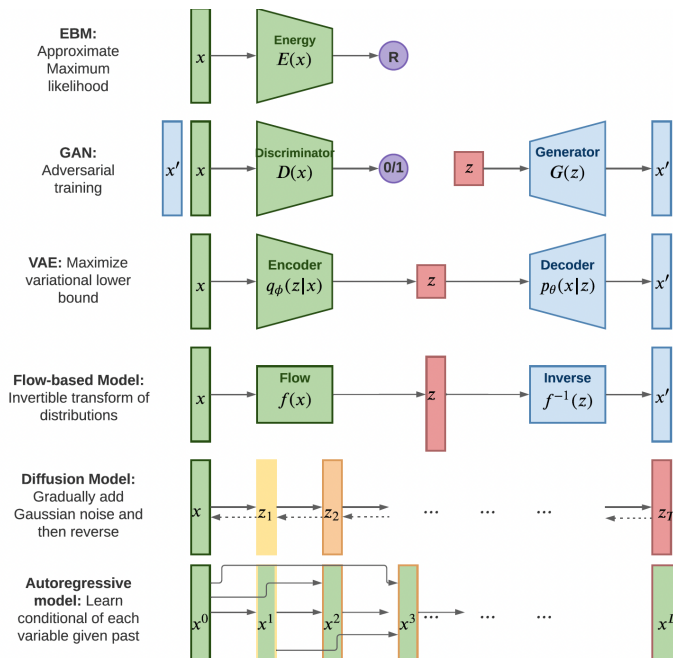
so maximizing ELBO *minimizes* the posterior KL.

What to optimize:

- ▶ **w.r.t. θ (decoder/model):** increase the reconstruction term $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x | z)]$; use Monte Carlo gradients with reparameterized z .
- ▶ **w.r.t. ϕ (encoder/inference):** tighten the bound by driving $q_\phi(z | x)$ toward $p_\theta(z | x)$ (reducing the posterior KL above), while also respecting the regularizer $\mathcal{D}_{\text{KL}}(q_\phi(z | x) \| p(z))$.
- ▶ **Training recipe:** joint SGD on (θ, ϕ) with mini-batches and the reparameterization trick $z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon, \varepsilon \sim \mathcal{N}(0, \mathbf{I})$.

FAMILIES OF GENERATIVE MODELS

VISUAL TAXONOMY



MODEL LANDSCAPE

AT A GLANCE

Model	Density	Sampling	Training	Latents	Architecture	Discussed
ARM	Exact, fast	Slow	MLE	None	Sequential	Here
Flows	Exact, slow / fast	Slow	MLE	$\mathbb{R}^d$	Invertible	Here
EBM	Approx, slow	Slow	MLE-A	Optional	Discriminative	Here
VAE	LB, fast	Fast	MLE-LB		Encoder–Decoder	Here and TP1
GAN	Jensen Approx	Fast	Min–max	$\mathbb{R}^m$	Generator–Discriminator	Session 2 and TP2
Diffusion	LB	Slow	MLE-LB	$\mathbb{R}^d$	Encoder–Decoder	Session 3 and TP3

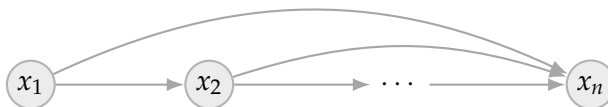
AUTOREGRESSIVE MODELS

LIKELIHOOD DECOMPOSITION

Chain rule factorization. For $x = (x_1, \dots, x_d)$ and any fixed ordering,

$$p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i \mid x_{<i}) \iff \log p_{\theta}(x) = \sum_{i=1}^d \log p_{\theta}(x_i \mid x_{<i}).$$

- ▶ The ordering (sequence order, raster scan for images, etc.) defines the conditional structure.
- ▶ Each factor is a *simple* conditional model (e.g., categorical over tokens/pixels, Gaussian for reals).
- ▶ Tractable likelihood: evaluation and gradients are exact.



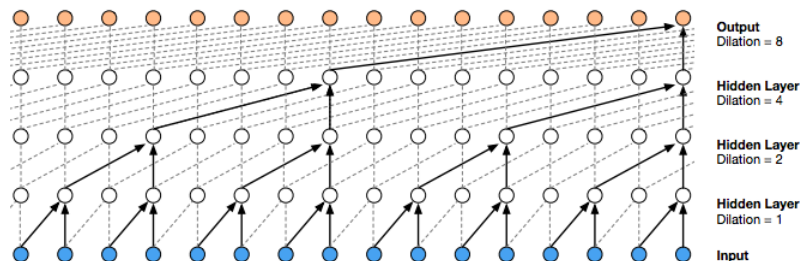
AUTOREGRESSIVE MODELS

TRAINING OBJECTIVE

Maximum likelihood = sum of conditional cross-entropies.

$$\max_{\theta} \mathbb{E}_{x \sim P} [\log p_{\theta}(x)] = \max_{\theta} \mathbb{E}_{x \sim P} \left[\sum_{i=1}^d \log p_{\theta}(x_i | x_{<i}) \right].$$

- ▶ Empirical objective: $-\frac{1}{N} \sum_n \sum_i \log p_{\theta}(x_{n,i} | x_{n,<i})$.
- ▶ *Teacher forcing*: condition on true prefixes $x_{<i}$ during training.
- ▶ Implementation: causal masking (sequences), masked convolutions (images), parallel loss over all positions.



AUTOREGRESSIVE MODELS

GENERATION (ANCESTRAL SAMPLING)

Ancestral sampling: draw variables one-by-one following the factorization order.

Algorithm 6: Autoregressive ancestral sampling

Input: learned conditionals $p_{\theta}(x_i \mid x_{<i})$, dimension d

Output: sample $x = (x_1, \dots, x_d)$

for $i \leftarrow 1$ **to** d **do**

 | Sample $x_i \sim p_{\theta}(\cdot \mid x_{<i})$

end

Notes:

- ▶ Exact and simple; sequential cost $\mathcal{O}(d)$ (limited parallelism at inference).
- ▶ For discrete outputs (text, pixels): categorical sampling; for continuous: Gaussian or mixture.
- ▶ Temperature/top- k /nucleus (p) sampling often used for text (heuristics, not MLE).

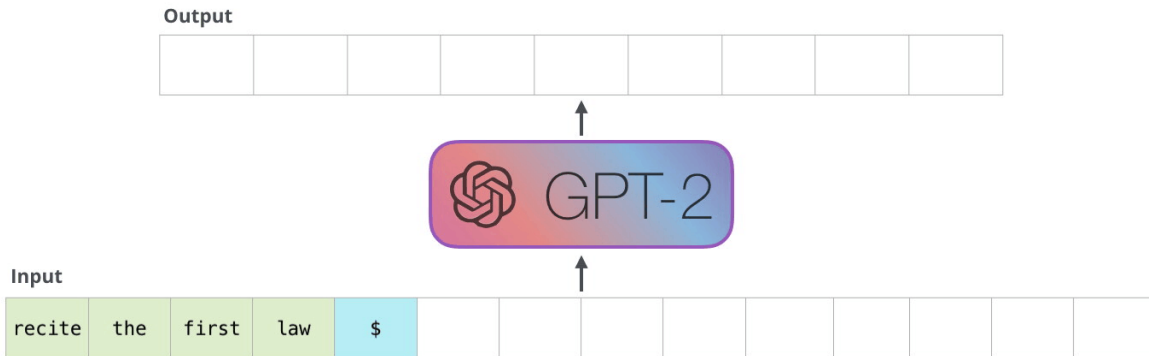
AUTOREGRESSIVE MODELS

LLMs AS AUTOREGRESSIVE NEXT-TOKEN PREDICTORS

Token sequence factorization. For tokens $w_{1:T}$,

$$p_{\theta}(w_{1:T}) = \prod_{t=1}^T p_{\theta}(w_t \mid w_{<t}), \quad \log p_{\theta}(w_{1:T}) = \sum_{t=1}^T \log p_{\theta}(w_t \mid w_{<t}).$$

- ▶ Transformer *decoder*-only with causal mask models $p_{\theta}(w_t \mid w_{<t})$.
- ▶ Training: minimize cross-entropy to true next token (teacher forcing, parallel over positions).
- ▶ Tokenization: subword units (BPE/WordPiece) turn text into discrete tokens; softmax over vocab.



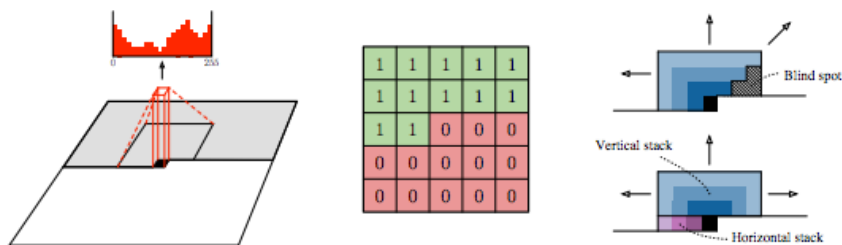
AUTOREGRESSIVE MODELS

PIXELCNN / PIXELRNN

Image factorization. Raster-scan ordering over pixels (and optionally channels):

$$p_{\theta}(x) = \prod_{i=1}^{HW \cdot C} p_{\theta}(x_i | x_{<i}).$$

- ▶ **Masked convolutions** enforce causality (no access to future pixels).
- ▶ **Receptive field** grows with layers; PixelRNN uses recurrent structure.
- ▶ Discrete pixels: categorical over 256 bins or mixture of logistics (PixelCNN++).



NORMALIZING FLOWS

OVERVIEW

A Normalizing Flow is usually seen as:

- ▶ a *generative model*,
- ▶ a *bijective mapping*,
- ▶ an *invertible neural network*,
- ▶ a *density estimator*.

NORMALIZING FLOWS

MAPPING BETWEEN DISTRIBUTIONS — POINT TO POINT

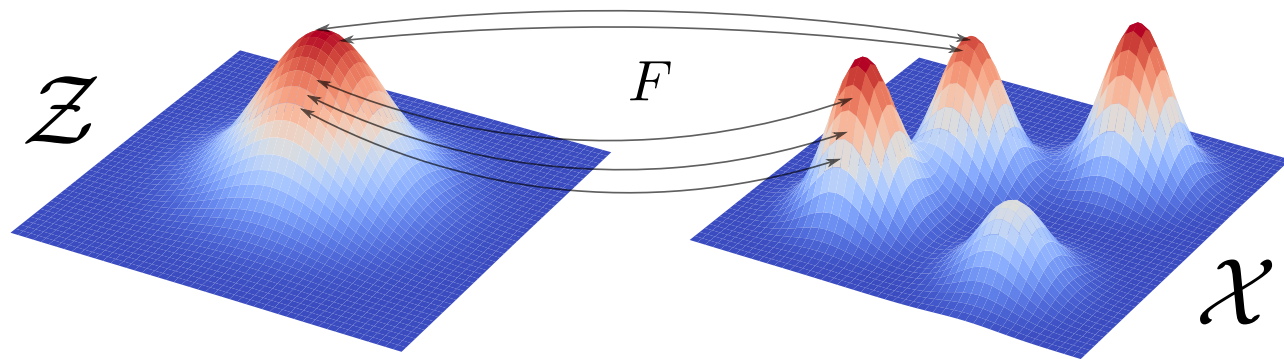


Figure. A mapping between two probability distributions
Point to point

NORMALIZING FLOWS

MAPPING BETWEEN DISTRIBUTIONS — SUBSET TO SUBSET

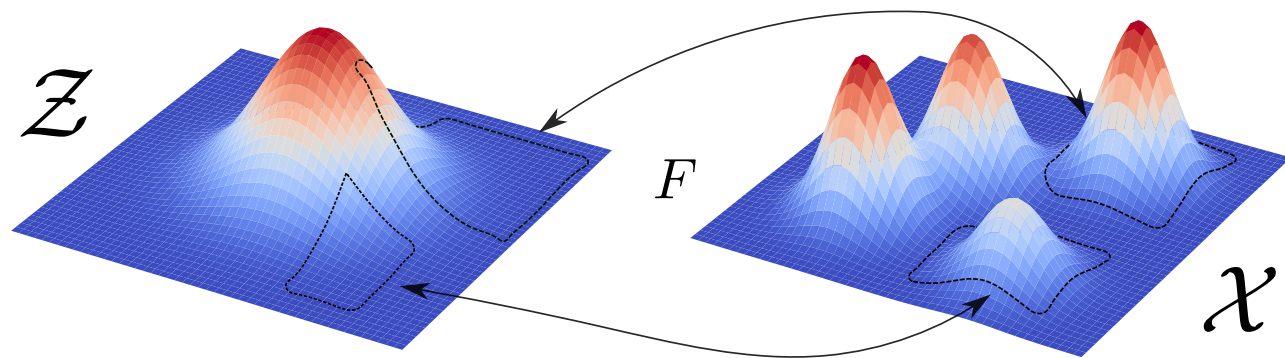


Figure. A mapping between two probability distributions
Subset to subset

NORMALIZING FLOWS

MATHEMATICAL FRAMEWORK

Normalizing Flow

A Normalizing Flow is a bijective function between a data space $\mathcal{X}$ and a latent space $\mathcal{Z}$, both subsets of $\mathbb{R}^d$.

$$\begin{aligned} F : \mathcal{X} &\longmapsto \mathcal{Z} \\ x &\longmapsto z = F(x) \end{aligned}$$

Data and Latent Distributions

In theory, a NF maps a target distribution P (the data distribution) to a simple latent distribution Q . Usually, Q is set to be a multivariate normal $\mathcal{N}(0, \mathbf{I}_d)$. p and q denote the densities of P and Q respectively.

NORMALIZING FLOWS

HOW DOES IT WORK?

In practice, the mapping is *not perfect*. P induces a distribution Q through F , and the latent distribution Q induces $\hat{P}$ through F^{-1} , which is the learned distribution. The forward pass F is called the *normalizing direction* while the inverse pass F^{-1} is called the *generative direction*.

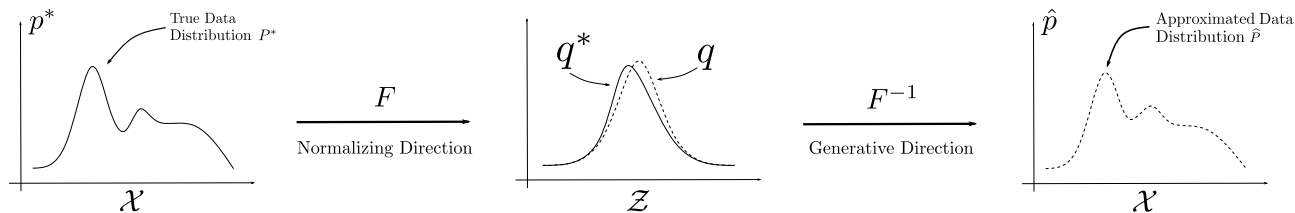


Figure. 1D Normalizing Flow process.

NORMALIZING FLOWS

CHANGE OF VARIABLES

Change of Variables Formula

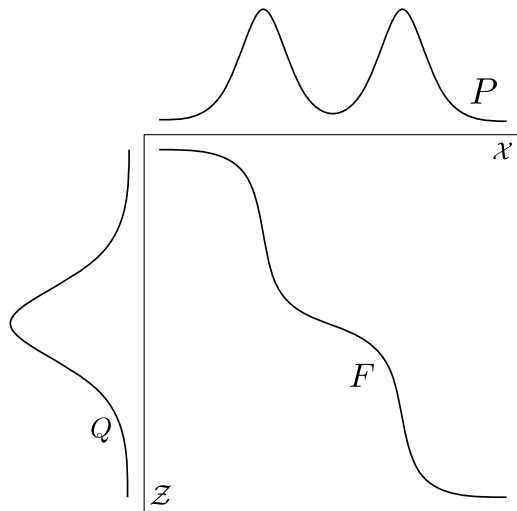
For a bijective and continuous function F and a latent distribution Q , the distribution induced by Q and F is defined by:

$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = |\det \text{Jac}_F(x)| q(F(x)). \quad (1)$$

NORMALIZING FLOWS

CHANGE OF VARIABLES — VISUAL INTUITION

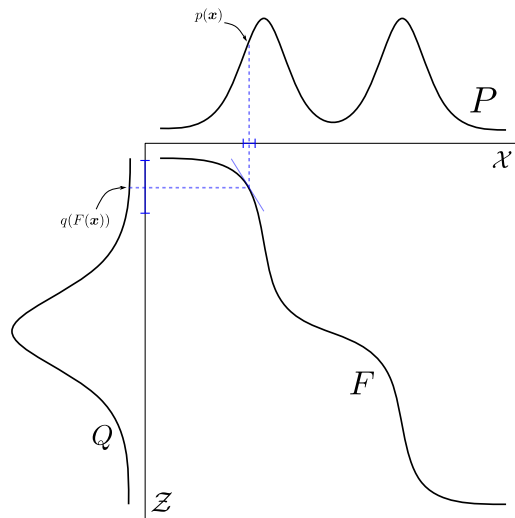
$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = |\det \text{Jac}_F(x)| q(F(x)).$$



NORMALIZING FLOWS

CHANGE OF VARIABLES — VISUAL INTUITION

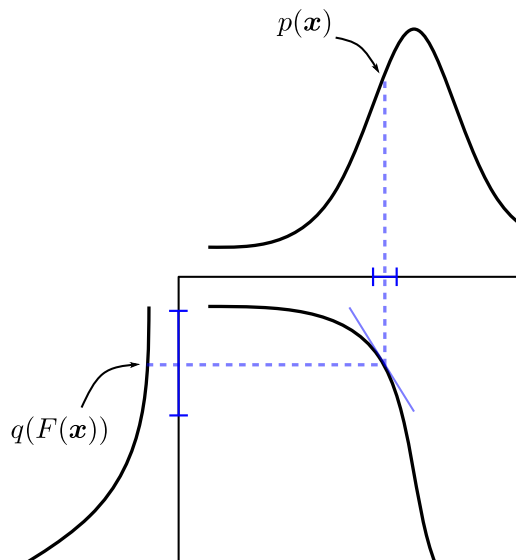
$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = |\det \text{Jac}_F(x)| q(F(x)).$$



NORMALIZING FLOWS

CHANGE OF VARIABLES — VISUAL INTUITION

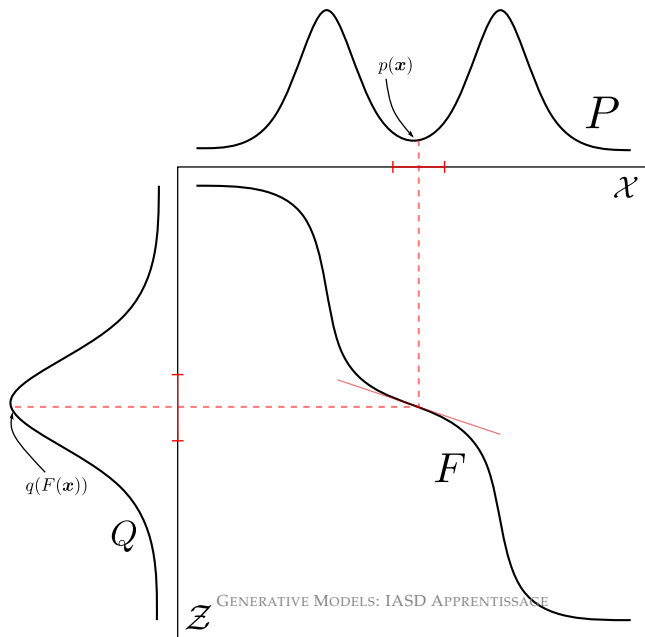
$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = |\det \text{Jac}_F(x)| q(F(x)).$$



NORMALIZING FLOWS

CHANGE OF VARIABLES — VISUAL INTUITION

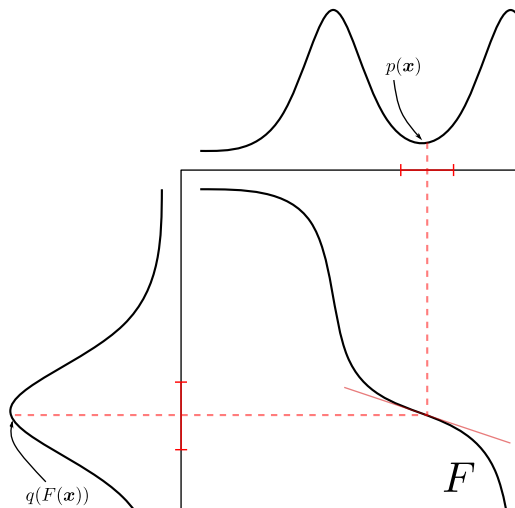
$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = |\det \text{Jac}_F(x)| q(F(x)).$$



NORMALIZING FLOWS

CHANGE OF VARIABLES — VISUAL INTUITION

$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = |\det \text{Jac}_F(x)| q(F(x)).$$



NORMALIZING FLOWS

DENSITY ESTIMATION

To perform density estimation:

1. Draw $x \sim P$,
2. Compute $F(x)$ and $|\det \text{Jac}_F(x)|$,
3. Compute $\hat{p}(x) = q(F(x)) |\det \text{Jac}_F(x)|$.

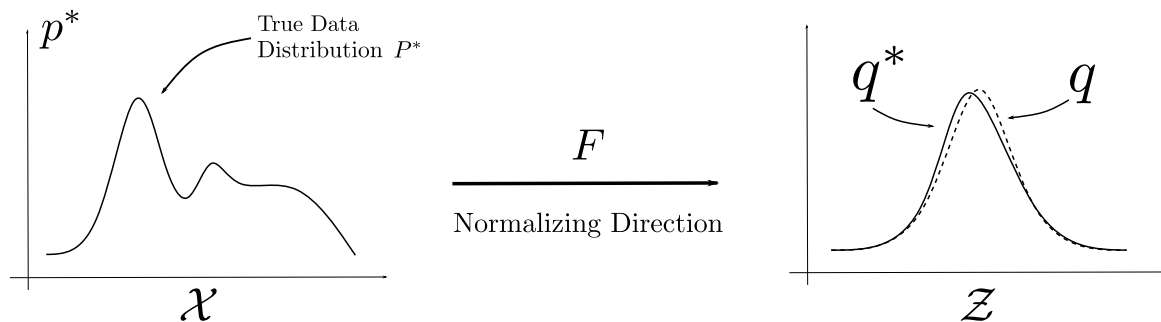


Figure. 1D Normalizing Flow process of density estimation.

NORMALIZING FLOWS

DATA GENERATION

To perform data generation:

1. Draw $z \sim Q$,
2. Compute $x = F^{-1}(z)$.

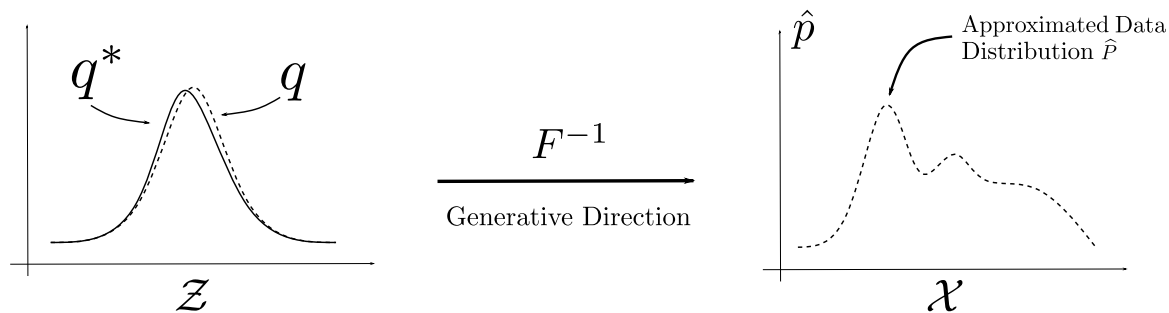


Figure. 1D Normalizing Flow process of generation.

NORMALIZING FLOWS

LEARNING STEPS

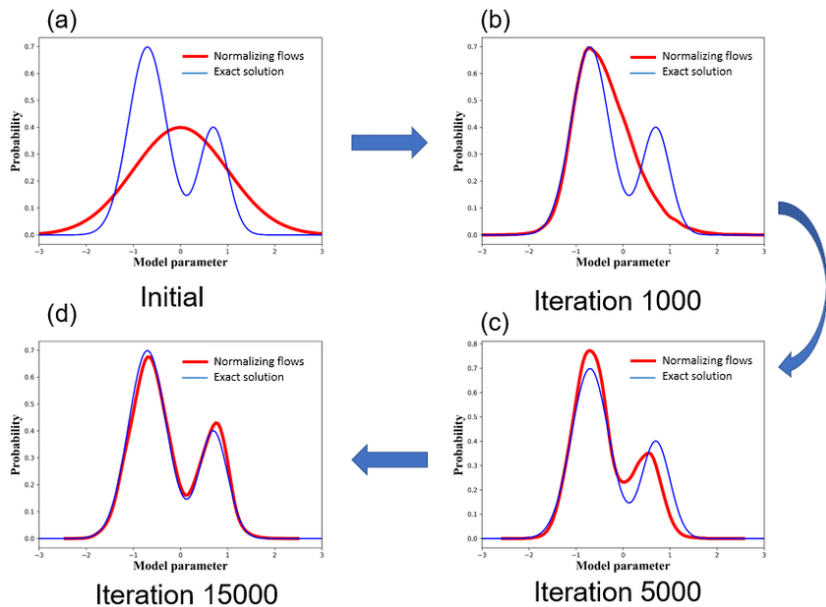


Figure. Learning process for a 1D Normalizing Flow.

ENERGY-BASED MODELS

PARTITION FUNCTION CHALLENGE

Definition. An Energy-Based Model (EBM) defines

$$p_{\theta}(x) = \frac{\exp(-E_{\theta}(x))}{Z(\theta)}, \quad Z(\theta) = \int \exp(-E_{\theta}(x)) dx.$$

- ▶ $E_{\theta}(x)$ is an *energy function* (low energy = high probability).
- ▶ $Z(\theta)$ is the *partition function* ensuring normalization.
- ▶ Problem: computing $Z(\theta)$ is generally intractable (high-dimensional integral).

Takeaway: normalization constant is the main pain in EBMs.

ENERGY-BASED MODELS

TRAINING — LOG-LIKELIHOOD AND GRADIENT

Log-likelihood for one sample x :

$$\log p_\theta(x) = -E_\theta(x) - \log Z(\theta), \quad \text{with } Z(\theta) = \int e^{-E_\theta(u)} du.$$

Gradient derivation:

$$\nabla_\theta \log p_\theta(x) = -\nabla_\theta E_\theta(x) - \nabla_\theta \log Z(\theta)$$

Compute $\nabla_\theta \log Z(\theta)$ explicitly:

$$\begin{aligned} Z(\theta) &= \int e^{-E_\theta(u)} du, & \nabla_\theta Z(\theta) &= \int e^{-E_\theta(u)} (-\nabla_\theta E_\theta(u)) du, \\ \nabla_\theta \log Z(\theta) &= \frac{1}{Z(\theta)} \nabla_\theta Z(\theta) \\ &= - \int \frac{e^{-E_\theta(u)}}{Z(\theta)} \nabla_\theta E_\theta(u) du \\ &= - \mathbb{E}_{u \sim p_\theta} [\nabla_\theta E_\theta(u)]. \end{aligned}$$

Substitute back:

$$\nabla_\theta \log p_\theta(x) = -\nabla_\theta E_\theta(x) - \nabla_\theta \log Z(\theta) = -\nabla_\theta E_\theta(x) + \mathbb{E}_{u \sim p_\theta} [\nabla_\theta E_\theta(u)].$$

ENERGY-BASED MODELS

TRAINING — PRACTICAL GRADIENT STEP

Empirical objective (dataset $\{x_n\}$):

$$\nabla_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_{\theta}(x_n) = -\frac{1}{N} \sum_{n=1}^N \nabla_{\theta} E_{\theta}(x_n) + \mathbb{E}_{u \sim p_{\theta}} [\nabla_{\theta} E_{\theta}(u)].$$

Gradient step (schematic):

$$\theta \leftarrow \theta - \eta \left(-\frac{1}{N} \sum_n \nabla_{\theta} E_{\theta}(x_n) + \mathbb{E}_{u \sim p_{\theta}} [\nabla_{\theta} E_{\theta}(u)] \right).$$

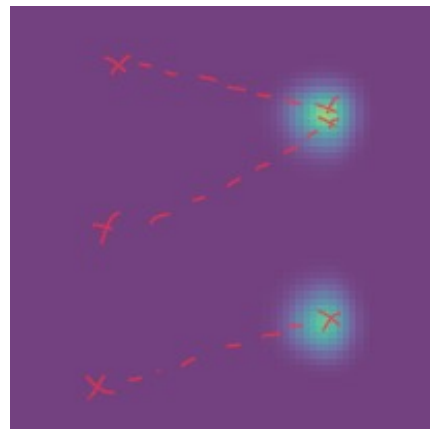
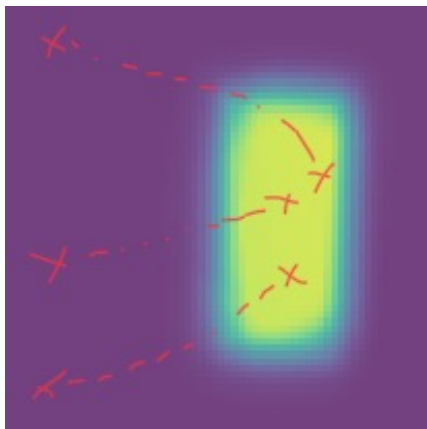
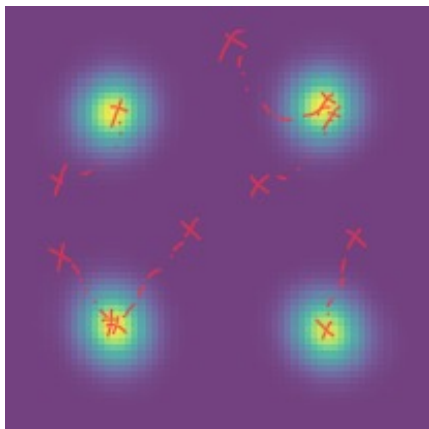
- ▶ *Pull down* energy on data (first term), *push up* on model samples (second term).
- ▶ Trade-offs: bias vs. mixing time; stability tricks (noise scale, step size, gradient clipping, spectral norm).

ENERGY-BASED MODELS

SAMPLING

Goal: generate $x \sim p_\theta(x) \propto e^{-E_\theta(x)}$.

- ▶ Direct sampling is impossible (requires $Z(\theta)$).
- ▶ Use MCMC methods (e.g., Langevin dynamics, Hamiltonian Monte Carlo).
- ▶ Iteratively update $x \leftarrow x - \eta \nabla_x E_\theta(x) + \sqrt{2\eta} \xi$, $\xi \sim \mathcal{N}(0, I)$.
- ▶ Paths follow the energy landscape toward low-energy regions (data modes).



AUTOENCODER

BASIC ARCHITECTURE AND OBJECTIVE

Architecture:

- ▶ **Encoder:** maps input x to a low-dimensional latent representation $z = f_\phi(x)$.
- ▶ **Decoder:** reconstructs input from latent z , i.e., $\hat{x} = g_\theta(z)$.
- ▶ The model is trained end-to-end to minimize the difference between x and $\hat{x}$.

Objective:

$$\min_{\theta, \phi} \mathbb{E}_{x \sim P_{\text{data}}} [\ell(x, g_\theta(f_\phi(x)))]$$

where ℓ is typically mean squared error: $\ell(x, \hat{x}) = \|x - \hat{x}\|^2$. **Limitations for generative modeling:**

- ▶ No explicit generative process for sampling new data from the latent space.
- ▶ Latent space may not follow a known distribution—sampling z at random often yields unrealistic outputs.
- ▶ Not a true probabilistic model; lacks explicit likelihood or regularization of latent space.

AUTOENCODER

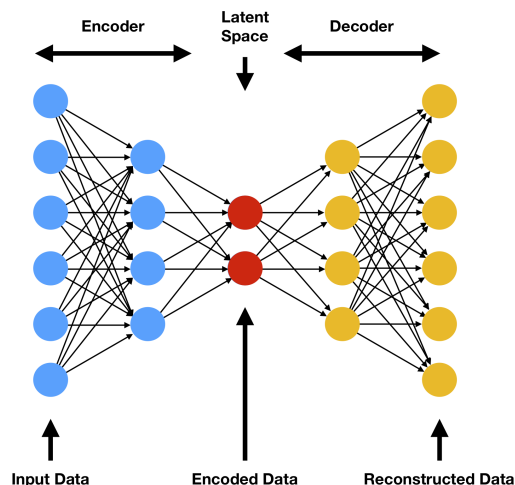
ILLUSTRATION AND EXAMPLE

Architecture:

- ▶ **Input:** x (e.g., image, signal)
- ▶ **Encoder:** compresses x to latent z
- ▶ **Decoder:** reconstructs $\hat{x}$ from z

Applications:

- ▶ Dimensionality reduction, denoising, anomaly detection, feature learning
- ▶ Not directly suited for generating novel samples



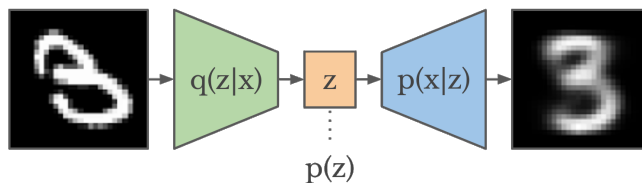
VARIATIONAL AUTOENCODER

INTUITION BEHIND VAE

Key ideas:

- ▶ **Probabilistic encoder:** Instead of mapping $x \rightarrow z$ deterministically, encode x as a *distribution* over latent variables: $q_\phi(z|x)$ (e.g., Gaussian with mean and variance predicted by encoder).
- ▶ **Probabilistic decoder:** Model $p_\theta(x|z)$, i.e., generate x from latent z .
- ▶ **Latent variable modeling:** Place a prior $p(z)$ (usually standard normal) on the latent space to encourage structure and enable sampling.
- ▶ **Regularization:** Use KL divergence $\mathcal{D}_{\text{KL}}(q_\phi(z|x) \| p(z))$ to encourage $q_\phi(z|x)$ to be close to the prior, making the latent space well-behaved and suitable for generative sampling.

Summary: VAE is a probabilistic autoencoder that learns both to reconstruct data and to regularize the latent space for generative use.



VARIATIONAL AUTOENCODER

EVIDENCE LOWER BOUND (ELBO)

Objective: Evidence Lower Bound (ELBO)

$$\log p_{\theta}(x) \geq \mathbb{E}_{q_{\phi}(z|x)}[\log p_{\theta}(x|z)] - \mathcal{D}_{\text{KL}}(q_{\phi}(z|x) || p(z))$$

- ▶ **First term:** Expected log-likelihood (reconstruction accuracy)
- ▶ **Second term:** KL divergence between encoder distribution and prior (regularization)

Relation to MLE: Maximizing ELBO approximates maximizing the marginal likelihood $p_{\theta}(x)$ (i.e., maximum likelihood estimation for latent variable models).

Training strategy:

- ▶ Optimize the ELBO jointly with respect to encoder (ϕ) and decoder (θ) parameters.
- ▶ Use stochastic gradient descent with the reparameterization trick to backpropagate through stochastic nodes.

VARIATIONAL AUTOENCODER

REPARAMETERIZATION TRICK

Intuition:

- ▶ Allows gradients to flow through random sampling by expressing sampling as a deterministic function of parameters and noise.
- ▶ Enables efficient and low-variance gradient estimation for stochastic variables.

Mathematical formulation:

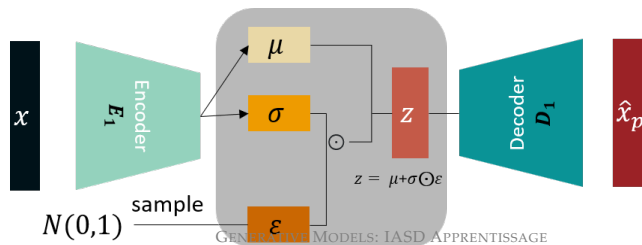
- ▶ For $q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x)^2)$,
- ▶ Sample z as:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

- ▶ Now, z is a deterministic function of x , ϕ , and random noise ϵ .

Benefits:

- ▶ Enables backpropagation through stochastic sampling.
- ▶ Crucial for training VAEs with gradient-based methods.



VARIATIONAL AUTOENCODER

LOSS

VAE Loss Function:

$$\mathcal{L}_{\text{VAE}}(x) = \underbrace{\mathbb{E}_{q_{\phi}(z|x)}[-\log p_{\theta}(x|z)]}_{\text{Reconstruction loss}} + \underbrace{\mathcal{D}_{\text{KL}}(q_{\phi}(z|x) || p(z))}_{\text{KL divergence (regularization)}}$$

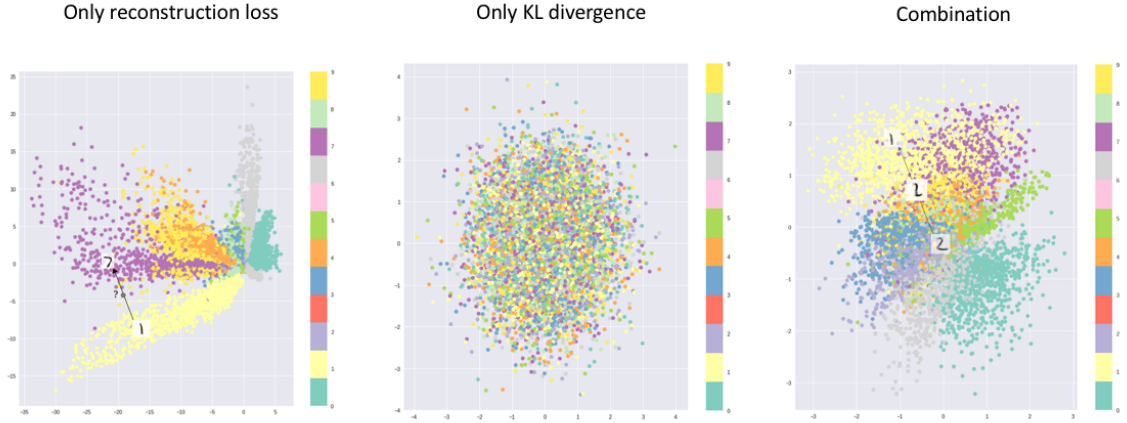
- ▶ **Reconstruction loss:** Measures how well the decoder can reconstruct the input from the latent code.
- ▶ **KL divergence:** Encourages the approximate posterior $q_{\phi}(z|x)$ to match the prior $p(z)$ (e.g., standard normal), regularizing the latent space.
- ▶ **Trade-off:** Balances data fidelity (reconstruction) and latent space regularity (generative quality). Too much weight on KL: blurry reconstructions; too little: latent space collapse.

Typical formula:

$$\mathcal{L}_{\text{VAE}}(x) = \frac{1}{2} \sum_j \left(\sigma_j^2(x) + \mu_j(x)^2 - 1 - \log \sigma_j^2(x) \right) + \text{Reconstruction loss}$$

VARIATIONAL AUTOENCODER

Loss



VARIANTS OF VAE

EXTENSIONS AND IMPROVEMENTS

Key VAE Variants:

- ▶ **Conditional VAE (CVAE):** Conditions both encoder and decoder on auxiliary information (e.g., labels, attributes) to enable conditional generation [26].
- ▶ **β -VAE:** Introduces a hyperparameter β to scale the KL term, encouraging disentangled latent representations [8].
- ▶ **Other notable extensions:**
 - *VampPrior*: Learnable mixture prior for more flexible latent space.
 - *Vector Quantized VAE (VQ-VAE)* [22]: Discrete latent variables via vector quantization.
 - *Hierarchical VAE*: Multiple layers of latent variables.
 - *FactorVAE*, *InfoVAE*, *WAE* (Wasserstein Autoencoder), etc.

References:

- ▶ Original VAE paper: [14]
- ▶ Conditional VAE: [26]
- ▶ β -VAE : [8]
- ▶ VQ-VAE: [22]

VQ-VAE (VAN DEN OORD ET AL., 2017)

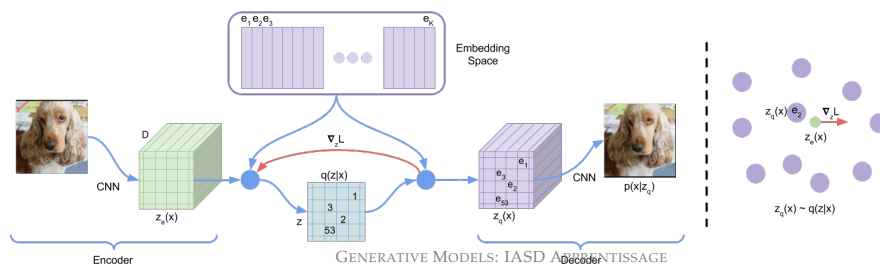
VECTOR QUANTIZED VAE

Key ideas:

- ▶ Introduces **discrete latent representations** via *vector quantization*—the encoder outputs are mapped to the nearest entry in a learned codebook.
- ▶ The decoder reconstructs x from the quantized latent code.
- ▶ Enables modeling of discrete structure in data (e.g., language, audio, images).
- ▶ Discrete latents are particularly beneficial for combining VAEs with powerful generative models (such as PixelCNN or Transformers) in the latent space.
- ▶ Facilitates improved sample quality and more interpretable representations.

Benefits:

- ▶ Enables use of GAN-like or autoregressive models in discrete latent space.
- ▶ Improved performance on high-fidelity image and audio generation tasks.



Part II

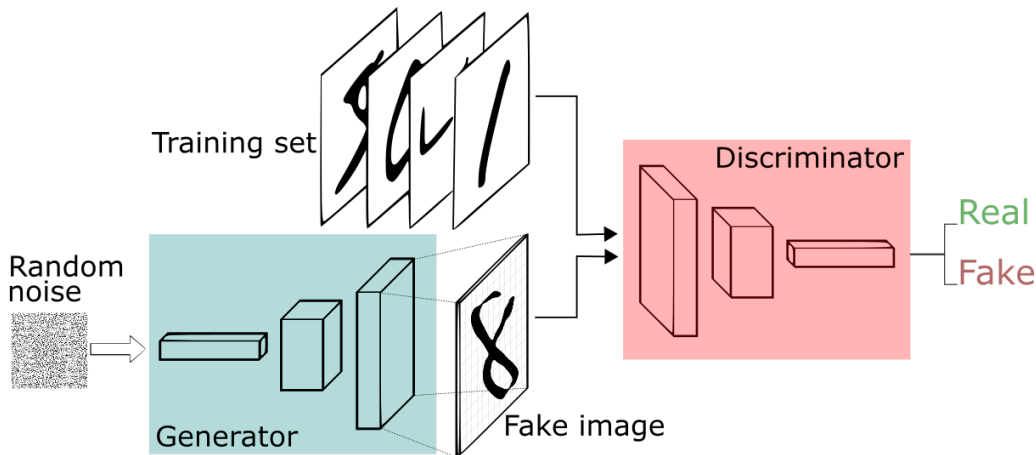
GENERATIVE ADVERSARIAL NETWORKS

GENERATIVE ADVERSARIAL NETWORKS (GANs)

INTRO: YEAR, REFERENCE, ROLES OF D & G

Introduced in: 2014a by Goodfellow et al. [6].

- ▶ **Discriminator** $D(x)$: estimates how likely a sample x is *real* (vs. generated).
- ▶ **Generator** $G(z)$: maps noise $z \sim Q$ to data space and aims to *fool* D .



TRAINING THE DISCRIMINATOR AS A CLASSIFIER

LABELS, LOSS SELECTION, BCE/LOGISTIC

Setup: assign labels $y \in \{-1, +1\}$ with $y = +1$ for real $x \sim P$ and $y = -1$ for fake $x \sim \hat{P}$.

Goal: Train a classifier parametrized by $D = \sigma(F_\theta)$ to distinguish real vs fake.

Question: which loss to train D as a classifier?

SIGMOID AND LOGIT

ACTIVATION & LINK FUNCTIONS

- ▶ The **sigmoid function** $\sigma : \mathbb{R} \rightarrow [0, 1]$ is defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \quad (2)$$

- ▶ It maps any real-valued number x to the interval $[0, 1]$, suitable for modeling probabilities.
- ▶ The **logit function** $\text{logit} : [0, 1] \rightarrow \mathbb{R}$ is defined as the inverse of the sigmoid:

$$\text{logit}(p) = \log \left(\frac{p}{1 - p} \right). \quad (3)$$

- ▶ Let $D(x) = \sigma(F_\theta(x))$ be a discriminator outputting probabilities. We call $F_\theta(x)$ the **logit** of D and $D(x)$ the **probit**.

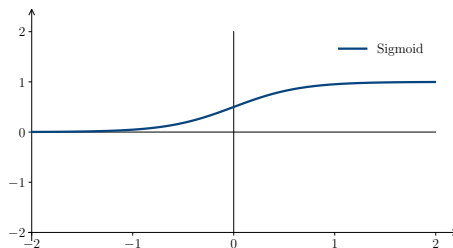


Figure. Sigmoid function mapping logits to probabilities.

TRAINING THE DISCRIMINATOR AS A CLASSIFIER

QUESTION → ANSWER → LOSS DECOMPOSITION

Setup: assign labels $y \in \{-1, +1\}$ with $y = +1$ for real $x \sim P$ and $y = -1$ for fake $x \sim \hat{P}$.

Goal: Train a classifier parametrized by $D = \sigma(F_\theta)$ to distinguish real vs fake.

Question: which loss to train D as a classifier?

TRAINING THE DISCRIMINATOR AS A CLASSIFIER

QUESTION → ANSWER → LOSS DECOMPOSITION

Setup: assign labels $y \in \{-1, +1\}$ with $y = +1$ for real $x \sim P$ and $y = -1$ for fake $x \sim \hat{P}$.

Goal: Train a classifier parametrized by $D = \sigma(F_\theta)$ to distinguish real vs fake.

Answer: binary cross-entropy / logistic loss (with logits).

TRAINING THE DISCRIMINATOR AS A CLASSIFIER

QUESTION → ANSWER → LOSS DECOMPOSITION

Setup: assign labels $y \in \{-1, +1\}$ with $y = +1$ for real $x \sim P$ and $y = -1$ for fake $x \sim \hat{P}$.

Goal: Train a classifier parametrized by $D = \sigma(F_\theta)$ to distinguish real vs fake.

Answer: binary cross-entropy / logistic loss (with logits).

Loss with labels:

$$\mathcal{L}_D = \mathbb{E}_{(x,y)} \left[\log(1 + \exp(-y F_\theta(x))) \right].$$

Decomposition over P and $\hat{P}$:

$$\begin{aligned} \mathcal{L}_D &= \mathbb{E}_{x \sim P} [\log(1 + \exp(-F_\theta(x)))] + \mathbb{E}_{x \sim \hat{P}} [\log(1 + \exp(F_\theta(x)))] \\ &= \mathbb{E}_{x \sim P} \left[-\log \left(\frac{1}{1 + \exp(-F_\theta(x))} \right) \right] + \mathbb{E}_{x \sim \hat{P}} \left[-\log \left(1 - \frac{1}{1 + \exp(-F_\theta(x))} \right) \right] \\ &= -\mathbb{E}_{x \sim P} [\log \sigma(F_\theta(x))] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - \sigma(F_\theta(x)))] \\ &= -\mathbb{E}_{x \sim P} [\log D(x)] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - D(x))]. \end{aligned}$$

TRAINING THE DISCRIMINATOR AS A CLASSIFIER

QUESTION → ANSWER → LOSS DECOMPOSITION

Setup: assign labels $y \in \{-1, +1\}$ with $y = +1$ for real $x \sim P$ and $y = -1$ for fake $x \sim \hat{P}$.

Goal: Train a classifier parametrized by $D = \sigma(F_\theta)$ to distinguish real vs fake.

Answer: binary cross-entropy / logistic loss (with logits).

Loss with labels:

$$\mathcal{L}_D = \mathbb{E}_{(x,y)} \left[\log(1 + \exp(-y F_\theta(x))) \right].$$

Decomposition over P and $\hat{P}$:

$$\begin{aligned} \mathcal{L}_D &= \mathbb{E}_{x \sim P} [\log(1 + \exp(-F_\theta(x)))] + \mathbb{E}_{x \sim \hat{P}} [\log(1 + \exp(F_\theta(x)))] \\ &= \mathbb{E}_{x \sim P} \left[-\log \left(\frac{1}{1 + \exp(-F_\theta(x))} \right) \right] + \mathbb{E}_{x \sim \hat{P}} \left[-\log \left(1 - \frac{1}{1 + \exp(-F_\theta(x))} \right) \right] \\ &= -\mathbb{E}_{x \sim P} [\log \sigma(F_\theta(x))] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - \sigma(F_\theta(x)))] \\ &= -\mathbb{E}_{x \sim P} [\log D(x)] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - D(x))]. \end{aligned}$$

Summarizing:

$$\text{Minimizing } \mathcal{L}_D = -\mathbb{E}_{x \sim P} [\log D(x)] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - D(x))].$$

TRAINING THE DISCRIMINATOR AS A CLASSIFIER

QUESTION → ANSWER → LOSS DECOMPOSITION

Setup: assign labels $y \in \{-1, +1\}$ with $y = +1$ for real $x \sim P$ and $y = -1$ for fake $x \sim \hat{P}$.

Goal: Train a classifier parametrized by $D = \sigma(F_\theta)$ to distinguish real vs fake.

Answer: binary cross-entropy / logistic loss (with logits).

Loss with labels:

$$\mathcal{L}_D = \mathbb{E}_{(x,y)} \left[\log(1 + \exp(-y F_\theta(x))) \right].$$

Decomposition over P and $\hat{P}$:

$$\begin{aligned} \mathcal{L}_D &= \mathbb{E}_{x \sim P} [\log(1 + \exp(-F_\theta(x)))] + \mathbb{E}_{x \sim \hat{P}} [\log(1 + \exp(F_\theta(x)))] \\ &= \mathbb{E}_{x \sim P} \left[-\log \left(\frac{1}{1 + \exp(-F_\theta(x))} \right) \right] + \mathbb{E}_{x \sim \hat{P}} \left[-\log \left(1 - \frac{1}{1 + \exp(-F_\theta(x))} \right) \right] \\ &= -\mathbb{E}_{x \sim P} [\log \sigma(F_\theta(x))] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - \sigma(F_\theta(x)))] \\ &= -\mathbb{E}_{x \sim P} [\log D(x)] - \mathbb{E}_{x \sim \hat{P}} [\log(1 - D(x))]. \end{aligned}$$

Summarizing:

$$\text{Maximizing } \mathcal{L}_D = \mathbb{E}_{x \sim P} [\log D(x)] + \mathbb{E}_{x \sim \hat{P}} [\log(1 - D(x))].$$

TRAINING THE GENERATOR

MINIMAX VS NON-SATURATING OBJECTIVES

Goal: decrease D 's confidence on $\hat{P}$ (make fakes look real).

TRAINING THE GENERATOR

MINIMAX VS NON-SATURATING OBJECTIVES

Goal: decrease D 's confidence on $\hat{P}$ (make fakes look real).

$$\text{Maximizing } \mathcal{L}_G = -\mathbb{E}_{x \sim \hat{P}}[\log(1 - D(x))]$$

TRAINING THE GENERATOR

MINIMAX VS NON-SATURATING OBJECTIVES

Goal: decrease D 's confidence on $\hat{P}$ (make fakes look real).

$$\text{Maximizing } \mathcal{L}_G = -\mathbb{E}_{z \sim Q}[\log(1 - D(G(z)))]$$

TRAINING THE GENERATOR

MINIMAX VS NON-SATURATING OBJECTIVES

Goal: decrease D 's confidence on $\hat{P}$ (make fakes look real).

$$\text{Minimizing } \mathcal{L}_G = \mathbb{E}_{z \sim Q} [\log(1 - D(G(z)))]$$

TRAINING THE GENERATOR

MINIMAX VS NON-SATURATING OBJECTIVES

Goal: decrease D 's confidence on $\hat{P}$ (make fakes look real).

$$\text{Minimizing } \mathcal{L}_G = \mathbb{E}_{z \sim Q}[\log(1 - D(G(z)))]$$

Minimax formulation: combine both objectives:

$$\min_G \max_D \mathbb{E}_{x \sim P}[\log D(x)] + \mathbb{E}_{z \sim Q}[\log(1 - D(G(z)))]$$

ORIGINAL GAN: FIGURES

TRAINING LOOP & DISTRIBUTIONS

Example of generated samples during training from Goodfellow et al. [6]. In dotted black: the real data distribution, in blue the estimated probabilities from the discriminator and in green the generated distribution.

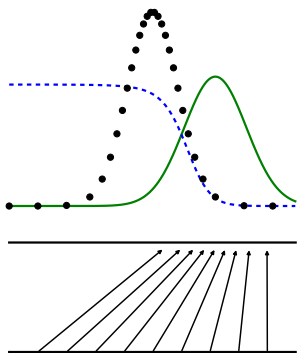


Figure. Before

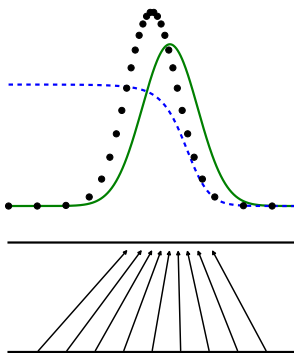


Figure. During

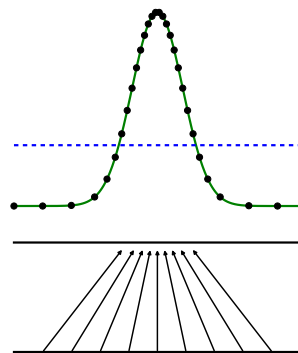


Figure. After

ORIGINAL GAN: GENERATION EXAMPLES

REAL VS GENERATED COMPARISONS

The right column shows the closest real samples to the generated ones on the left, illustrating that the model is not simply memorizing training data.

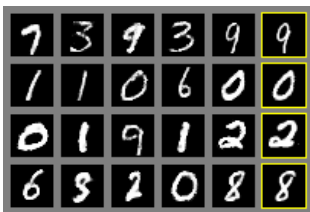


Figure. MNIST samples

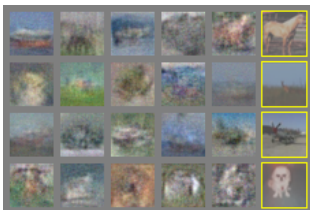


Figure. CIFAR-10 samples (FC)

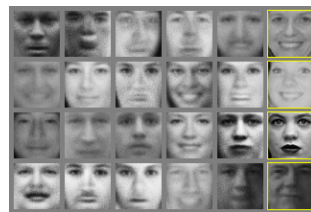


Figure. Toronto Face Dataset samples

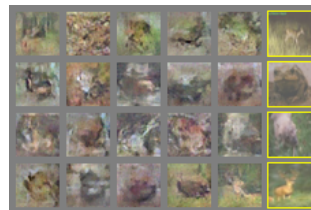
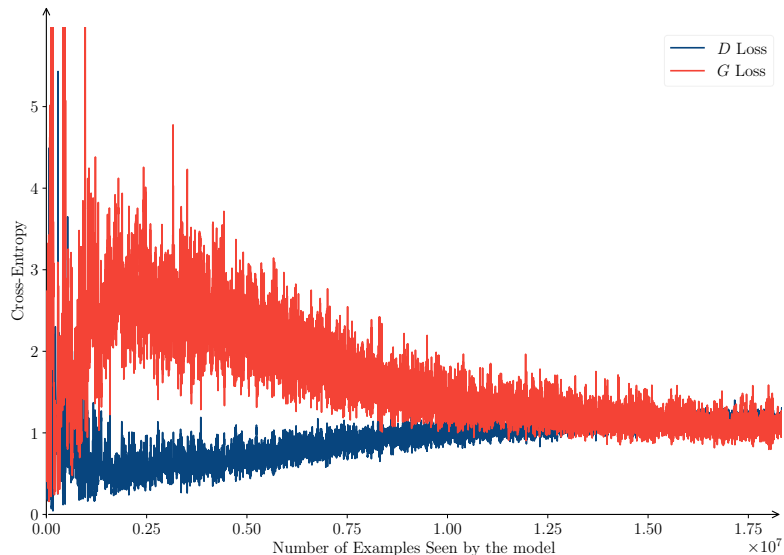


Figure. CIFAR-10 samples (ConvNet)

ORIGINAL GAN: LOSS FUNCTIONS

DISCRIMINATOR AND GENERATOR LOSSES

Example of loss curves during training for MNIST:

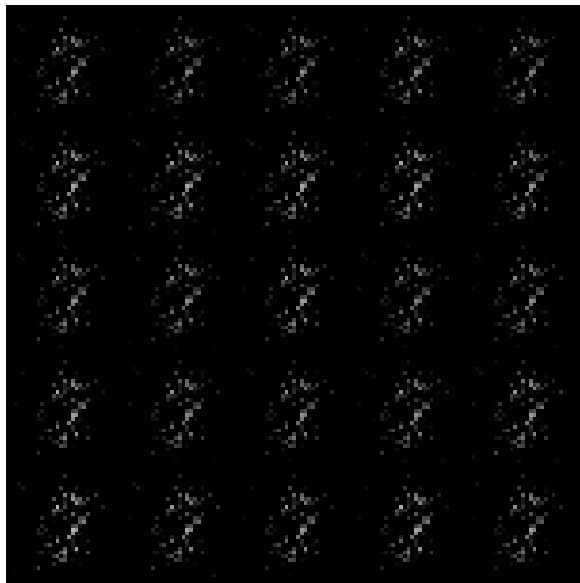


Observation: One loss is increasing while the other decreases, indicating the adversarial nature of the training.

GAN TRAINING PROGRESSION

SAMPLES ACROSS EPOCHS

Generated samples from a GAN trained on MNIST over different training iterations:

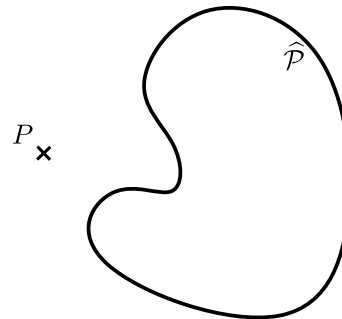


WHY GANs WORK: DIVERGENCE MINIMIZATION

MINIMIZE $D(P||\hat{P}_G)$

Traditionally, the goal is to minimize *a dissimilarity measure* between the target distribution P and the learned distribution $\hat{P}$:

$$\min_G D(P, \hat{P}_G) \quad (4)$$



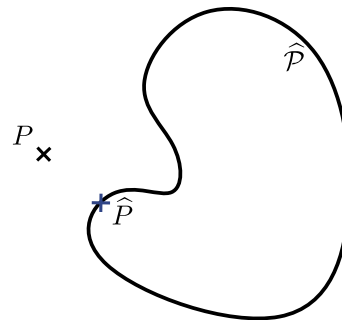
Distribution Space

WHY GANs WORK: DIVERGENCE MINIMIZATION

MINIMIZE $D(P||\hat{P}_G)$

Traditionally, the goal is to minimize *an f -divergence* between the target distribution P and the learned distribution $\hat{P}$:

$$\min_G \mathcal{D}_f(P||\hat{P}_G) \quad (4)$$



Distribution Space

FROM DIVERGENCE MINIMIZATION TO f -DIVERGENCES

NOWOZIN ET AL. (2016) f -GAN

f -divergences form a broad class of dissimilarity measures between probability distributions:

Definition 1.1 (f -divergences)

For any two probability distributions P and $\hat{P}$ in $\mathcal{P}(\mathcal{X})$ such that $P, \hat{P} \ll \mu$. Let p and $\hat{p}$ be the Radon-Nikodym densities of P and $\hat{P}$ with respect to μ , respectively. Let f be any convex lower semi-continuous function $f : [0, \infty] \rightarrow [-\infty, +\infty]$ such that $f(1) = 0$, the f -divergence between P and $\hat{P}$ is

$$\mathcal{D}_f(P \parallel \hat{P}) = \int_{\mathcal{X}} \hat{p}(\mathbf{x}) f\left(\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})}\right) d\mu(\mathbf{x}). \quad (5)$$

FROM DIVERGENCE MINIMIZATION TO f -DIVERGENCES

NOWOZIN ET AL. (2016) f -GAN

f -divergences form a broad class of dissimilarity measures between probability distributions:

Definition 1.1 (f -divergences)

For any two probability distributions P and $\hat{P}$ in $\mathcal{P}(\mathcal{X})$ such that $P, \hat{P} \ll \mu$. Let p and $\hat{p}$ be the Radon-Nikodym densities of P and $\hat{P}$ with respect to μ , respectively. Let f be any convex lower semi-continuous function $f : [0, \infty] \rightarrow [-\infty, +\infty]$ such that $f(1) = 0$, the f -divergence between P and $\hat{P}$ is

$$\mathcal{D}_f(P \parallel \hat{P}) = \int_{\mathcal{X}} \hat{p}(\mathbf{x}) f\left(\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})}\right) d\mu(\mathbf{x}). \quad (5)$$

They can be written as expectations over $\hat{P}$:

$$\mathcal{D}_f(P \parallel \hat{P}) = \mathbb{E}_{\mathbf{x} \sim \hat{P}} \left[f\left(\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})}\right) \right]$$

With this formulation, f can be interpreted as a penalty function on the density ratio $p(x)/\hat{p}(x)$. The further from 1 this ratio is, the higher the penalty.

EXAMPLES OF f -DIVERGENCES

GENERATOR FUNCTIONS $f(u)$

Name	Generator $f(u)$	Comment
Kullback–Leibler (KL)	$u \log u$	mode-covering
Reverse KL	$-\log u$	mode-seeking
Jensen–Shannon (JS)	$(u + 1) \log \frac{u+1}{2} - u \log u$	bounded, symmetric
Total variation (TV)	$\frac{1}{2} u - 1 $	metric
Hellinger ²	$(\sqrt{u} - 1)^2$	metric
Pearson χ^2	$(u - 1)^2$	emphasizes q small
Neyman χ^2	$\frac{(1 - u)^2}{u}$	emphasizes p small

Table. Examples of f -divergence generators. *Note:* f is defined up to an affine term $a(u - 1)$, which leaves $\mathcal{D}_f$ unchanged; forms are chosen with $f(1) = 0$.

FROM THE FIRST GAN TO f -GAN AND WGAN

EXAMPLES OF f AND $\mathcal{D}_f$

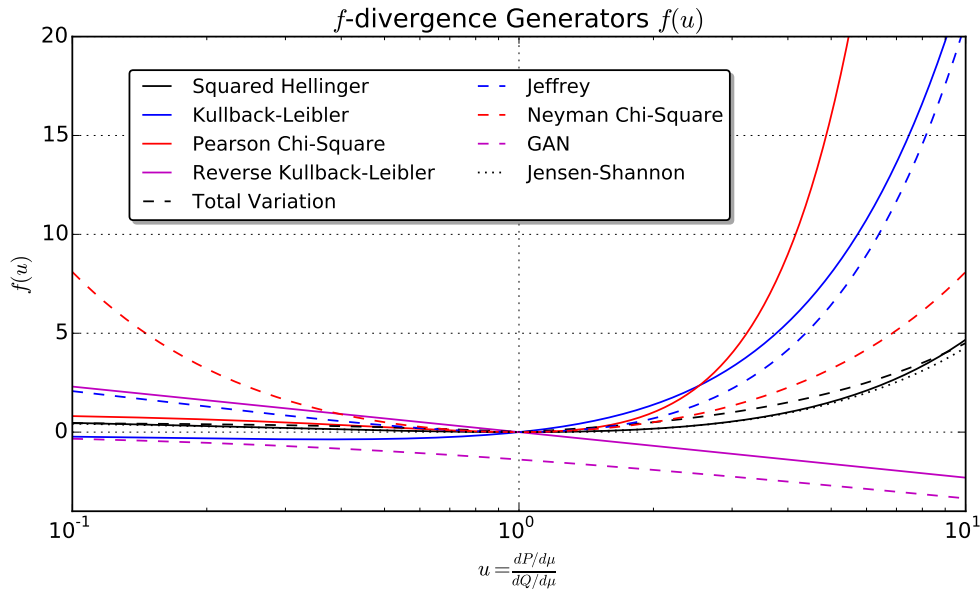


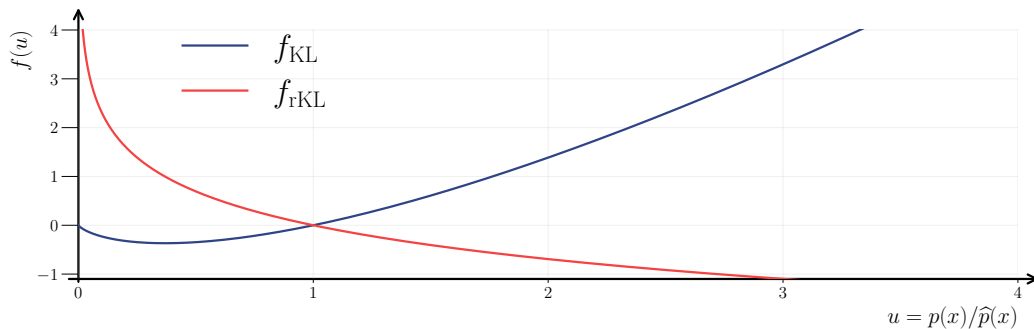
Figure. Common f -divergences with their corresponding f functions. Source: Nowozin et al. [21].

FROM THE FIRST GAN TO f -GAN AND WGAN

EFFECT OF THE f -DIVERGENCE ON THE LEARNED DISTRIBUTION

All f -divergences are not equal:

$$\mathcal{D}_f(P \parallel \hat{P}) = \mathbb{E}_{\mathbf{x} \sim \hat{P}} \left[f \left(\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})} \right) \right]$$



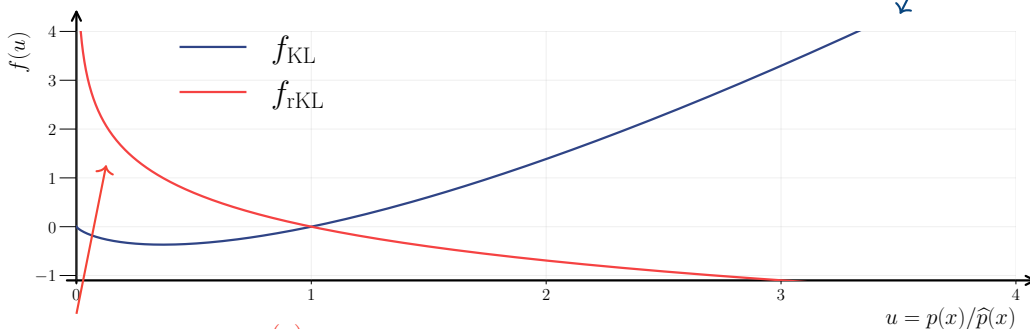
FROM THE FIRST GAN TO f -GAN AND WGAN

EFFECT OF THE f -DIVERGENCE ON THE LEARNED DISTRIBUTION

All f -divergences are not equal:

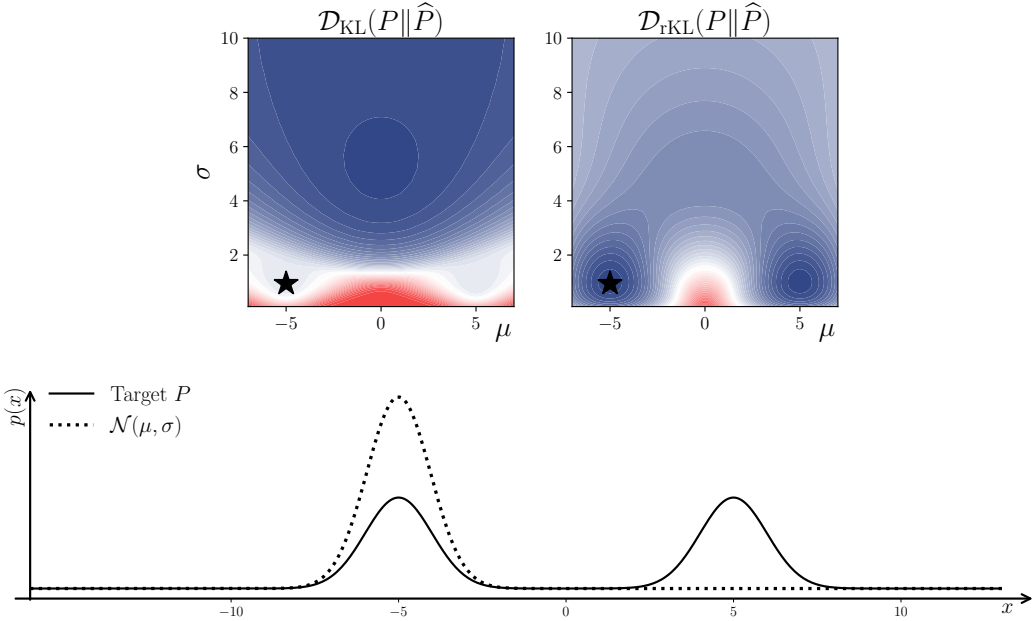
$$\mathcal{D}_f(P\|\hat{P}) = \mathbb{E}_{\mathbf{x} \sim \hat{P}} \left[f \left(\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})} \right) \right]$$

Penalizing high values of $\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})}$

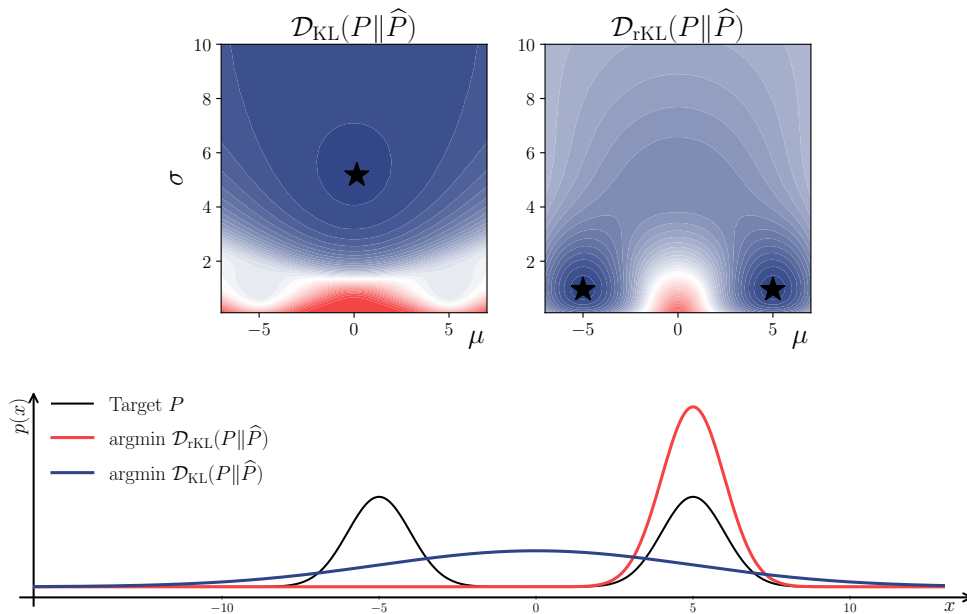


Penalizing low values of $\frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})}$

EXAMPLES OF f -DIVERGENCE MINIMIZATION



EXAMPLES OF f -DIVERGENCE MINIMIZATION



WHY DIRECT f -DIVERGENCE ESTIMATION IS HARD

IMPLICIT MODELS & UNKNOWN DENSITY RATIOS

- ▶ **Implicit generator:** $G(z)$ defines $\hat{P}$ only via samples; no tractable density $\hat{p}$.
- ▶ **Definition requires a density ratio:**

$$\mathcal{D}_f(P\|\hat{P}) = \mathbb{E}_{x \sim \hat{P}} \left[f \left(\underbrace{\frac{p(x)}{\hat{p}(x)}}_{\text{unknown}} \right) \right].$$

- ▶ **Unavailable ratio $p/\hat{p}$:** dataset provides samples from P ; the generator provides samples from $\hat{P}$, but neither p nor $\hat{p}$ is known.
- ▶ **High-dimensional densities are hard:** plug-in density estimation is sample-inefficient and unstable; numerical integration over $\mathcal{X}$ is intractable.
- ▶ **Need a workaround:** leverage samples from both P and $\hat{P}$ while *avoiding* explicit densities $\Rightarrow$ a variational/dual form.

FROM THE FIRST GAN TO f -GAN AND WGAN

FENCHEL (LEGENDRE-FENCHEL) CONJUGATE

Definition (Fenchel conjugate). For a proper, convex, lower semicontinuous $f : \mathbb{R} \rightarrow \mathbb{R} \cup \{+\infty\}$,

$$f^*(t) = \sup_{u \in \mathbb{R}} \{ t u - f(u) \}.$$

Key facts:

- ▶ *Fenchel–Young inequality:* $f(u) + f^*(t) \geq t u$, with equality when $t \in \partial f(u)$.
- ▶ *Biconjugate / envelope:* for convex l.s.c. f , $f(u) = \sup_{t \in \mathbb{R}} \{ t u - f^*(t) \}$.

Intuition: f is the *upper envelope* of its supporting affine functions $u \mapsto t u - f^*(t)$ ("family of tangents"). This lets us replace $f(\cdot)$ by a *supremum* over linear forms.

ESTIMATING f -DIVERGENCES

DUAL VARIATIONAL FORM

$$\begin{aligned}\mathcal{D}_f(P\|\widehat{P}) &= \mathbb{E}_{\widehat{P}}\left[f\left(\frac{p(x)}{\widehat{p}(x)}\right)\right] \\ &= \mathbb{E}_{\widehat{P}}\left[\sup_{t \in \mathbb{R}} \left\{t \frac{p(x)}{\widehat{p}(x)} - f^*(t)\right\}\right] \\ &\geq \sup_{T \in \mathcal{T}} \mathbb{E}_{\widehat{P}}\left[T(x) \frac{p(x)}{\widehat{p}(x)} - f^*(T(x))\right] \\ &= \sup_{T \in \mathcal{T}} \left\{ \underbrace{\mathbb{E}_{\widehat{P}}\left[T(x) \frac{p(x)}{\widehat{p}(x)}\right]}_{= \mathbb{E}_P[T(x)]} - \mathbb{E}_{\widehat{P}}[f^*(T(x))] \right\} \\ &= \sup_{T \in \mathcal{T}} \left(\mathbb{E}_P[T(x)] - \mathbb{E}_{\widehat{P}}[f^*(T(x))] \right).\end{aligned}$$

Under mild conditions, one can interchange sup and expectation to obtain equality; see Nguyen et al. [20], Nowozin et al. [21] for details.

ESTIMATING f -DIVERGENCES

DUAL VARIATIONAL FORM

Theorem 1.2 (Dual variational form of an f -divergence- Nguyen et al. [20])

Let $P, \hat{P} \in \mathcal{P}(\mathcal{X})$ two distributions such that P is absolutely continuous with respect to $\hat{P}$ and f a suitable generator function. The f -divergence between P and $\hat{P}$ admits a dual variational form:

$$\mathcal{D}_f(P \parallel \hat{P}) = \sup_{T \in \mathcal{T}} \left(\mathbb{E}_{\mathbf{x} \sim P} [T(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}} [f^*(T(\mathbf{x}))] \right). \quad (6)$$

We use $T^{\text{opt}} \in \mathcal{T}$ to denote the function that achieves the supremum.

ESTIMATING f -DIVERGENCES

DUAL VARIATIONAL FORM

Theorem 1.2 (Dual variational form of an f -divergence- Nguyen et al. [20])

Let $P, \hat{P} \in \mathcal{P}(\mathcal{X})$ two distributions such that P is absolutely continuous with respect to $\hat{P}$ and f a suitable generator function. The f -divergence between P and $\hat{P}$ admits a dual variational form:

$$\mathcal{D}_f(P||\hat{P}) = \sup_{T \in \mathcal{T}} \left(\mathbb{E}_{\mathbf{x} \sim P} [T(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}} [f^*(T(\mathbf{x}))] \right). \quad (6)$$

We use $T^{\text{opt}} \in \mathcal{T}$ to denote the function that achieves the supremum.

If $\mathcal{T}$ is rich enough to contain T^{opt} , then the dual form is exact. In practice, we parametrize T with a neural network and optimize over its parameters:

$$\mathcal{D}_f(P||\hat{P}) \approx \max_{\theta} \left(\mathbb{E}_{\mathbf{x} \sim P} [T_{\theta}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}} [f^*(T_{\theta}(\mathbf{x}))] \right).$$

FROM THE FIRST GAN TO f -GAN AND WGAN

MIN-MAX PROBLEM FOR f -GAN

Training objective: Minimizing $\mathcal{D}_f(P||\hat{P}_G)$:

$$\min_G \mathcal{D}_f(P||\hat{P}_G)$$

FROM THE FIRST GAN TO f -GAN AND WGAN

MIN-MAX PROBLEM FOR f -GAN

Training objective: Minimizing $\mathcal{D}_f(P \parallel \hat{P}_G)$:

$$\min_G \sup_{T \in \mathcal{T}} \left(\mathbb{E}_{\mathbf{x} \sim P} [T(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}_G} [f^*(T(\mathbf{x}))] \right)$$

FROM THE FIRST GAN TO f -GAN AND WGAN

MIN-MAX PROBLEM FOR f -GAN

Training objective: Minimizing $\mathcal{D}_f(P||\hat{P}_G)$:

$$\min_G \max_T \mathbb{E}_{\mathbf{x} \sim P} [T(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim Q} [f^*(T(G(\mathbf{z})))]$$

Interpretation: The inner maximization trains a critic T to estimate the f -divergence, while the outer minimization updates the generator G to reduce it.

FROM THE FIRST GAN TO f -GAN AND WGAN

MIN-MAX PROBLEM FOR f -GAN

Training objective: Minimizing $\mathcal{D}_f(P \parallel \hat{P}_G)$:

$$\min_G \max_T \mathbb{E}_{\mathbf{x} \sim P} [T(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim Q} [f^*(T(G(\mathbf{z})))]$$

Interpretation: The inner maximization trains a critic T to estimate the f -divergence, while the outer minimization updates the generator G to reduce it.

- If we set $f(u) = u \log u - (u + 1) \log(u + 1)$ (Jensen-Shannon), the corresponding conjugate is

$$f^*(t) = -\log(1 - e^t), \quad t < 0.$$

- We can define $T(x) = \log D(x)$, where $D(x) \in (0, 1)$ is a sigmoid output to ensure $T(x) < 0$.
- We get:

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim P} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim Q} [\log(1 - \exp(\log D(G(\mathbf{z}))))]$$

FROM THE FIRST GAN TO f -GAN AND WGAN

MIN-MAX PROBLEM FOR f -GAN

Training objective: Minimizing $\mathcal{D}_f(P \parallel \hat{P}_G)$:

$$\min_G \max_T \mathbb{E}_{\mathbf{x} \sim P} [T(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim Q} [f^*(T(G(\mathbf{z})))]$$

Interpretation: The inner maximization trains a critic T to estimate the f -divergence, while the outer minimization updates the generator G to reduce it.

- If we set $f(u) = u \log u - (u + 1) \log(u + 1)$ (Jensen-Shannon), the corresponding conjugate is

$$f^*(t) = -\log(1 - e^t), \quad t < 0.$$

- We can define $T(x) = \log D(x)$, where $D(x) \in (0, 1)$ is a sigmoid output to ensure $T(x) < 0$.
- We get:

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim P} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim Q} [\log(1 - D(G(\mathbf{z})))]$$

This recovers the original GAN objective from Goodfellow et al. [6]. **Note:** the original GAN minimizes the JS divergence between P and $\hat{P}_G$.

EXAMPLE OF f -GAN TRAINING

DIFFERENT f -DIVERGENCES ON MNIST

The paper of Nowozin et al. [21] demonstrates training f -GANs with different f -divergences on MNIST. Here are samples generated after training with different divergences:

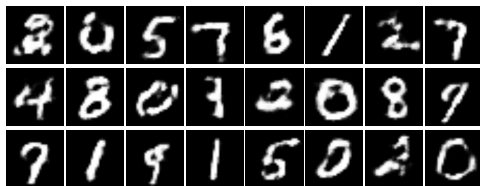


Figure. Jensen-Shannon samples



Figure. Hellinger samples



Figure. Kullback-Leibler samples



Figure. Reverse Kullback-Leibler samples

LIMITATIONS OF f -DIVERGENCES

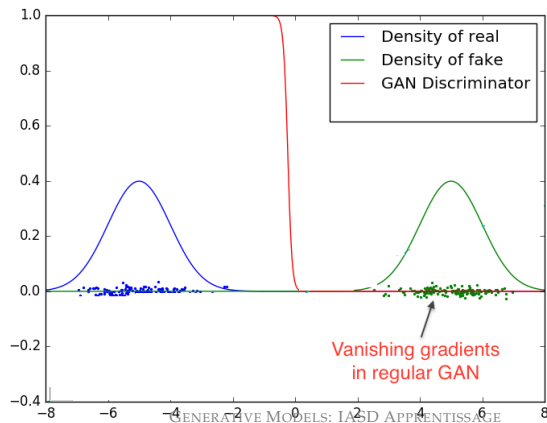
SUPPORT MISMATCH, VANISHING GRADIENTS, COLLAPSE

Vanishing gradients with JS (disjoint supports)

$$D^*(x) = \frac{p(x)}{p(x) + \hat{p}(x)} \text{ if } \text{supp}(P) \cap \text{supp}(\hat{P}) = \emptyset \quad \Rightarrow D^*(x) = \begin{cases} 1, & x \in \text{supp}(P) \\ 0, & x \in \text{supp}(\hat{P}) \end{cases}$$

For the generator loss:

$$\mathcal{L}_G = \mathbb{E}_{z \sim Q} [\log(1 - D(G(z)))] \quad \Rightarrow \quad \nabla_{\theta} \mathcal{L}_G = 0 \quad (\text{since } D^*(G(z)) = 0).$$



WASSERSTEIN DISTANCE (EARTH MOVER)

PRIMAL DEFINITION & INTUITION

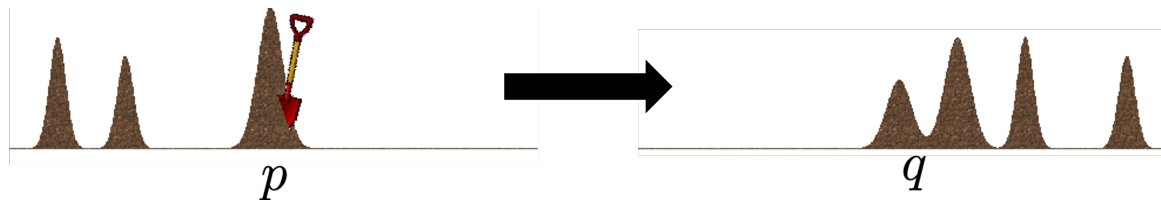
Wasserstein-1 distance: The Wasserstein-1 (or Earth Mover) distance between two distributions P and $\hat{P}$ is defined as

$$W_1(P, \hat{P}) = \inf_{\gamma \in \Pi(P, \hat{P})} \int_{\mathcal{X} \times \mathcal{X}} d(x, y) \, d\gamma(x, y),$$

where $\Pi(P, \hat{P})$ is the set of *couplings* (transport plans) having marginals P and $\hat{P}$.

Intuition:

- ▶ Move “earth” (probability mass) from P to $\hat{P}$ at cost = mass $\times$ distance.
- ▶ Sensitive to the *geometry* of $\mathcal{X}$; small shifts $\Rightarrow$ small W_1 .
- ▶ Well-defined and informative even when supports are disjoint (unlike JS near $\log 2$).



Earth Mover's intuition

WHY WASSERSTEIN HELPS: DIRAC EXAMPLE

JS vs W_1 BEHAVIOR

If $P = \delta_0$ and $\hat{P} = \delta_\theta$ (Dirac masses at 0 and θ):

- ▶ $W_1(P, \hat{P}) = |\theta|$ (distance between points)
- ▶ $JS(P || \hat{P}) = \begin{cases} \log 2, & \theta \neq 0 \\ 0, & \theta = 0 \end{cases}$

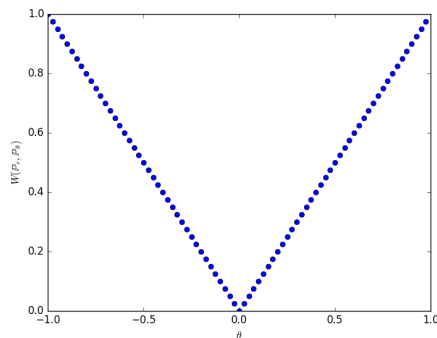


Figure. Wasserstein distance vs θ

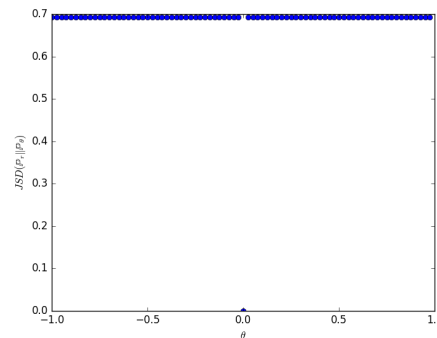


Figure. JS divergence vs θ

Both images are from Arjovsky et al. [2].

WHY WASSERSTEIN HELPS: GRADIENTS

MEANINGFUL UPDATES WITH DISJOINT SUPPORTS

What happens to the gradients when the supports of P and $\hat{P}$ are disjoint?

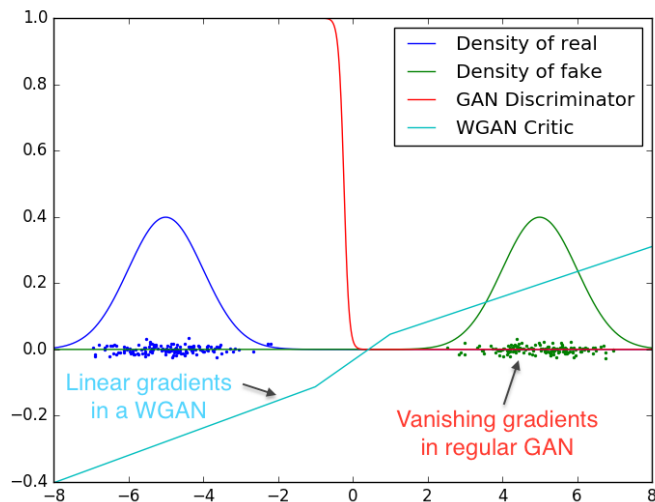


Figure. Generator gradients under JS vs WGAN when supports are disjoint. Source: Arjovsky et al. [2].

WGAN OBJECTIVE & 1-LIPSCHITZ CRITIC

KR DUALITY, CLIPPING VS GRADIENT PENALTY

Similar to f -divergences, the Wasserstein distance admits a dual form via the Kantorovich-Rubinstein duality:

$$W_1(P, \hat{P}) = \sup_{\|T\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim P}[T(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}}[T(\mathbf{x})],$$

where the supremum is over all 1-Lipschitz functions $T : \mathcal{X} \rightarrow \mathbb{R}$.

WGAN OBJECTIVE & 1-LIPSCHITZ CRITIC

KR DUALITY, CLIPPING VS GRADIENT PENALTY

Similar to f -divergences, the Wasserstein distance admits a dual form via the Kantorovich-Rubinstein duality:

$$W_1(P, \hat{P}) = \sup_{\|T\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim P}[T(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}}[T(\mathbf{x})],$$

where the supremum is over all 1-Lipschitz functions $T : \mathcal{X} \rightarrow \mathbb{R}$. The only change from f -divergence-GANs is the Lipschitz constraint on T . The WGAN training objective becomes:

$$\min_G \max_{\|T\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim P}[T(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim Q}[T(G(\mathbf{z}))].$$

WGAN OBJECTIVE & 1-LIPSCHITZ CRITIC

KR DUALITY, CLIPPING VS GRADIENT PENALTY

Similar to f -divergences, the Wasserstein distance admits a dual form via the Kantorovich-Rubinstein duality:

$$W_1(P, \hat{P}) = \sup_{\|T\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim P}[T(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \hat{P}}[T(\mathbf{x})],$$

where the supremum is over all 1-Lipschitz functions $T : \mathcal{X} \rightarrow \mathbb{R}$. The only change from f -divergence-GANs is the Lipschitz constraint on T . The WGAN training objective becomes:

$$\min_G \max_{\|T\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim P}[T(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim Q}[T(G(\mathbf{z}))].$$

Enforcing the Lipschitz constraint: original WGAN uses weight clipping; WGAN-GP [7] uses a gradient penalty regularizer:

$$\lambda \mathbb{E}[(\|\nabla_{\hat{\mathbf{x}}} T(\hat{\mathbf{x}})\|_2 - 1)^2],$$

where $\hat{\mathbf{x}}$ are points sampled uniformly along straight lines between real and generated samples.

MODERN GAN LOSSES & PRACTICE

HINGE LOSS, SPECTRAL NORM, TAKEAWAYS

WGAN and f -divergence-GANs form the basis of modern GAN training. WGAN has opened the door to more stable training and better sample quality by controlling the discriminator's Lipschitz constant.

Next major milestones:

- ▶ **Hinge loss:** used in SNGAN [19], improves stability and sample quality:

$$\begin{aligned}\mathcal{L}_D &= \mathbb{E}_{\mathbf{x} \sim P}[\max(0, 1 - D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim Q}[\max(0, 1 + D(G(\mathbf{z})))] \\ \mathcal{L}_G &= -\mathbb{E}_{\mathbf{z} \sim Q}[D(G(\mathbf{z}))]\end{aligned}$$

- ▶ **Spectral Normalization:** alternative to gradient penalty for enforcing Lipschitz constraint [19].
- ▶ Any many other architectural and training improvements covered in the next sections.

NEURAL ARCHITECTURES FOR GANs

WHY THEY MATTER & MILESTONES

Why architectures matter

- ▶ Stability of the adversarial game (gradient flow, Lipschitz control)
- ▶ Fidelity & diversity of samples (texture, long-range structure)
- ▶ Scalability to high resolution and many classes

Roadmap (milestones)

- ▶ **2014** — Early GANs (MLPs; small images)
- ▶ **2015** — *DCGAN*: conv/deconv stacks, BatchNorm, strided conv, (Leaky)ReLU
- ▶ **2017** — *Progressive Growing*: fade-in layers for high-res synthesis
- ▶ **2018** — *BigGAN*: class-conditional ResBlocks, self-attention, spectral norm
- ▶ **2018–2022** — *StyleGAN* → *StyleGAN2* → *StyleGAN-XL*: style modulation, noise injection, path-length reg

Cross-cutting ideas

- ▶ Normalization (BatchNorm in G/D; Spectral Norm in D)
- ▶ Conditioning (class embeddings, conditional BN)
- ▶ Architectural blocks (ResBlocks, attention) & up/downsampling choices

EARLY GAN ARCHITECTURES (MLP ERA)

TEMPLATE, LIMITATIONS, TAKEAWAY

Template (circa 2014)

- ▶ **Generator G (MLP):** $z \sim \mathcal{N}(0, I) \rightarrow$ dense layers $\rightarrow \tanh$ output (image range $[-1, 1]$)
- ▶ **Discriminator D (MLP):** $x \rightarrow$ dense layers $\rightarrow$ sigmoid probability $D(x) \in (0, 1)$
- ▶ Activations: ReLU/LeakyReLU in hidden layers; occasional Dropout; limited/no BN
- ▶ Target resolutions: small ($28 \times 28, 32 \times 32$)

Limitations observed

- ▶ Poor inductive bias for images (no convolutions $\Rightarrow$ weak locality/translation handling)
- ▶ Training instability & mode collapse at higher resolutions
- ▶ Limited long-range structure; textures often unrealistic

Takeaway Convolutional hierarchies (DCGAN) became the default to encode locality, share weights, and stabilize training.

DCGAN

CONVOLUTIONAL GUIDELINES & DESIGN CHOICES

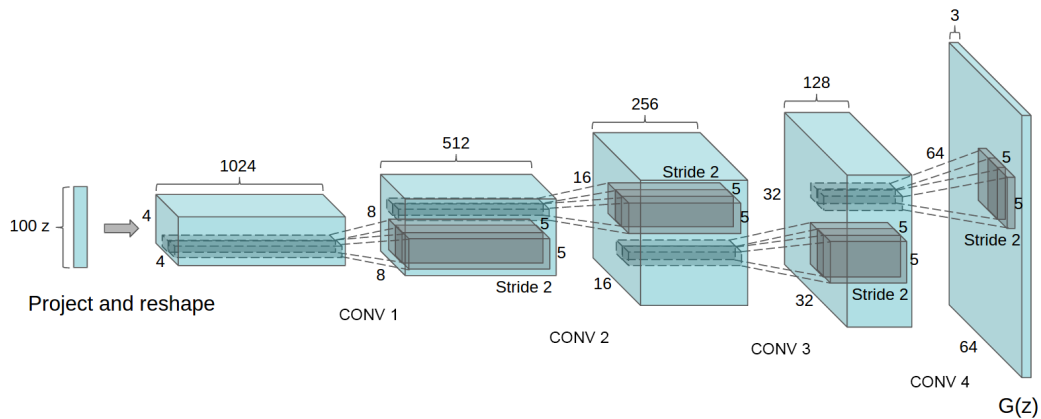
Architecture guidelines for stable Deep Convolutional GANs (DCGAN) [24]:

- ▶ **Replace pooling** with *strided convolutions* (discriminator) and *fractional-strided convolutions* (generator).
- ▶ **Use BatchNorm** in both the generator and the discriminator.
- ▶ **Remove fully connected hidden layers** for deeper architectures.
- ▶ **Use ReLU in the generator** for all layers **except** the output, which uses `tanh`.
- ▶ **Use LeakyReLU in the discriminator** for *all* layers.

Rationale: encourage stable gradients, better conditioning, and convolutional inductive biases (locality, translation equivariance).

DCGAN

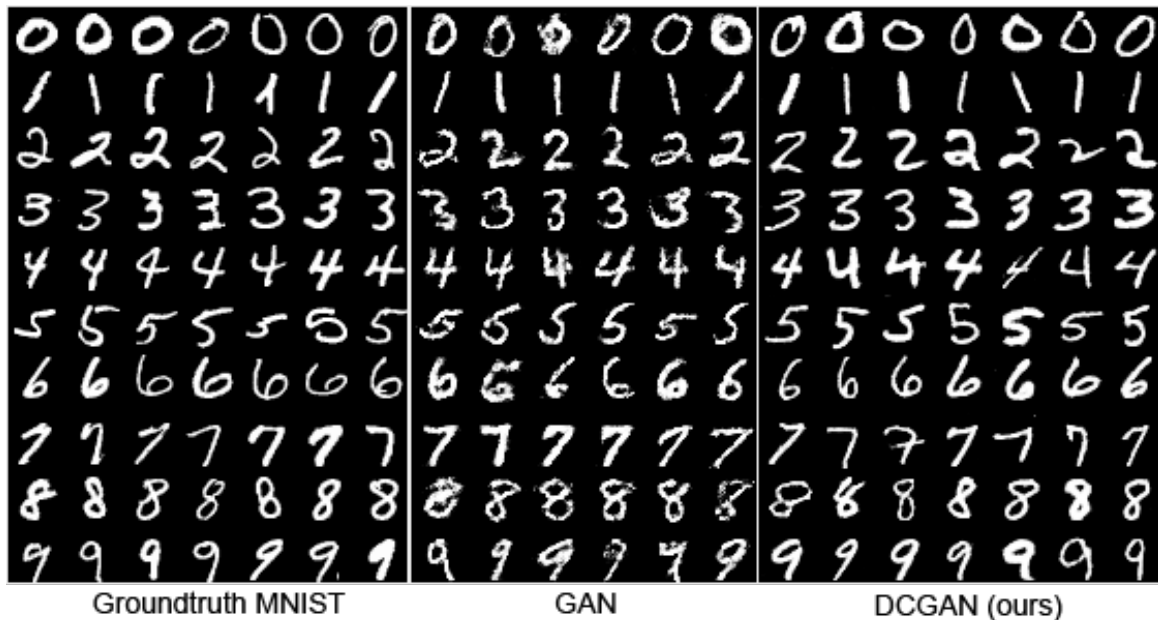
ARCHITECTURE DIAGRAM & RULES OF THUMB



DCGAN architecture (Radford et al., 2015/2016).

DCGAN

SAMPLE QUALITY: MNIST & FACES



MNIST samples from DCGAN

DCGAN

SAMPLE QUALITY: MNIST & FACES



Faces (CelebA) from DCGAN

DCGAN

SAMPLE QUALITY: MNIST & FACES



Faces (CelebA) from DCGAN

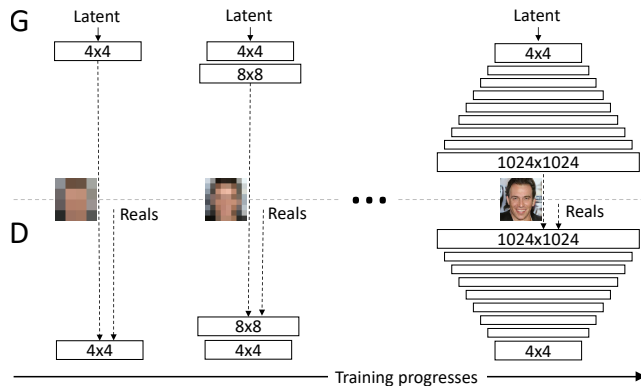
Note: quality improves markedly compared to early MLP GANs, but remains imperfect and at relatively low resolutions / latent dimensionalities. First decent generation in size 128x128

PROGRESSIVE GROWING OF GANs

FADE-IN LAYERS FOR HIGH-RESOLUTION SYNTHESIS

Key idea of Karras et al. [10]: progressively grow both the generator and discriminator during training.

- ▶ Train from low to high resolution (e.g., $4 \times 4 \rightarrow 8 \times 8 \rightarrow \dots \rightarrow 1024 \times 1024$).
- ▶ *Fade-in*: smoothly blend a newly added block over a transition period.
- ▶ Stabilizes training and improves high-res detail; reduces compute at start.
- ▶ Generator: upsampling + conv blocks; Discriminator: mirror downsampling blocks.

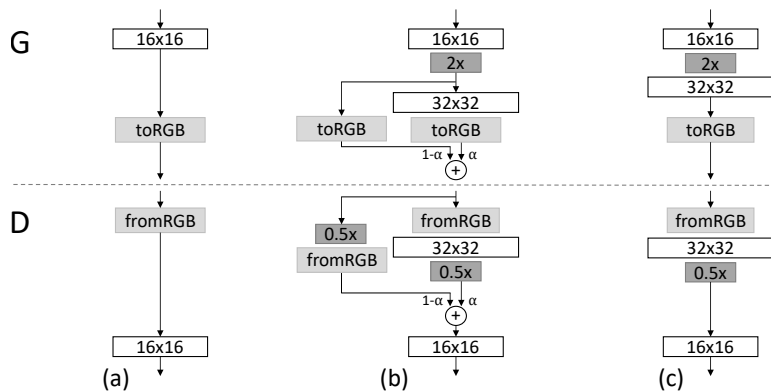


Progressive growing architecture

CONDITIONAL PROGRESSIVE MODELS

MULTI-SCALE SUPER-RESOLUTION CASCADE

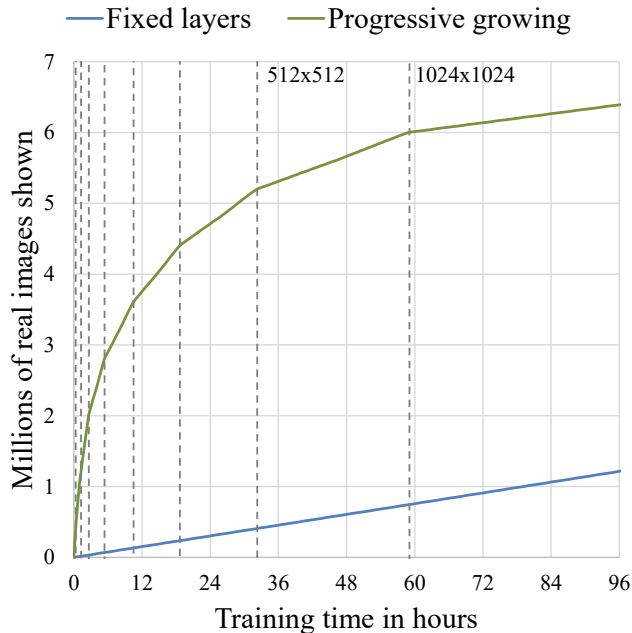
- ▶ Conditional variants can be built as cascades of super-resolution modules (e.g., $4 \rightarrow 8 \rightarrow 16 \rightarrow \dots$).
- ▶ Each stage upsamples and refines details conditioned on class/attributes.
- ▶ Useful for class-conditional synthesis and guided upscaling.



Architectures of conditional super-resolution models

PROGRESSIVE GROWING: TRAINING COST

TIME VS IMAGE SIZE



Training time as a function of image size

Note: early phases are cheap (small grids), heavy compute spent only at high resolutions.

PGGAN SAMPLES

CELEBA, LSUN

Note: PGGAN produces high-quality, high-resolution samples (1024x1024) with good global structure and fine details.



PGGAN SAMPLES

CELEBA, LSUN

LSUN dataset contains large-scale scene categories (bedrooms, churches, etc.) with diverse layouts and objects.

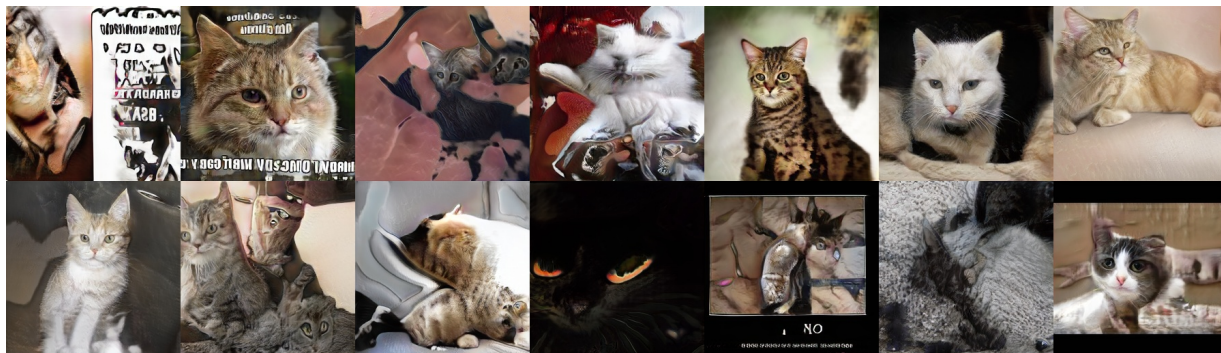


LSUN

PGGAN SAMPLES

CELEBA, LSUN

LSUN dataset contains large-scale scene categories (bedrooms, churches, etc.) with diverse layouts and objects.

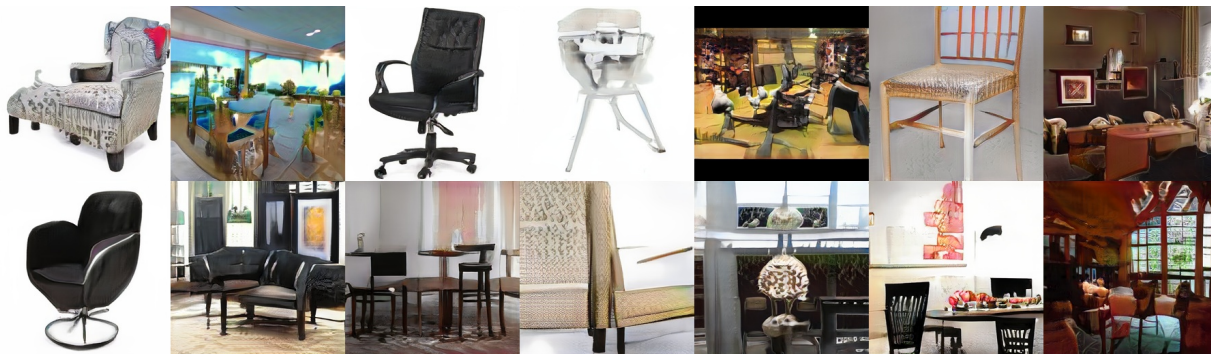


LSUN

PGGAN SAMPLES

CELEBA, LSUN

LSUN dataset contains large-scale scene categories (bedrooms, churches, etc.) with diverse layouts and objects.



LSUN

PGGAN SAMPLES

CELEBA, LSUN

LSUN dataset contains large-scale scene categories (bedrooms, churches, etc.) with diverse layouts and objects.

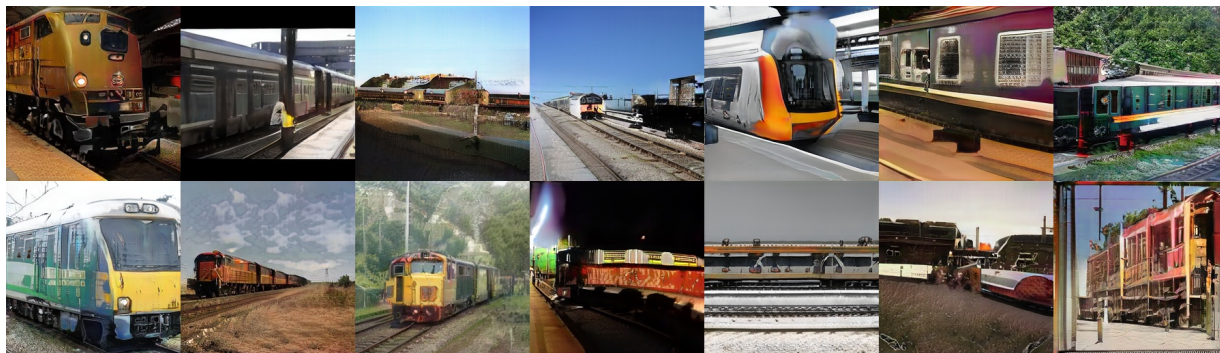


LSUN

PGGAN SAMPLES

CELEBA, LSUN

LSUN dataset contains large-scale scene categories (bedrooms, churches, etc.) with diverse layouts and objects.



LSUN

BIGGAN

CLASS-CONDITIONAL, ATTENTION, SPECTRAL NORM

Key improvements Brock et al. [4]:

- ▶ **Class-conditional architecture:** shared class embeddings + *conditional BatchNorm* in G ; deep residual blocks.
- ▶ **Long-range modeling:** non-local *self-attention* layers (SAGAN-style) to capture global structure.
- ▶ **Stability & fidelity:** *spectral normalization* in D and G , very large-batch training, and the *truncation trick* (trade-off between FID and diversity).



BigGAN: class-conditional + attention + SN

STYLEGAN & STYLEGAN-XL

STYLE MODULATION, NOISE, PATH LENGTH REG

Three milestones

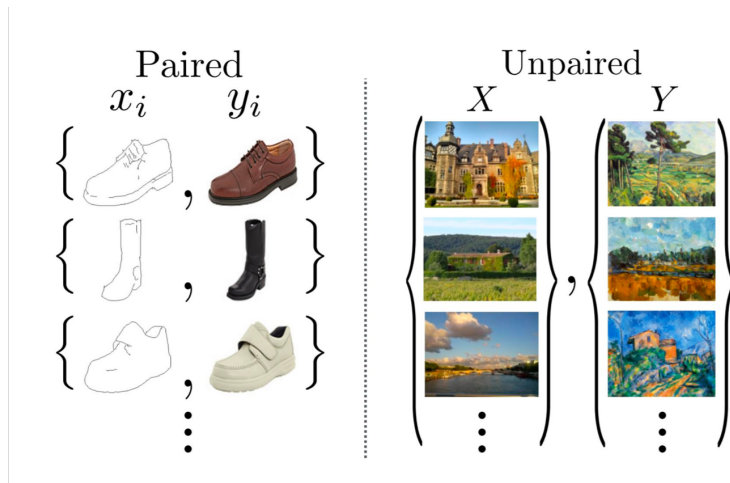
- ▶ **StyleGAN (2019):** style-based generator with mapping network; *style modulation* (AdaIN) per layer; per-pixel noise for stochastic detail; improved disentanglement. (Karras et al. [11])
- ▶ **StyleGAN2 (2020):** removes AdaIN artifacts via *weight demodulation*; *path length regularization*; architectural refinements for higher fidelity and fewer texture droplet artifacts. (Karras et al. [12])
- ▶ **StyleGAN-XL (2022):** scales StyleGAN to large, diverse datasets (e.g., ImageNet); stronger conditioning and training tricks; state-of-the-art FID/IS at higher resolutions. (Sauer et al. [25])

Example of realism: ThisPersonDoesNotExist.com (StyleGAN faces).

DOMAIN ADAPTATION WITH GANS

MOTIVATION & SETUP

- ▶ **Goal:** translate from a *source* domain X to a *target* domain Y *without* paired examples (unpaired domain adaptation).
- ▶ **Challenge:** many valid mappings $X \rightarrow Y$; need additional structure to avoid degenerate solutions.
- ▶ **Key idea:** use *consistency losses* (cycle/identity) to regularize adversarial training.



Source: Zhu et al., 2016

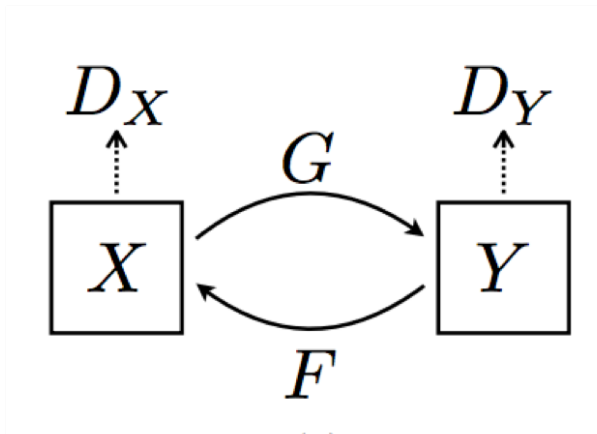
CYCLEGAN: MODEL

TWO GENERATORS, TWO DISCRIMINATORS

Let $G : X \rightarrow Y$ and $F : Y \rightarrow X$ with discriminators D_Y and D_X .

$$\mathcal{L}_{\text{GAN}}(G, D_Y) = \mathbb{E}_{y \sim p_Y}[\log D_Y(y)] + \mathbb{E}_{x \sim p_X}[\log(1 - D_Y(G(x)))],$$

$$\mathcal{L}_{\text{GAN}}(F, D_X) = \mathbb{E}_{x \sim p_X}[\log D_X(x)] + \mathbb{E}_{y \sim p_Y}[\log(1 - D_X(F(y)))].$$



Source: Zhu et al., 2016

CYCLEGAN: CONSISTENCY LOSSES

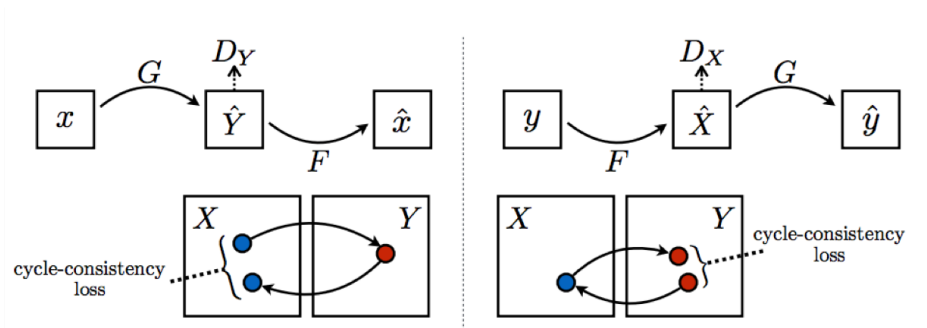
WHY UNPAIRED TRANSLATION WORKS

Cycle-consistency:

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_X} [\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_Y} [\|G(F(y)) - y\|_1].$$

Total objective:

$$\min_{G, F} \max_{D_X, D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y) + \mathcal{L}_{\text{GAN}}(F, D_X) + \lambda_{\text{cyc}} \mathcal{L}_{\text{cyc}}.$$

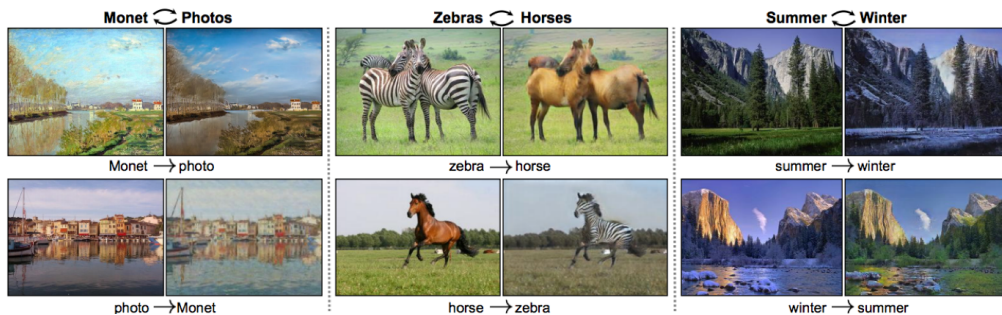


Source: Zhu et al., 2016

Consistency regularization

CYCLEGAN: QUALITATIVE RESULTS

MONET $\leftrightarrow$ PHOTO; HORSE $\leftrightarrow$ ZEBRA; SUMMER $\leftrightarrow$ WINTER



Unpaired image-to-image translation

CYCLEGAN: QUALITATIVE RESULTS

PHOTOGRAPHY $\leftrightarrow$ MONET, CEZANNE, VAN GOGH, UKIYO-E



Source: Zhu et al., 2016

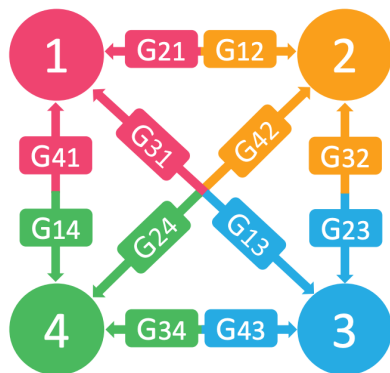
Unpaired image-to-image translation

STARGAN: ONE MODEL, MANY DOMAINS

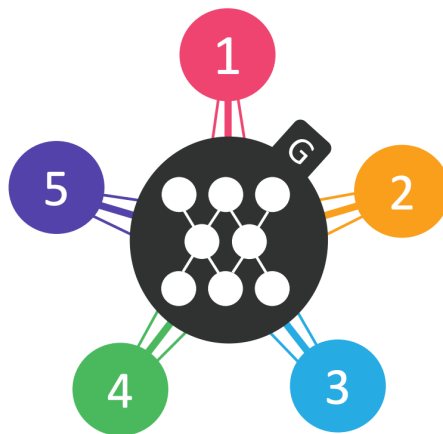
MOTIVATION & IDEA

- ▶ Multi-domain translation with a *single* generator $G(x, c)$ conditioned on target domain/attribute c .
- ▶ One discriminator D predicts *source* (real/fake) and *domain class*.
- ▶ Avoids $\mathcal{O}(K^2)$ models for K domains; parameters are shared.

(a) Cross-domain models



(b) StarGAN



Cross-domain generator

GENERATIVE MODELS: IASD APPRENTISSAGE

STARGAN: ARCHITECTURE & LOSSES

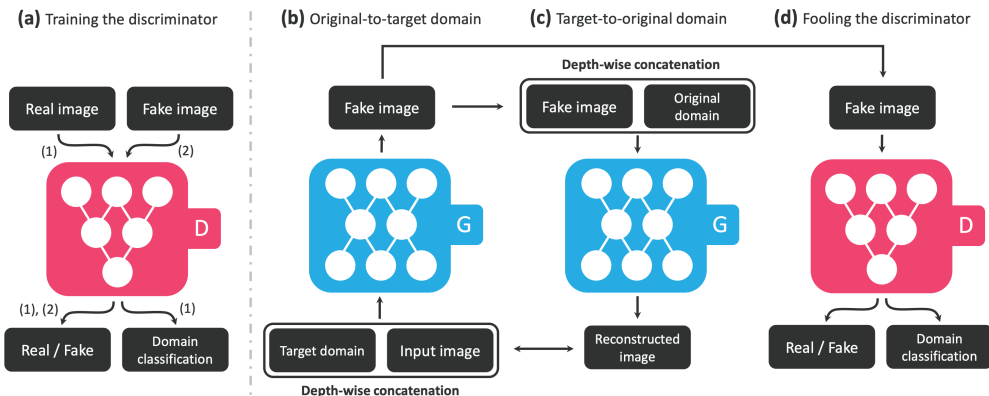
ADVERSARIAL, DOMAIN CLASSIFICATION, RECONSTRUCTION

Losses:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_x[\log D_{\text{src}}(x)] + \mathbb{E}_{x,c}[\log(1 - D_{\text{src}}(G(x, c)))],$$

$$\mathcal{L}_{\text{cls}}^{\text{real}} = \mathbb{E}_{x,c}[-\log D_{\text{cls}}(c | x)], \quad \mathcal{L}_{\text{cls}}^{\text{fake}} = \mathbb{E}_{x,c}[-\log D_{\text{cls}}(c | G(x, c))],$$

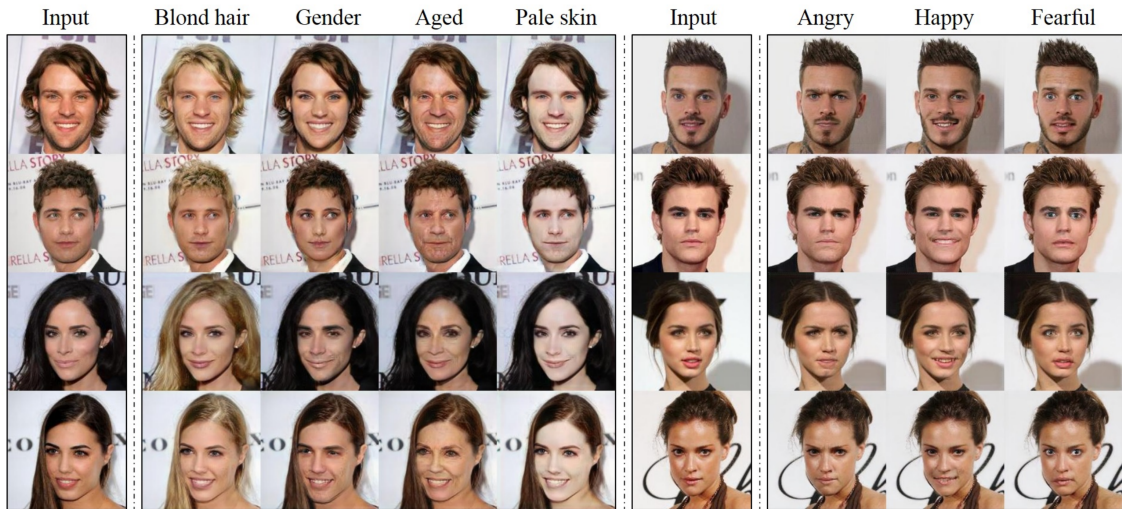
$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{x,c,c'}[\|x - G(G(x, c), c')\|_1], \quad c' = \text{original domain/attributes}.$$



StarGAN training overview

STARGAN: ATTRIBUTE TRANSFER ON CELEBA

RESULTS



Many-to-many translation with one model

DOMAIN ADAPTATION WITH CONSISTENCY: TAKEAWAYS

WHEN TO USE WHICH

- ▶ **CycleGAN** excels for *two* domains with strong cycle structure; identity loss helps preserve color/lighting.
- ▶ **StarGAN** scales to *many* domains/attributes via conditioning; add classification loss for controllability.
- ▶ Consistency losses provide structure in unpaired settings, stabilizing training and reducing mode drift.

YOUR TURN !

PRACTICAL SESSION ON GAN

Find the practical session on GANs in the link below:

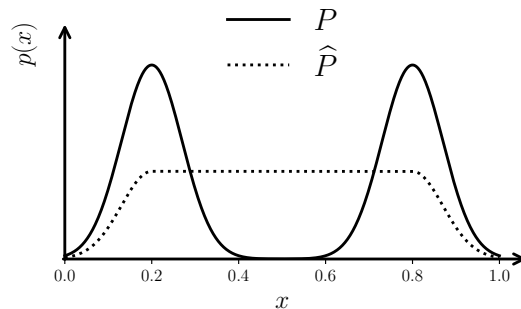
<https://www.di.ens.fr/~cbourdrez/genai.html>

SAMPLING FROM A GENERATIVE MODEL

GENERAL SETTING

To sample a point from the learned distribution $\hat{P}$:

- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.



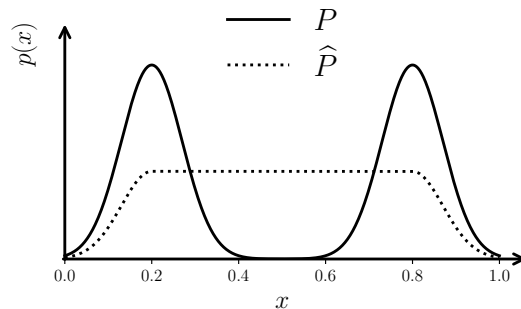
SAMPLING FROM A GENERATIVE MODEL

GENERAL SETTING

To sample a point from the learned distribution $\hat{P}$:

- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.

$$P \neq \hat{P}$$



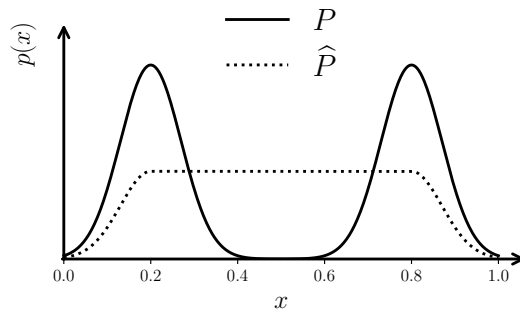
SAMPLING FROM A GENERATIVE MODEL

GENERAL SETTING

To sample a point from the learned distribution $\hat{P}$:

- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.

We have an estimation of $\frac{p(x)}{\hat{p}(x)}$ using $D(x) = \frac{p(x)}{p(x) + \hat{p}(x)}$.



SAMPLING FROM A GENERATIVE MODEL

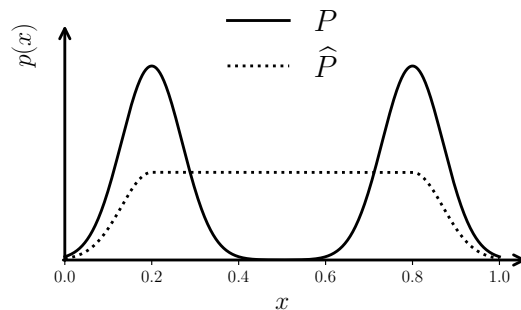
GENERAL SETTING

To sample a point from the learned distribution $\hat{P}$:

- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.

We have an estimation of $\frac{p(x)}{\hat{p}(x)}$ using $D(x) = \frac{p(x)}{p(x) + \hat{p}(x)}$.

$$\frac{p(x)}{\hat{p}(x)} = \frac{D(x)}{1 - D(x)}$$



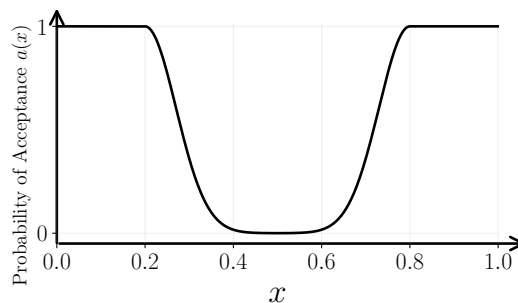
SAMPLING FROM A GENERATIVE MODEL

REJECTION SAMPLING

To sample a point from the refined distribution $\tilde{P}$:

- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.
- ▶ Accept x with probability $a(x)$.

Using $a(x) = \frac{p(x)}{\hat{p}(x)M}$ where $M \geq \sup_x \frac{p(x)}{\hat{p}(x)}$ in rejection sampling allows sampling from P .



SAMPLING FROM A GENERATIVE MODEL

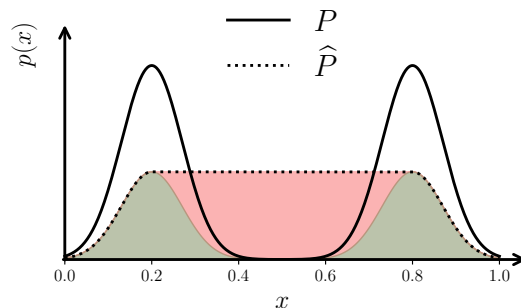
REJECTION SAMPLING

To sample a point from the refined distribution $\tilde{P}$:

- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.
- ▶ Accept x with probability $a(x)$.

The acceptance rate is :

$$\mathbb{E}_{\hat{P}}[a(x)].$$



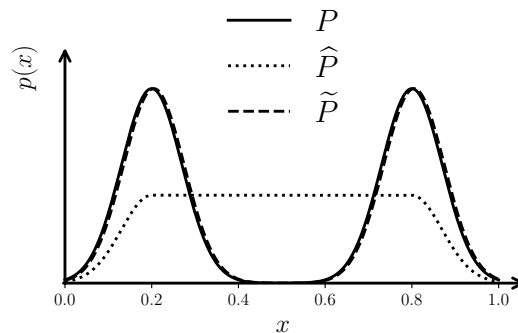
SAMPLING FROM A GENERATIVE MODEL

REJECTION SAMPLING

To sample a point from the refined distribution $\tilde{P}$:

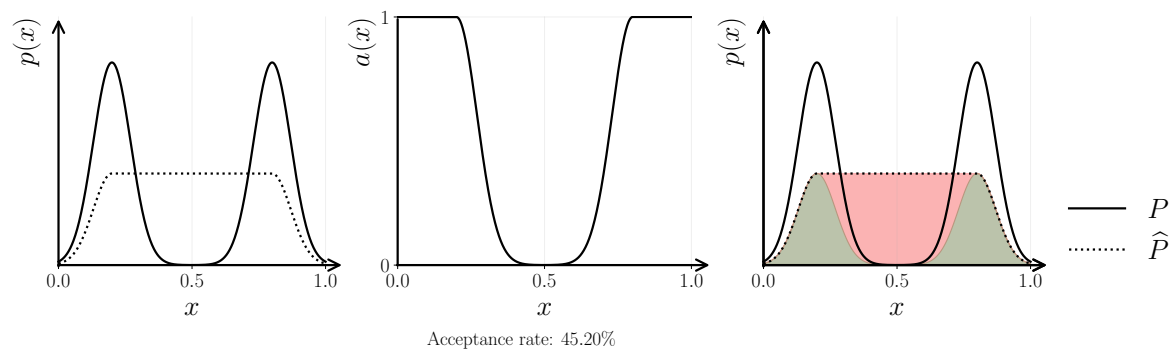
- ▶ Sample $z \sim Q$.
- ▶ Compute $x = G(z)$.
- ▶ Accept x with probability $a(x)$.

It defines a new distribution $\tilde{P}$.



SAMPLING FROM A GENERATIVE MODEL

REJECTION SAMPLING IN HIGH DIMENSION



Part III

DIFFUSION MODELS

FROM GANs TO DIFFUSION

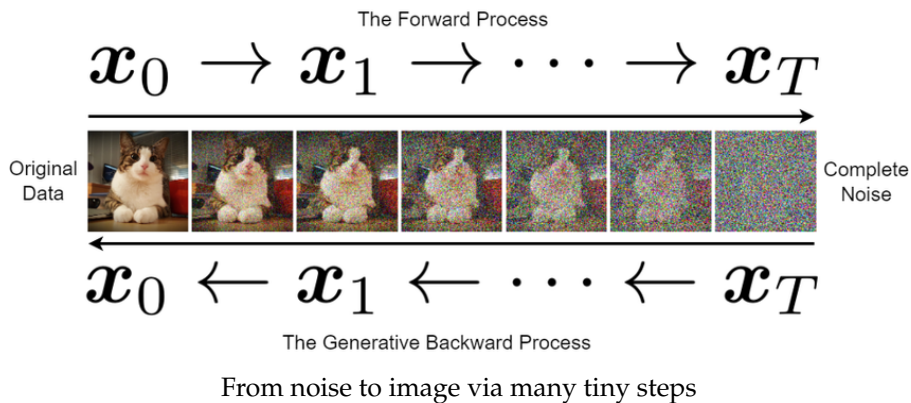
MOTIVATION & SHIFTS

- ▶ **Why move beyond GANs?** Adversarial training is unstable (mode collapse, delicate tuning) and hard to scale reliably.
- ▶ **New paradigm:** generate images by *iterative denoising* instead of a single jump. Each step makes a tiny, easy correction.
- ▶ **Architectures matter:** U-Nets (encoder–decoder with skip connections) became very strong backbones for denoising.
- ▶ **Landmarks:** DDPM ([9]), Score-based SDEs ([27]), modern EDM-style recipes ([13]).

DIFFUSION INTUITION

DECOMPOSE GENERATION INTO MANY SMALL STEPS

- ▶ Instead of one-shot synthesis, run a *multi-step denoising* chain.
- ▶ Start from pure noise and gradually remove it until a clean image emerges.
- ▶ Small steps are easier to learn than perfect generation in one go.



GAUSSIAN NOISING FAMILIES

VE vs VP; PREDICT x_0 vs PREVIOUS vs NOISE

- ▶ **Two popular forward processes:**
 - *VP (Variance Preserving)*: discrete chain with a variance schedule β_t (DDPM, [9]).
 - *VE (Variance Exploding)*: continuous SDE with increasing noise scale $\sigma(t)$ (Score SDE, [27]).
- ▶ **What to predict?** Either the clean image x_0 , the previous step x_{t-1} , or the noise ε . In practice, *noise prediction* is simple and robust.
- ▶ **Bottom line:** several choices work; pick the one that gives stable training and good gradients for your data.

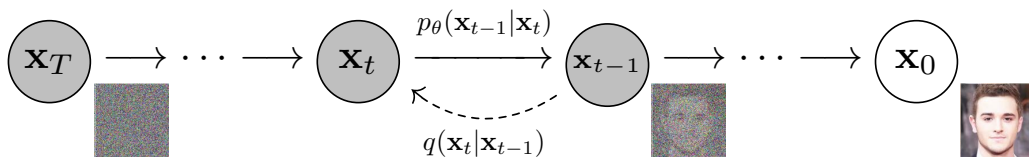
DENOISING DIFFUSION PROBABILISTIC MODELS (DDPM)

THE ORIGINAL DISCRETE-TIME APPROACH (HO ET AL., 2020)

- **Forward:** gradually add Gaussian noise step by step (Markov chain). **Gaussian transition kernel:**

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t \mathbf{I}), \quad \alpha_t = 1 - \beta_t.$$

- **Reverse:** learn a denoiser (U-Net) to remove a little noise at each step.
- **Training:** simple MSE on predicted noise; **Sampling:** ancestral steps from $t = T \rightarrow 1$.



DDPM overview

DDPM: CLOSED-FORM NOISING

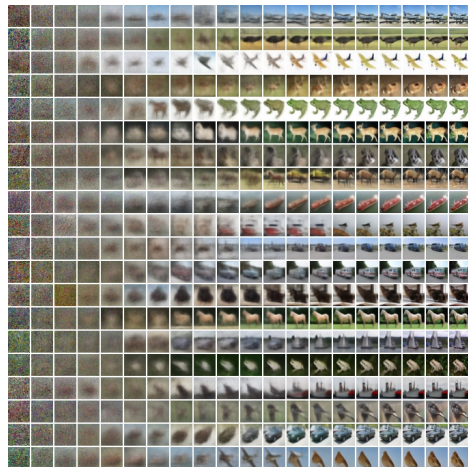
DIRECT SAMPLE OF x_t FROM x_0

One-shot formula (from repeated kernels):

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \mathbf{I}).$$

where $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ and $\alpha_s = 1 - \beta_s$.

- Shows the forward process remains **Gaussian** at all times.
- Useful to generate (x_t, t) training pairs efficiently.

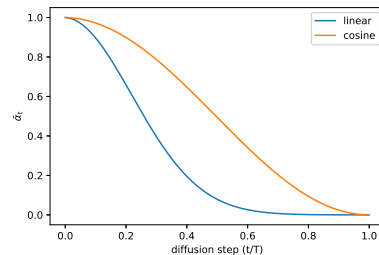
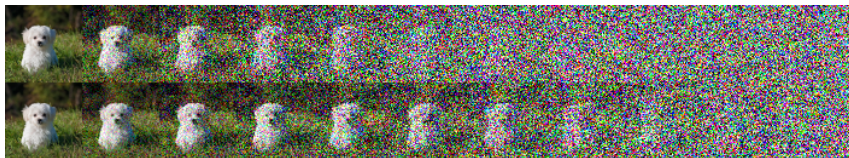


Noised images at various t

DDPM SCHEDULES

FROM IDDPM

- ▶ **Linear β_t (baseline):** original DDPM uses a linear variance schedule.
- ▶ **Cosine schedule on $\bar{\alpha}_t$:** smoother noise growth, better SNR at mid-times. $\bar{\alpha}_t \propto \cos^2\left(\frac{(t/T+s)\pi}{2}\right)$ (with offset s).



Scheduling shapes influence gradient quality and sample fidelity

DDPM: WHAT TO PREDICT?

MEAN/PREVIOUS STEP VS NOISE

- ▶ **Mean parameterization** (μ_θ): learn the mean of the reverse Gaussian $p_\theta(x_{t-1} | x_t) = \mathcal{N}(\mu_\theta(x_t, t), \sigma_t^2 \mathbf{I})$.
- ▶ **Clean-image parameterization** ($x_{0,\theta}$): predict the denoised image directly.
- ▶ **Noise parameterization** (ε_θ): predict the noise added at step t ; enables a simple, stable MSE loss.

Practical note: many systems adopt ε -prediction for robustness; all three are convertible into each other.

DDPM TRAINING

INPUTS, CONDITIONING, OUTPUTS

Inputs

- ▶ Noisy image x_t
- ▶ Timestep embedding t (positional/time encoding)

Outputs

- ▶ Typically $\varepsilon_\theta(x_t, t, y)$ (noise prediction)

Training signal

- ▶ Minimize MSE between target noise ε and $\varepsilon_\theta(x_t, t, y)$

$$\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{t, x_0, \varepsilon} \left[w(t) \left\| \varepsilon - \varepsilon_\theta(x_t, t) \right\|^2 \right]$$

$$\text{where } x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon.$$

Algorithm 1 Training

```
1: repeat
2:    $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ 
3:    $t \sim \text{Uniform}(\{1, \dots, T\})$ 
4:    $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5:   Take gradient descent step on
        $\nabla_\theta \left\| \varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t) \right\|^2$ 
6: until converged
```

DDPM training algorithm

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



DDPM denoising steps from $t = 11$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



Reconstructed $\hat{x}_0$ at each step from $t = 100$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



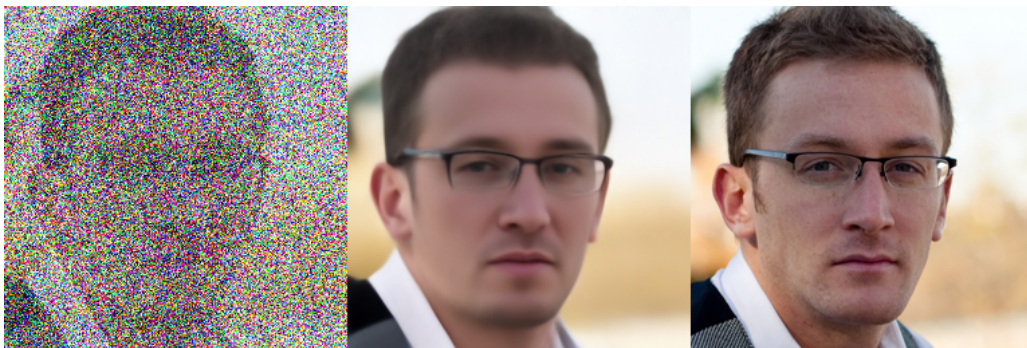
Reconstructed $\hat{x}_0$ at each step from $t = 200$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



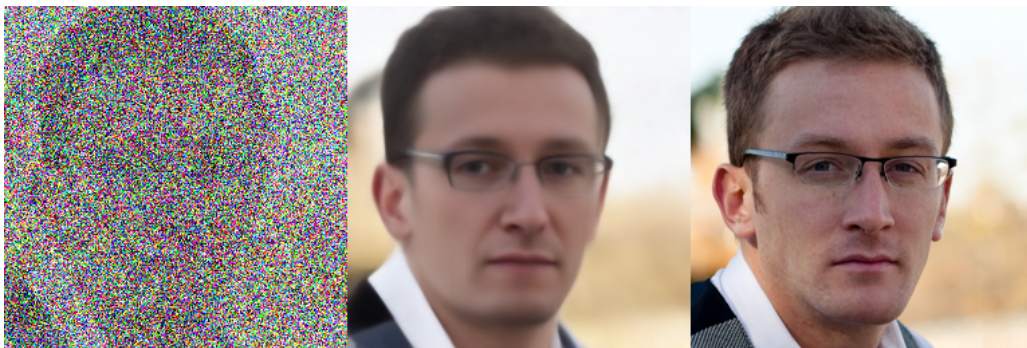
Reconstructed $\hat{x}_0$ at each step from $t = 400$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



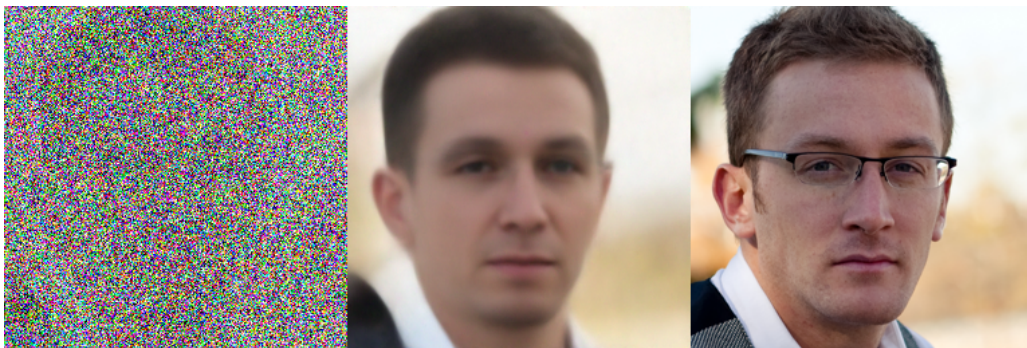
Reconstructed $\hat{x}_0$ at each step from $t = 500$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



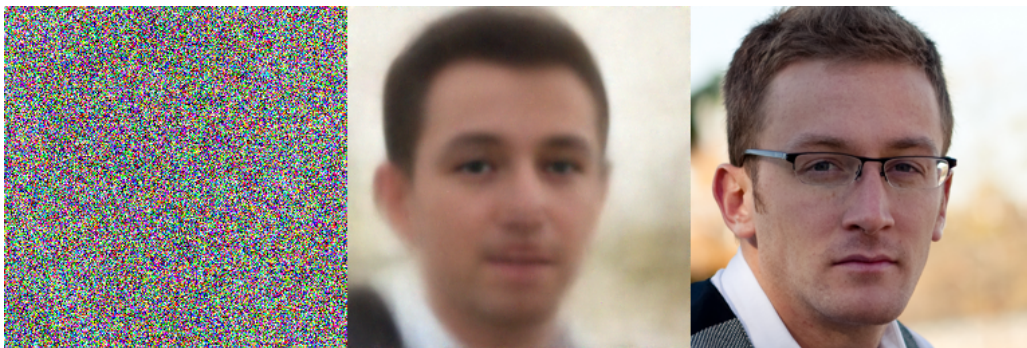
Reconstructed $\hat{x}_0$ at each step from $t = 600$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



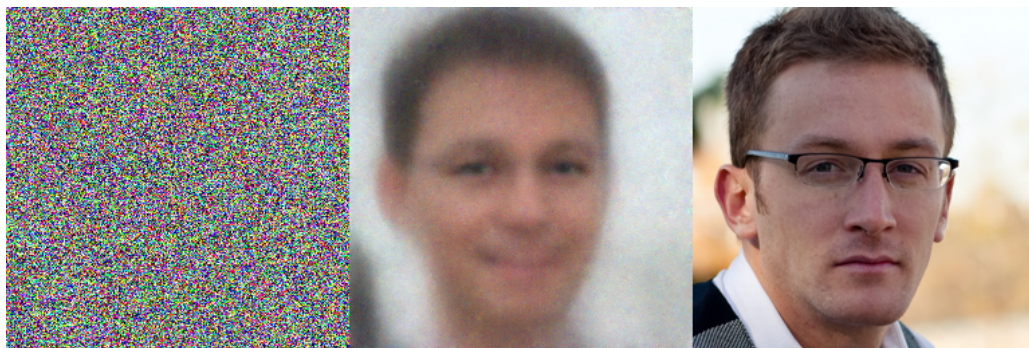
Reconstructed $\hat{x}_0$ at each step from $t = 700$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



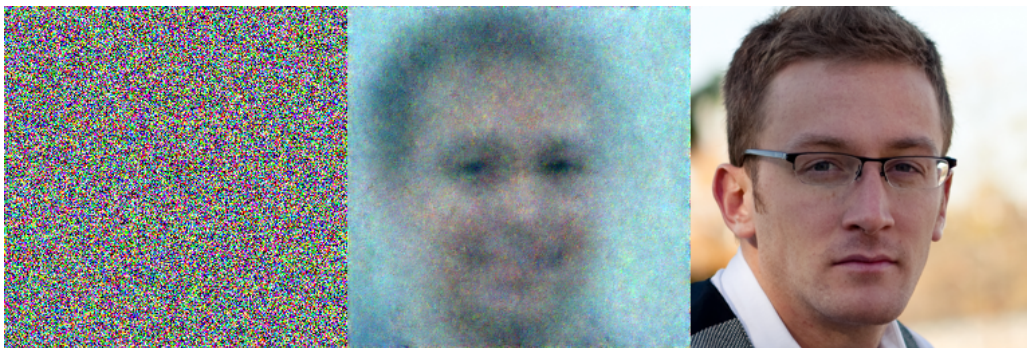
Reconstructed $\hat{x}_0$ at each step from $t = 800$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



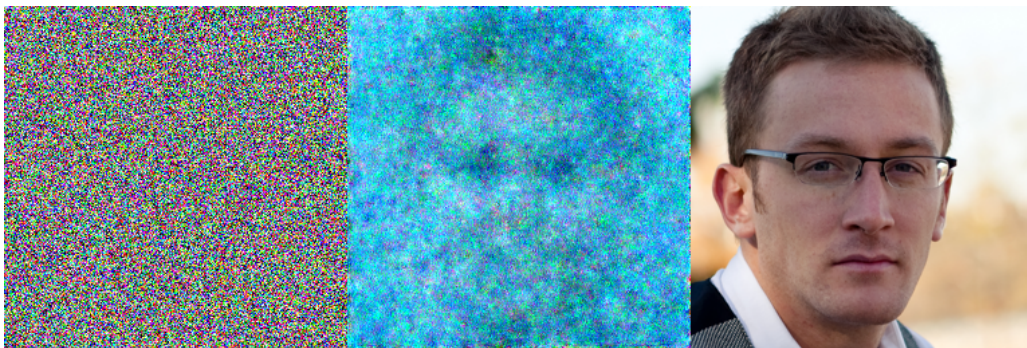
Reconstructed $\hat{x}_0$ at each step from $t = 900$ to $t = 0$

DDPM TRAINED MODEL

ILLUSTRATION

Using the reconstruction formula:

$$\hat{x}_0(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon_{\theta}(x_t, t) \right).$$



Reconstructed $\hat{x}_0$ at each step from $t = 1000$ to $t = 0$

DDPM: SAMPLING

STOCHASTIC REVERSE PROCESS

What should we compute to reverse the noising?

- ▶ The reverse conditional is **Gaussian**:
 $p_{\theta}(x_{t-1} | x_t) = \mathcal{N}(\mu_t(x_t), \sigma_t^2 \mathbf{I})$.
- ▶ To produce the next state x_{t-1} , we need the **mean** $\mu_t(x_t)$.

Mean from noise prediction

$$\mu_t(x_t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_{\theta}(x_t, t) \right)$$

$$x_{t-1} = \mu_t(x_t) + \sigma_t \mathbf{z},$$

where σ_t^2 is a variance schedule (often set to β_t) and $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$.

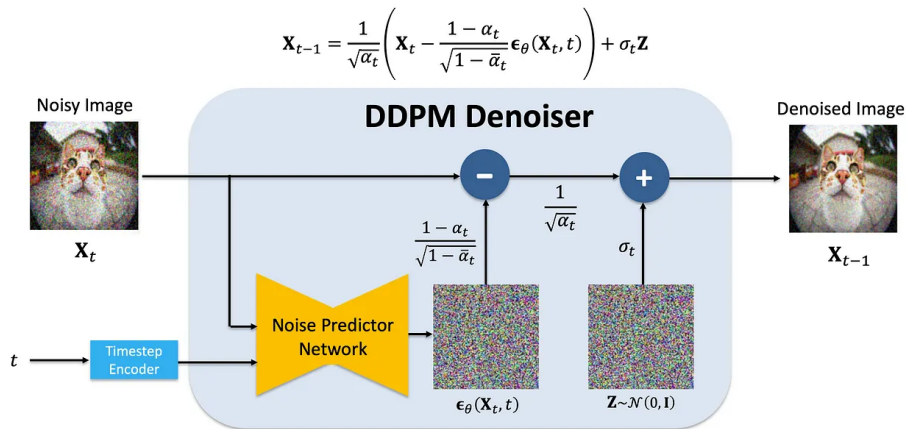
Algorithm 2 Sampling

```
1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2: for  $t = T, \dots, 1$  do
3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  if  $t > 1$ , else  $\mathbf{z} = \mathbf{0}$ 
4:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
5: end for
6: return  $\mathbf{x}_0$ 
```

Single reverse step

DDPM: SAMPLING

NET



Model I/O used by the sampler

DDPM: SAMPLING

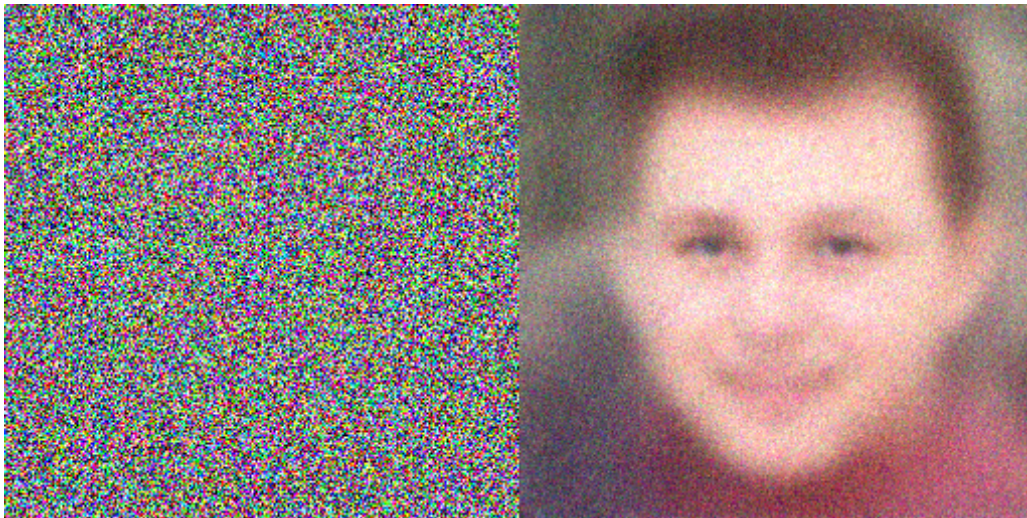
ILLUSTRATION



DDPM sampling steps from at $t = 1000$ to $t = 0$

DDPM: SAMPLING

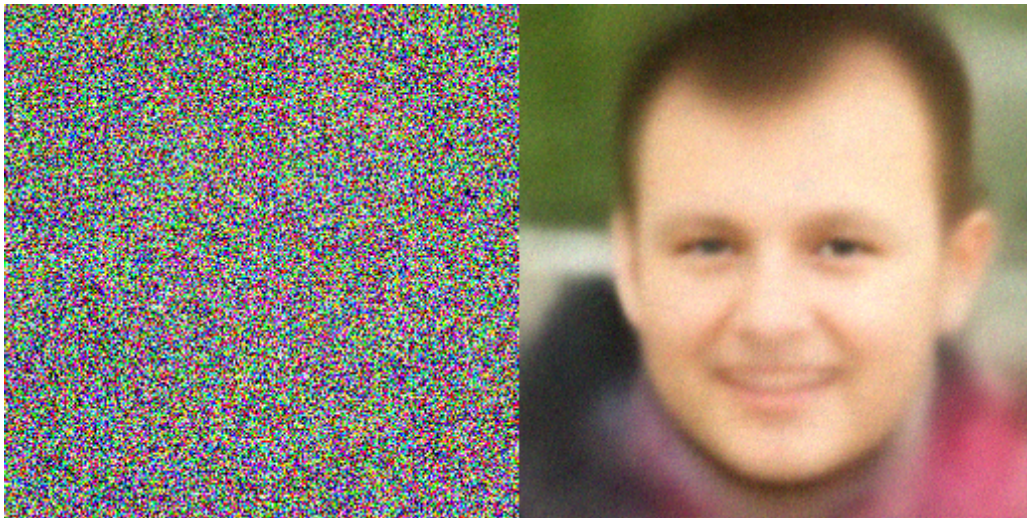
ILLUSTRATION



DDPM sampling steps from at $t = 900$ to $t = 0$

DDPM: SAMPLING

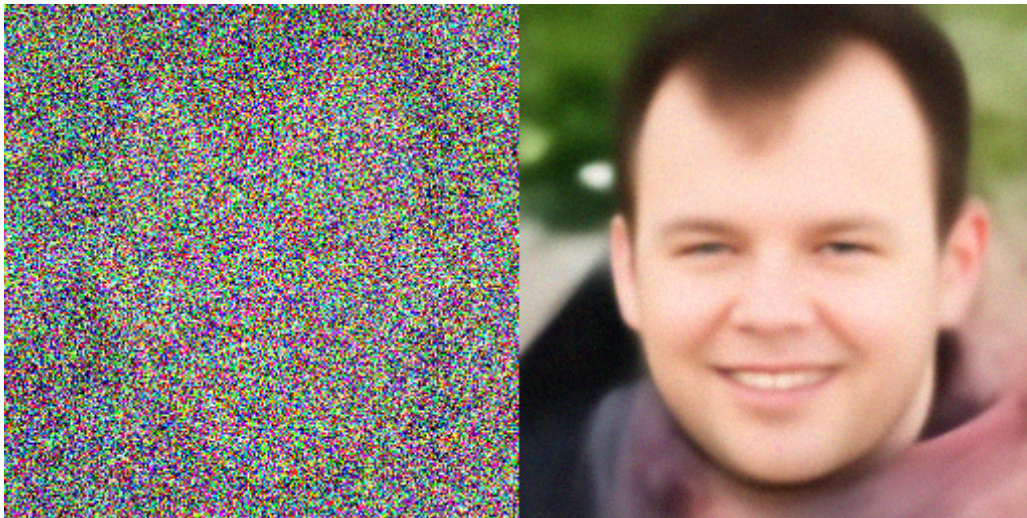
ILLUSTRATION



DDPM sampling steps from at $t = 800$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION



DDPM sampling steps from at $t = 700$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION



DDPM sampling steps from at $t = 600$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION



DDPM sampling steps from at $t = 500$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION



DDPM sampling steps from at $t = 400$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION



DDPM sampling steps from at $t = 300$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION



DDPM sampling steps from at $t = 200$ to $t = 0$

DDPM: SAMPLING

ILLUSTRATION

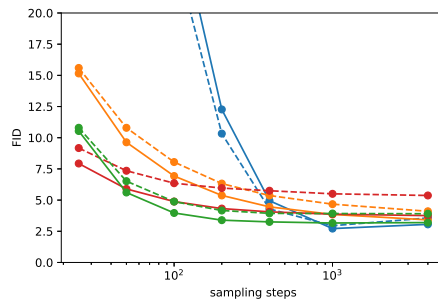


DDPM sampling steps from at $t = 100$ to $t = 0$

DDPM: RESULTS

EXAMPLE GENERATIONS

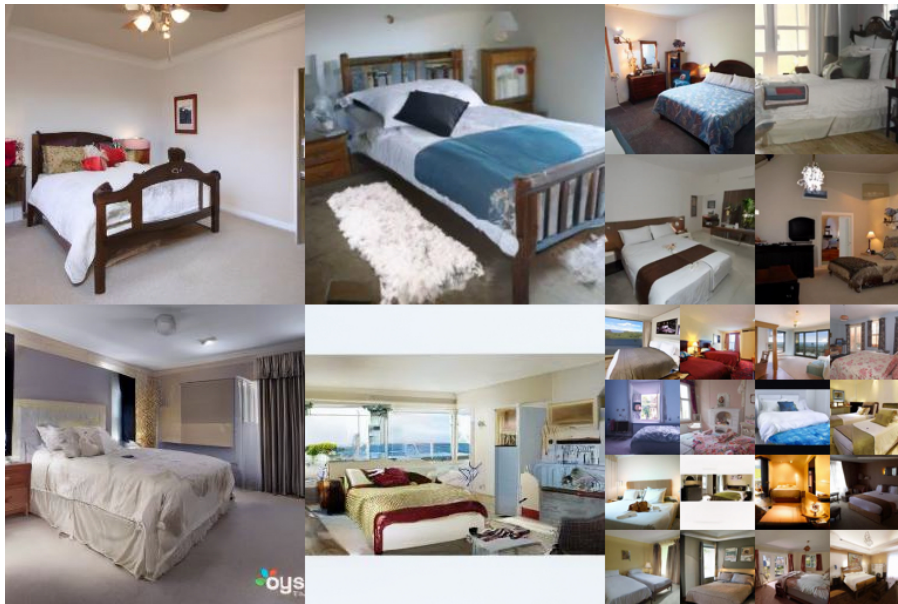
Better than GANs in diversity; high quality with enough steps



CIFAR-10 samples at 50, 100, 200, 400, 1000 steps

DDPM: RESULTS

EXAMPLE GENERATIONS



LSUN Bedroom samples (DDPM, 256 steps)

GENERALIZING TO SCORE-BASED MODELS

GAUSSIAN CORRUPTION, TWEEDIE & SCORE

Generic Gaussian corruption

$$x_t = s_t x_0 + \sigma_t \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \mathbf{I}), \quad s_t \in (0, 1], \quad \sigma_t > 0.$$

What can the network learn?

- ▶ Noise $\varepsilon_\theta(x_t, t)$
- ▶ Original $x_{0,\theta}(x_t, t)$

Tweedie formula (MMSE denoising)

$$\mathbb{E}[x_0 \mid x_t] = \frac{1}{\sigma_t} \left(x_t + \sigma_t^2 \underbrace{\nabla_x \log p_t(x)}_{\text{score}} \right) \Big|_{x=x_t}.$$

The **score** $\nabla_x \log p_t(x)$ directly steers the optimal denoiser.



Denoiser Input across t



Denoiser outputs across t

WHAT IS THE SCORE?

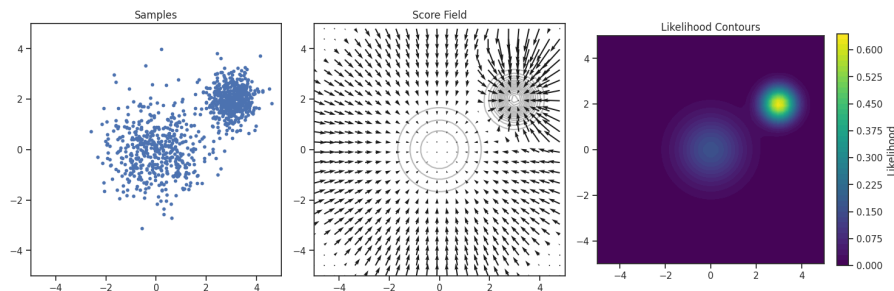
GEOMETRY AND INTUITION

Definition

$$s(x, t) = \nabla_x \log p_t(x).$$

Intuition

- ▶ Vector field pointing toward higher density regions.
- ▶ Local and scale-aware: magnitude relates to uncertainty (via σ_t).
- ▶ Small steps along $s(x, t)$ denoise and increase likelihood.



Score field visualization

FORWARD SDE FAMILIES

VE AND VP (SONG ET AL.)

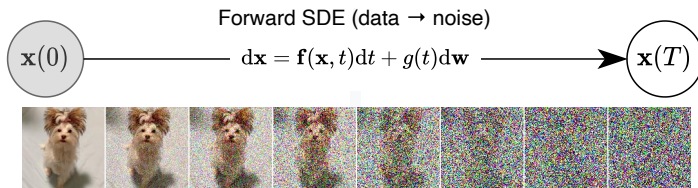
Generic forward SDE

$$dx = f(x, t) dt + g(t) dW_t.$$

Two popular choices

- ▶ *VE (Variance Exploding)*: $f=0$, $g(t)$ grows; noise variance increases over time.
- ▶ *VP (Variance Preserving)*: $f(x, t) = -\frac{1}{2}\beta(t)x$, $g(t) = \sqrt{\beta(t)}$; SNR controlled.

Effect Different schedules shape $p_t(x)$ and the training signal.



Forward diffusion as SDE

FORWARD SDE FAMILIES

VE AND VP (SONG ET AL.)

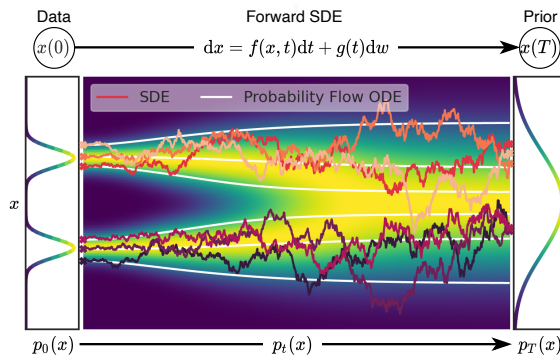
Generic forward SDE

$$dx = f(x, t) dt + g(t) dW_t.$$

Two popular choices

- ▶ VE (Variance Exploding): $f=0, g(t)$ grows; noise variance increases over time.
- ▶ VP (Variance Preserving): $f(x, t) = -\frac{1}{2}\beta(t)x, g(t) = \sqrt{\beta(t)}$; SNR controlled.

Effect Different schedules shape $p_t(x)$ and the training signal.



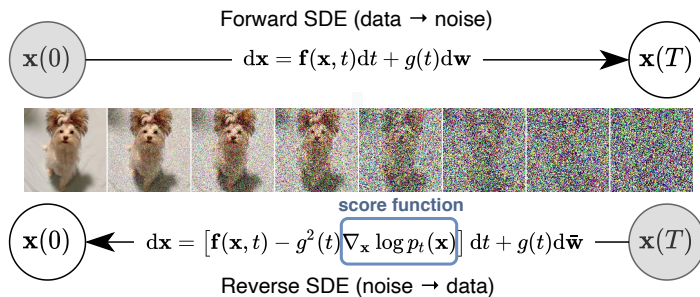
Forward diffusion as SDE

REVERSE-TIME DYNAMICS

REVERSE SDE DEPENDS ON THE SCORE

Concept

- ▶ To run time backward, we need the score $\nabla_x \log p_t(x)$.
- ▶ Train $s_\theta(x, t)$ to approximate the score at all noise levels.
- ▶ Reverse step uses s_θ to move $x_t \rightarrow x_{t-1}$ (stochastic).



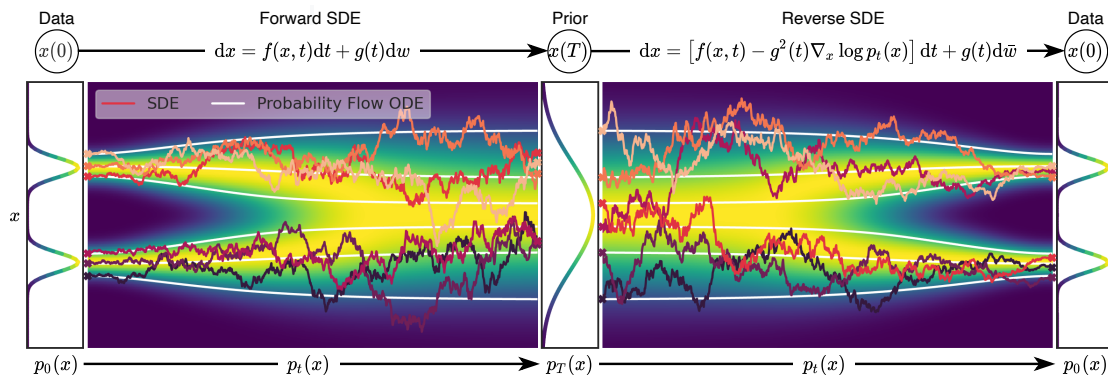
Forward diffusion as SDE

REVERSE-TIME DYNAMICS

REVERSE SDE DEPENDS ON THE SCORE

Concept

- ▶ To run time backward, we need the score $\nabla_x \log p_t(x)$.
- ▶ Train $s_\theta(x, t)$ to approximate the score at all noise levels.
- ▶ Reverse step uses s_θ to move $x_t \rightarrow x_{t-1}$ (stochastic).



Forward diffusion as SDE

TRAINING THE SCORE NETWORK

DENOISING SCORE MATCHING (VINCENT, 2011)

Objective

$$\mathcal{L}(\theta) = \mathbb{E}_{t, x_0, \varepsilon} \left[\|w(t) \| s_{\theta}(x_t, t) - \nabla_{x_t} \log p_t(x_t) \|_2^2 \right], \quad x_t = s_t x_0 + \sigma_t \varepsilon.$$

Vincent (2011): denoising score matching

- ▶ Replacing the intractable $\nabla_{x_t} \log p_t(x_t)$ by the *conditional score* is unbiased in expectation:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, x_0, \varepsilon} \left[\|w(t) \| s_{\theta}(x_t, t) - \nabla_{x_t} \log p_t(x_t | x_0) \|_2^2 \right], \quad x_t = s_t x_0 + \sigma_t \varepsilon.$$

- ▶ For Gaussian corruption: $p_t(x_t | x_0) = \mathcal{N}(x_t; s_t x_0, \sigma_t^2 \mathbf{I}) = \frac{1}{(2\pi\sigma_t^2)^{d/2}} \exp \left(-\frac{\|x_t - s_t x_0\|^2}{2\sigma_t^2} \right)$, so

$$\nabla_{x_t} \log p(x_t | x_0) = -\frac{x_t - s_t x_0}{\sigma_t^2}$$

Practical supervised target

$$\mathcal{L}(\theta) = \mathbb{E}_{t, x_0, \varepsilon} \left[\|w(t) \| s_{\theta}(x_t, t) + \frac{x_t - s_t x_0}{\sigma_t^2} \|_2^2 \right].$$

EQUIVALENCE TO DDPM PARAMETERIZATIONS

FROM SCORE $\leftrightarrow \varepsilon \leftrightarrow x_0$

From score to denoiser and noise

Using Tweedie: $\mathbb{E}[x_0 \mid x_t] = (x_t + \sigma_t^2 \nabla \log p_t(x_t)) / s_t$.

$$\hat{x}_0^{(\text{score})}(x_t, t) = \frac{x_t + \sigma_t^2 s_\theta(x_t, t)}{s_t},$$

$$\hat{\varepsilon}^{(\text{score})}(x_t, t) = \frac{x_t - s_t \hat{x}_0^{(\text{score})}}{\sigma_t} = -\sigma_t s_\theta(x_t, t).$$

ALGORITHMS (PSEUDOCODE)

SCORE TRAINING AND DDPM-STYLE DENOISING

Algorithm 7: Training (DSM)

Inputs: $\rho(t)$, $s(t)$, $\sigma(t)$, $w(t)$, η

for $step \leftarrow 1$ **to** N **do**

 Sample $t \sim \rho(t)$

 Sample $x_0 \sim p_{\text{data}}, \varepsilon \sim \mathcal{N}(0, \mathbf{I})$

$x_t \leftarrow s(t) x_0 + \sigma(t) \varepsilon$

$\text{target} \leftarrow -\frac{x_t - s(t) x_0}{\sigma(t)^2}$

 // conditional score

$\mathcal{L} \leftarrow w(t) \|s_\theta(x_t, t) - \text{target}\|_2^2$

$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$

end

Algorithm 8: Sampling (Score-based, ancestral step)

Inputs: noise schedule $\{(s_t, \sigma_t)\}_{t=0}^T$, score network $s_\theta(x, t)$

Sample $x_T \sim \mathcal{N}(0, \sigma_T^2 \mathbf{I})$

for $t \leftarrow T; t \geq 1; t \leftarrow t - 1$ **do**

$\hat{\varepsilon} \leftarrow -\sigma_t s_\theta(x_t, t)$ // from $\nabla_x \log p_t(x)$

$\hat{x}_0 \leftarrow \frac{x_t - \sigma_t \hat{\varepsilon}}{\sigma_t}$ // invert

$x_t = s_t x_0 + \sigma_t \varepsilon$

if $t > 1$ **then**

 Sample $z \sim \mathcal{N}(0, \mathbf{I})$

$x_{t-1} \leftarrow s_{t-1} \hat{x}_0 + \sigma_{t-1} z$ // re-noise
 at level $t-1$

else

$x_0 \leftarrow \hat{x}_0$

end

end

return x_0

SDE: RESULTS

SAMPLES



CelebA-HQ samples and interpolations (Score SDE, 1000 steps)

SDE: RESULTS

SAMPLES



CelebA-HQ samples (Score SDE, 1000 steps)

PROBABILITY-FLOW ODE

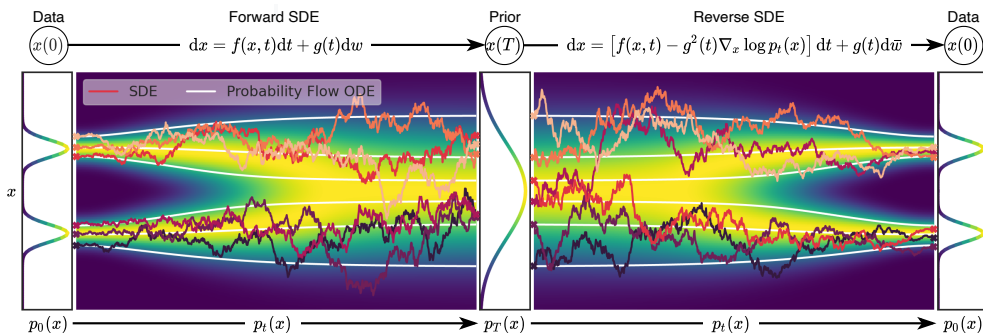
DETERMINISTIC COUNTERPART (SONG ET AL., 2021)

Forward SDE (general form)

$$dx = f(x, t) dt + g(t) dW_t.$$

Reverse-time SDE (stochastic sampler)

$$dx = \left(f(x, t) - g(t)^2 \nabla_x \log p_t(x) \right) dt + g(t) d\bar{W}_t.$$



Song et al. (2021): SDE vs probability-flow ODE

PROBABILITY-FLOW ODE

DETERMINISTIC COUNTERPART (SONG ET AL., 2021)

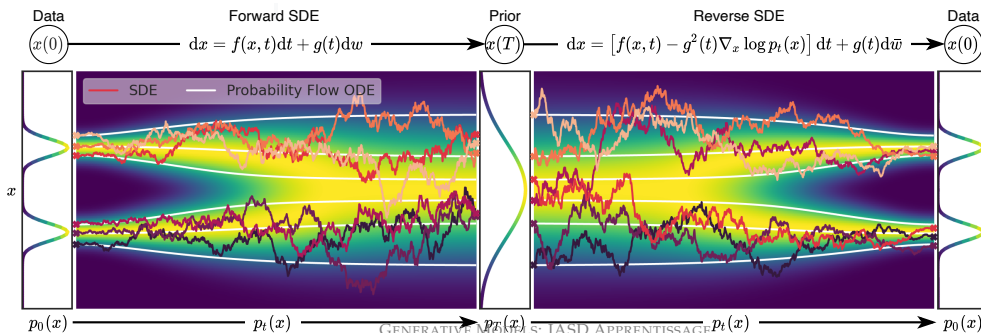
Forward SDE (general form)

$$dx = f(x, t) dt + g(t) dW_t.$$

Probability-flow ODE (deterministic sampler)

$$\frac{dx}{dt} = f(x, t) - \frac{1}{2} g(t)^2 \nabla_x \log p_t(x) \approx f(x, t) - \frac{1}{2} g(t)^2 s_\theta(x, t).$$

- ▶ Same marginals p_t as the SDE, but **no injected noise**.
- ▶ Can be solved with standard ODE solvers (Euler / Heun / RK).
- ▶ Useful for **fast sampling**.



SAMPLING WITH THE PROBABILITY-FLOW ODE

EULER / HEUN DISCRETIZATION

Velocity field

$$v_{\theta}(x, t) := f(x, t) - \frac{1}{2} g(t)^2 s_{\theta}(x, t) \Rightarrow \frac{dx}{dt} = v_{\theta}(x, t).$$

Algorithm 9: ODE Sampling (Probability-flow)

Inputs: time grid $T = t_K > \dots > t_0 = 0, f, g$, score network $s_{\theta}(x, t)$

```
Sample  $x_{t_K} \sim p_{t_K}$  // typically
 $\mathcal{N}(0, \sigma(T)^2 \mathbf{I})$  for  $k \leftarrow K; k \geq 1; k \leftarrow k - 1$  do
     $\Delta t \leftarrow t_{k-1} - t_k$  // negative step
    // Euler (1st order)
     $x_{t_{k-1}} \leftarrow x_{t_k} + \Delta t v_{\theta}(x_{t_k}, t_k)$ 
    // Heun (2nd order, optional)
    // uncomment for better quality at
    // low steps
     $\tilde{x} \leftarrow x_{t_k} + \Delta t v_{\theta}(x_{t_k}, t_k)$ 
     $x_{t_{k-1}} \leftarrow x_{t_k} + \frac{\Delta t}{2} \left( v_{\theta}(x_{t_k}, t_k) + v_{\theta}(\tilde{x}, t_{k-1}) \right)$ 
end
return  $x_{t_0}$ 
```

Notes

- Deterministic trajectory (no noise injected).
- High-order solvers reduce NFE for a given quality.
- Same learned score as the reverse SDE.

SDE vs ODE

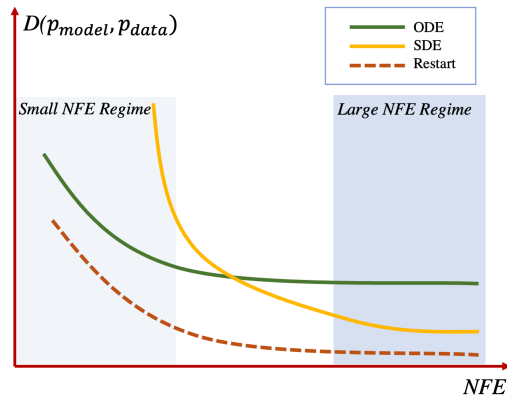
PROS AND CONS

Reverse-time SDE sampler

- ▶ **Stochastic:** injects Gaussian noise at each step.
- ▶ Often **better diversity / mode coverage** for a fixed small number of steps.
- ▶ More robust exploration of the data manifold.

Probability-flow ODE sampler

- ▶ **Deterministic:** fixed output given the initial noise seed.
- ▶ Enables **faster sampling** with high-order solvers (Heun / RK).
- ▶ Supports **exact likelihood evaluation** (continuous change-of-variables).
- ▶ May require slightly more careful discretization for best perceptual quality.



Sample quality vs number of steps

CLASSIFIER GUIDANCE

FROM UNCONDITIONAL TO CONDITIONAL SCORE

Goal: sample from the class-conditional density $p_t(x | y)$.

Key identity (Bayes)

$$\nabla_x \log p_t(x | y) = \nabla_x \log p_t(x) + \nabla_x \log p_t(y | x) - \underbrace{\nabla_x \log p_t(y)}_{=0}.$$

- ▶ $\nabla_x \log p_t(x)$: unconditional score (learned by the diffusion model).
- ▶ $\nabla_x \log p_t(y | x)$: **extra term needed** to steer toward class y .

Practical approximation

$$\nabla_x \log p_t(x | y) \approx s_\theta(x, t) + \underbrace{\nabla_x \log p_\phi(y | x, t)}_?$$

This is the only change: replace the score by a *conditional score*.

CLASSIFIER GUIDANCE

REVERSE UPDATE WITH AN EXTRA GRADIENT TERM

Reverse-time SDE drift

$$dx = \left(f(x, t) - g(t)^2 \nabla_x \log p_t(x) \right) dt + g(t) d\bar{W}_t.$$

Guided drift (class y)

$$dx = \left(f(x, t) - g(t)^2 \underbrace{\nabla_x \log p_t(x | y)}_{\text{conditional score}} \right) dt + g(t) d\bar{W}_t.$$

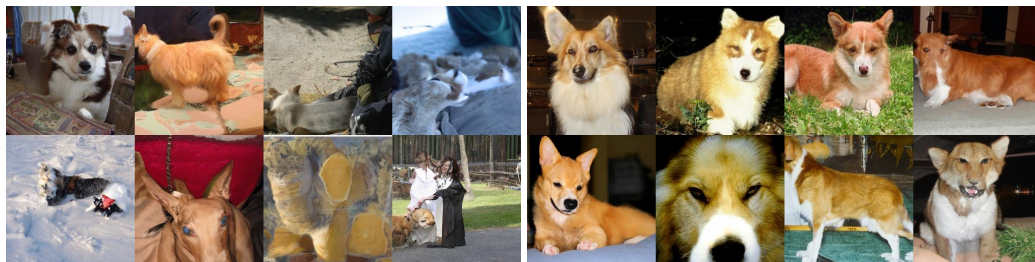
Using Slide 18:

$$\nabla_x \log p_t(x | y) \approx s_\theta(x, t) + \lambda \underbrace{\nabla_x \log p_\phi(y | x, t)}$$

where $\lambda \geq 0$ is the **guidance scale**.

CLASSIFIER GUIDANCE

QUALITATIVE EXAMPLES



Corgi samples with varying guidance scale λ (1 and 10)

CLASSIFIER-FREE GUIDANCE (CFG)

NO EXTERNAL CLASSIFIER NEEDED

Idea

- ▶ Train the same diffusion model in two modes:
 - **Conditional:** with condition c (class label / text prompt)
 - **Unconditional:** with condition dropped ($\emptyset$)
- ▶ At sampling time, combine both predictions to emulate guidance.

Noise-prediction form (most common)

$$\varepsilon_{\text{cond}} = \varepsilon_{\theta}(x_t, t, c), \quad \varepsilon_{\text{uncond}} = \varepsilon_{\theta}(x_t, t, \emptyset)$$

$$\varepsilon_{\text{CFG}} = (1 + w) \varepsilon_{\text{cond}} - w \varepsilon_{\text{uncond}}, \quad w \geq 0$$

Interpretation: push toward what the condition changes *relative to* the unconditional model.

Score form (equivalent)

$$s_{\text{CFG}}(x, t, c) = (1 + w) s_{\theta}(x, t, c) - w s_{\theta}(x, t, \emptyset).$$

Why it works

- ▶ Learns both $p_t(x | c)$ and $p_t(x)$.
- ▶ Avoids training a separate classifier at noisy levels.
- ▶ Guidance scale w trades:
 - larger w : stronger condition fidelity
 - too large w : artifacts / reduced diversity

Typical ranges

$w \approx 1\text{--}3$ (class cond)
 $w \approx 3\text{--}12$ (text-to-image)

CFG IN THE SAMPLER

PLUG THE COMBINED PREDICTION INTO THE USUAL UPDATE

Algorithm 10: Sampling with CFG (DDPM-style sketch)

Inputs: $\{\alpha_t, \beta_t, \bar{\alpha}_t, \sigma_t\}_{t=1}^T$, model ε_θ , condition c , scale w

Sample $x_T \sim \mathcal{N}(0, \mathbf{I})$

for $t \leftarrow T; t \geq 1; t \leftarrow t - 1$ **do**

$\varepsilon_{\text{cond}} \leftarrow \varepsilon_\theta(x_t, t, c)$

$\varepsilon_{\text{uncond}} \leftarrow \varepsilon_\theta(x_t, t, \emptyset)$

$\varepsilon_{\text{CFG}} \leftarrow (1 + w)\varepsilon_{\text{cond}} - w\varepsilon_{\text{uncond}}$

$\mu_t \leftarrow \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_{\text{CFG}} \right)$

if $t > 1$ **then**

 Sample $z \sim \mathcal{N}(0, \mathbf{I})$

$x_{t-1} \leftarrow \mu_t + \sigma_t z$

else

$x_0 \leftarrow \mu_t$

end

end

return x_0

Training trick (one line)

$$c \leftarrow \begin{cases} \emptyset & \text{with prob. } p_{\text{drop}} \\ c & \text{otherwise} \end{cases}$$

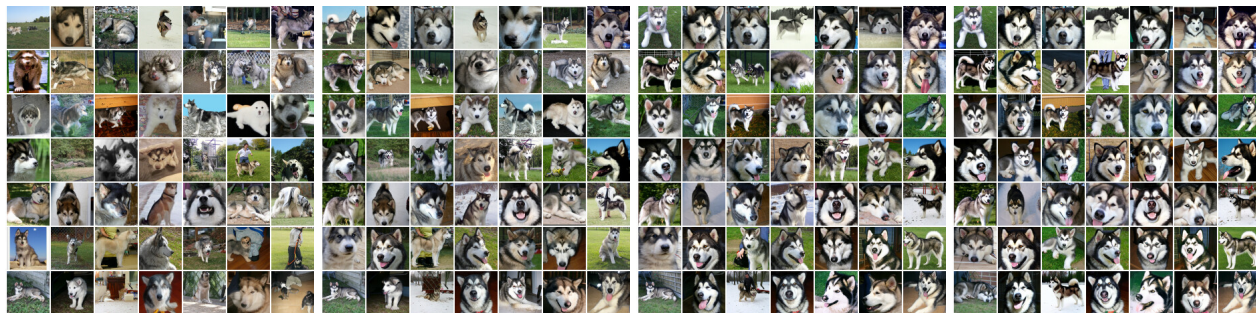
so the same network learns both modes.

What students should notice

- ▶ Only change is replacing ε_θ by ε_{CFG} .
- ▶ No classifier gradients needed.

CFG IN PRACTICE

SAMPLES FROM (HO ET AL., 2022)



Text-to-image samples with classifier-free guidance

CLIP GUIDANCE

USING TEXT-IMAGE ALIGNMENT AS AN ENERGY

CLIP (Radford et al., 2021) learns a joint embedding space for images and text.

Encoders

$$v(x) \in \mathbb{R}^d \quad (\text{image encoder}),$$

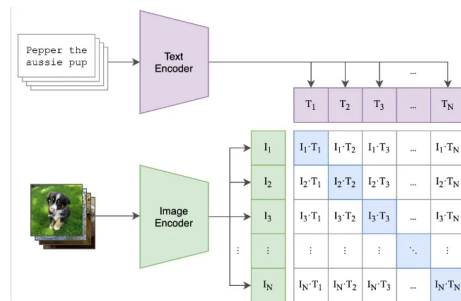
$$u(p) \in \mathbb{R}^d \quad (\text{text encoder, prompt } p).$$

Similarity score (typically cosine)

$$\text{sim}(x, p) = \frac{\langle v(x), u(p) \rangle}{\|v(x)\| \|u(p)\|}.$$

Training objective (contrastive)

- ▶ Pull matched (image, caption) pairs together.
- ▶ Push mismatched pairs apart.
- ▶ Result: **open-vocabulary** text-image alignment.



CLIP encoders (Radford et al., 2021)

Why it matters for diffusion

- ▶ CLIP provides a **generic differentiable preference**: x should maximize $\text{sim}(x, p)$.
- ▶ Can be used as a **guidance signal** without a class label.

CLIP GUIDANCE

HOW CLIP SIMILARITY ENTERS THE SAMPLER

Turn similarity into an energy

$$E(x, t; p) := -\text{sim}(x, p) \Rightarrow \nabla_x \log p(p \mid x_t) \propto \nabla_x \text{sim}(x_t, p).$$

Guided score (analogy with classifier guidance)

$$s_{\text{guided}}(x, t, p) = s_{\theta}(x, t) + \lambda \underbrace{\nabla_x \text{sim}(x, p)}_{\text{CLIP gradient}}$$

where $\lambda \geq 0$ is the guidance scale.

One-step schematic update (SDE)

$$x \leftarrow x + \Delta t \left(f(x, t) - g(t)^2 s_{\text{guided}}(x, t, p) \right) + g(t) \sqrt{|\Delta t|} z.$$

Notes

- ▶ CLIP turns a text prompt into a **differentiable direction** in image space.
- ▶ This is conceptually simple but can be **unstable** (adversarial directions).
- ▶ CFG usually gives a better quality/speed trade-off in practice.

LATENT DIFFUSION MODELS (LDM)

DIFFUSE IN A COMPRESSED SPACE

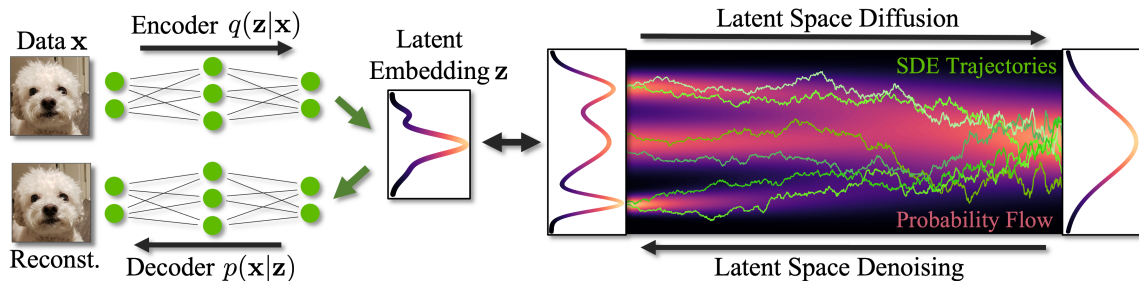
Problem: pixel-space diffusion is expensive (memory & compute).

Key idea (Rombach et al., 2022)

- ▶ Learn an autoencoder E_ψ, D_ψ such that $z = E_\psi(x)$ is a **low-dimensional latent**.
- ▶ Run diffusion in latent space z instead of image space x .
- ▶ Decode at the end: $\hat{x} = D_\psi(z_0)$.

Why it works

- ▶ Diffusion focuses on **semantics/structure** in a smoother space.
- ▶ Massive speedups enable high-res text-to-image.



Latent Diffusion Model pipeline (Rombach et al., 2022)

PROMPTING IN LATENT DIFFUSION

TEXT CONDITIONING & NEGATIVE PROMPTS

Text conditioning (e.g. Stable Diffusion)

- ▶ Tokenize the prompt p and embed it: $u(p) \in \mathbb{R}^{L \times d}$.
- ▶ Inject text via **cross-attention** in the U-Net blocks.
- ▶ The denoiser becomes $\varepsilon_{\theta}(z_t, t, u(p))$ (or score form).

Classifier-free guidance with prompts

$$\varepsilon_{\text{CFG}}(z_t, t, p) = (1 + w) \varepsilon_{\theta}(z_t, t, p) - w \varepsilon_{\theta}(z_t, t, \emptyset).$$

Negative prompt (same mechanism)

- ▶ Provide a second text p^- describing what you *do not want*.
- ▶ Use it as the “unconditional” branch: $\varepsilon_{\theta}(z_t, t, \emptyset) \leftarrow \varepsilon_{\theta}(z_t, t, p^-)$.

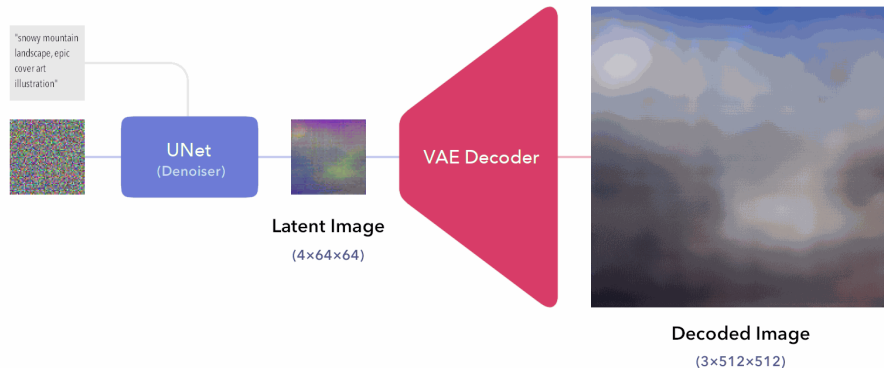
Practical knob: increasing w enforces the prompt, but too large may reduce diversity or introduce artifacts.

LATENT DIFFUSION

WHY IT IS FAST (ORDER-OF-MAGNITUDE VIEW)

- ▶ Pixel-space U-Net cost scales roughly with spatial size: $\mathcal{O}(HW)$.
- ▶ If the autoencoder downsamples by f , latent spatial size is $H/f \times W/f$.
- ▶ Compute in the denoiser becomes $\approx f^2$ **times cheaper**.

Example: $f = 8 \Rightarrow 64\times$ fewer spatial positions.



WRAP-UP

KEY TAKEAWAYS ON DIFFUSION

- ▶ **Core principle:** generation as *iterative denoising* from Gaussian noise.
- ▶ **Forward process:** simple Gaussian corruption $x_t = s_t x_0 + \sigma_t \varepsilon$ defines a continuum of easier problems.
- ▶ **What we learn:** a time-dependent denoiser / score $s_\theta(x, t) \approx \nabla_x \log p_t(x)$.
- ▶ **Sampling = reverse dynamics:**
 - DDPM (discrete): ancestral stochastic steps.
 - **Reverse SDE** (stochastic) and **probability-flow ODE** (deterministic) are two equivalent views.
 -
- ▶ **Control via guidance:**
 - classifier guidance adds an external gradient term,
 - CFG is the practical default for text-to-image.
- ▶ **Scaling to high resolution:** **latent diffusion** makes denoising $\approx f^2$ cheaper.
- ▶ **Main trade-offs:** #steps / compute $\leftrightarrow$ sample quality, and guidance strength $\leftrightarrow$ diversity.

Part IV

FLOW MATCHING MODELS

Reminders on optimal transport and flow matching (next section) taken from Samuel Boité.

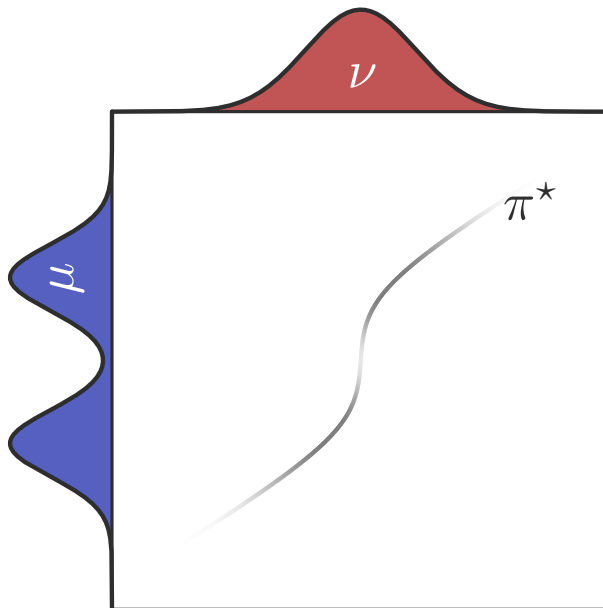
Probability measures μ and ν with finite second moments.

Probability measures μ and ν with finite second moments.

$$W_2^2(\mu, \nu) = \min_{\substack{\pi \text{ coupling} \\ \text{of } \mu \text{ and } \nu}} \iint_{\mathbb{R}^d \times \mathbb{R}^d} \|x - y\|^2 \, \mathrm{d}\pi(x, y)$$

Probability measures μ and ν with finite second moments.

$$W_2^2(\mu, \nu) = \min_{\substack{\pi \text{ coupling} \\ \text{of } \mu \text{ and } \nu}} \iint_{\mathbb{R}^d \times \mathbb{R}^d} \|x - y\|^2 d\pi(x, y)$$



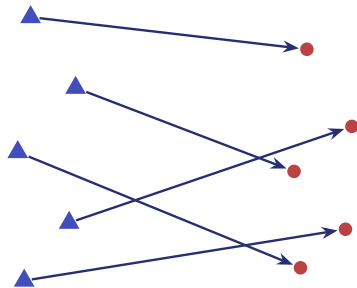
Empirical measures $\hat{\mu}_k = \frac{1}{k} \sum_{i=1}^k \delta_{x_i}$ and $\hat{\nu}_k = \frac{1}{k} \sum_{j=1}^k \delta_{y_j}$ on $\mathbb{R}^d$.

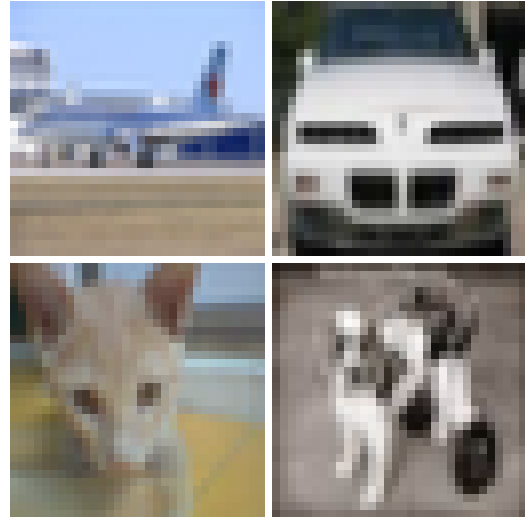
Empirical measures $\hat{\mu}_k = \frac{1}{k} \sum_{i=1}^k \delta_{x_i}$ and $\hat{\nu}_k = \frac{1}{k} \sum_{j=1}^k \delta_{y_j}$ on $\mathbb{R}^d$.

$$W_2^2(\hat{\mu}_k, \hat{\nu}_k) = \min_{\sigma \in \mathfrak{S}_k} \frac{1}{k} \sum_{i=1}^k \|x_i - y_{\sigma(i)}\|^2.$$

Empirical measures $\hat{\mu}_k = \frac{1}{k} \sum_{i=1}^k \delta_{x_i}$ and $\hat{\nu}_k = \frac{1}{k} \sum_{j=1}^k \delta_{y_j}$ on $\mathbb{R}^d$.

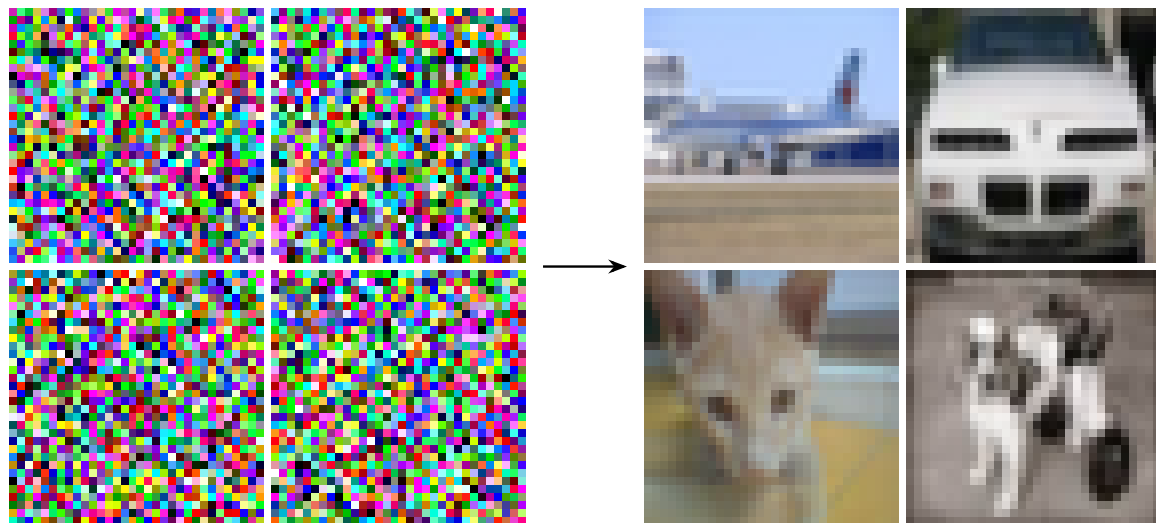
$$W_2^2(\hat{\mu}_k, \hat{\nu}_k) = \min_{\sigma \in \mathfrak{S}_k} \frac{1}{k} \sum_{i=1}^k \|x_i - y_{\sigma(i)}\|^2.$$





ν





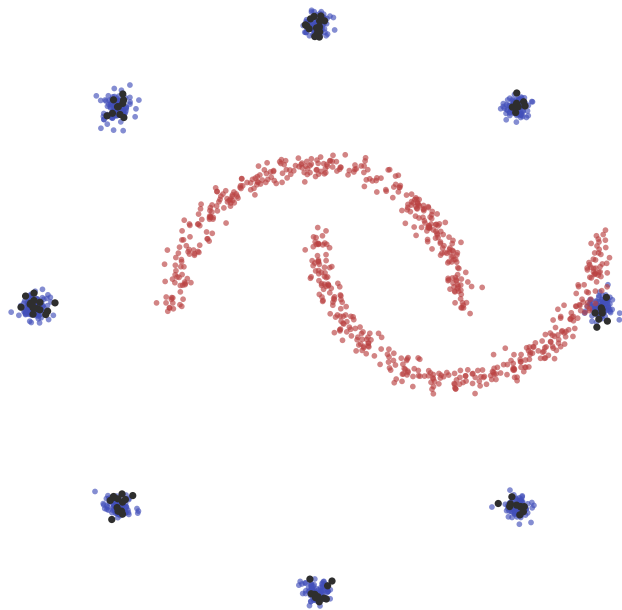
μ



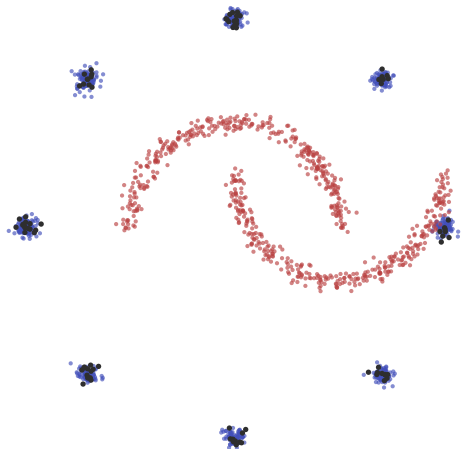
ν



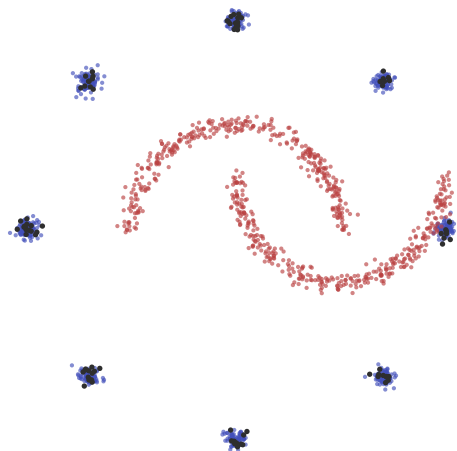
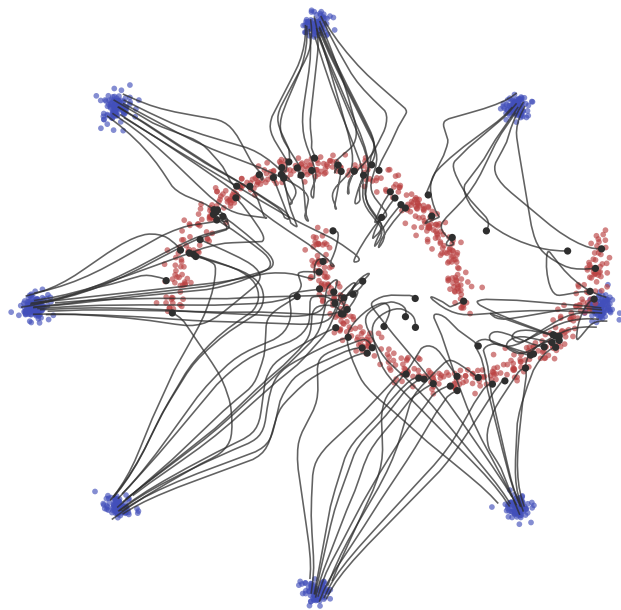
$$\dot{z}(t) = v_t^\theta(z(t)) \quad \text{with} \quad z(0) \sim \mu$$



$$\dot{z}(t) = v_t^\theta(z(t)) \quad \text{with} \quad z(0) \sim \mu$$



$$\dot{z}(t) = v_t^\theta(z(t)) \quad \text{with} \quad z(0) \sim \mu$$



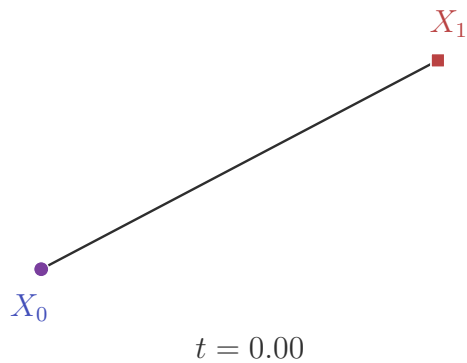
$(X_0, X_1) \sim \pi$ coupling of μ and ν

X_1



X_0

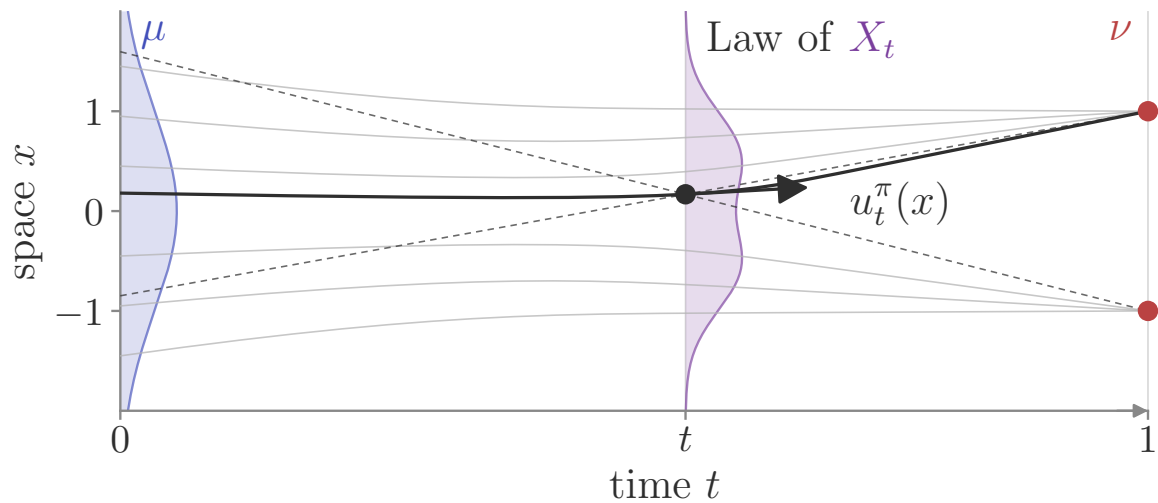
$(X_0, X_1) \sim \pi$ coupling of μ and ν



$$X_t = (1 - t)X_0 + tX_1$$

$$u_t^\pi(x) = \mathbb{E}_{(\textcolor{blue}{X}_0, \textcolor{red}{X}_1) \sim \pi}[\textcolor{red}{X}_1 - \textcolor{blue}{X}_0 \mid \textcolor{violet}{X}_t = x]$$

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

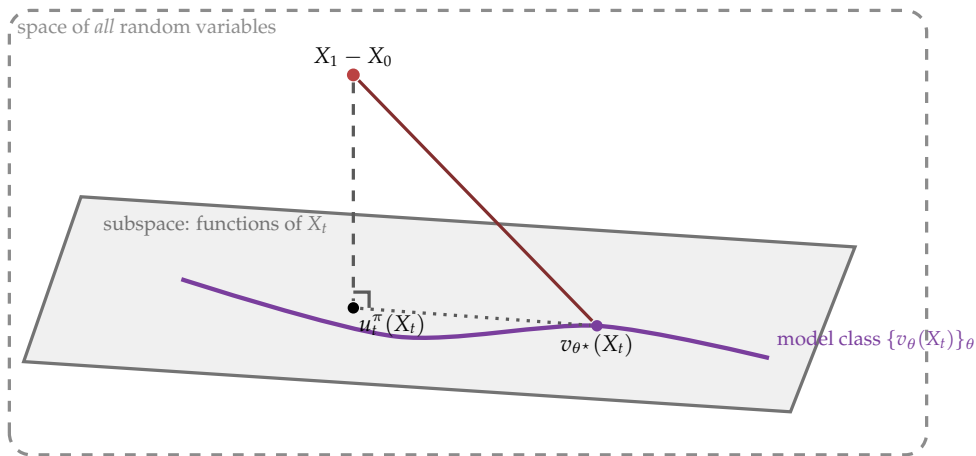


$$u_t^\pi(x) = \mathbb{E}_{(\textcolor{blue}{X}_0, \textcolor{red}{X}_1) \sim \pi}[\textcolor{red}{X}_1 - \textcolor{blue}{X}_0 \mid \textcolor{violet}{X}_t = x]$$

$$\min_{\theta} \mathbb{E}_{(\textcolor{blue}{X}_0, \textcolor{red}{X}_1) \sim \pi}[\|v_t^\theta(\textcolor{violet}{X}_t) - u_t^\pi(\textcolor{violet}{X}_t)\|^2]$$

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

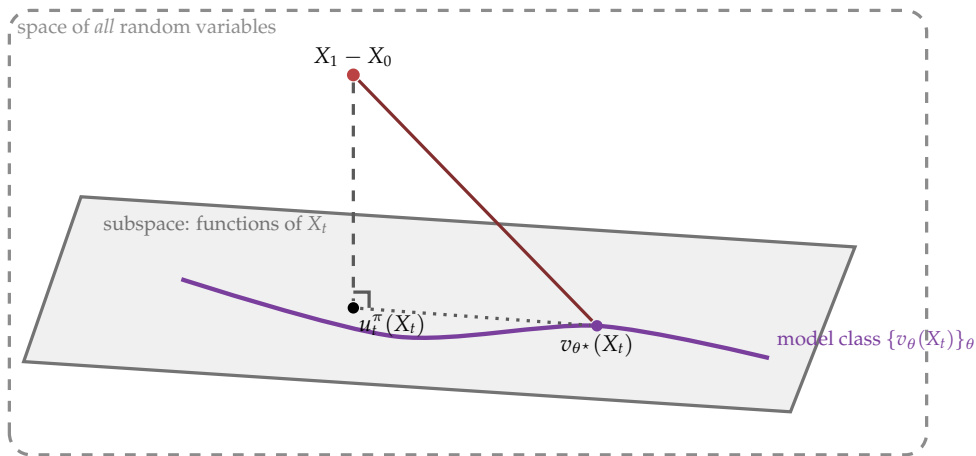
$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - u_t^\pi(X_t)\|^2]$$



[Pierret et al., 2026]

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - (X_1 - X_0)\|^2]$$



[Pierret et al., 2026]

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - (X_1 - X_0)\|^2]$$

Stochastic gradient descent:

- 1 Draw $(x_0, x_1) \leftarrow \pi$ and $t \leftarrow \mathcal{U}[0, 1]$

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - (X_1 - X_0)\|^2]$$

Stochastic gradient descent:

- 1 Draw $(x_0, x_1) \leftarrow \pi$ and $t \leftarrow \mathcal{U}[0, 1]$
- 2 $x_t = (1 - t)x_0 + tx_1$

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - (X_1 - X_0)\|^2]$$

Stochastic gradient descent:

- 1 Draw $(x_0, x_1) \leftarrow \pi$ and $t \leftarrow \mathcal{U}[0, 1]$
- 2 $x_t = (1 - t)x_0 + tx_1$
- 3 $\ell(\theta) = \|v_t^\theta(x_t) - (x_1 - x_0)\|^2$

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - (X_1 - X_0)\|^2]$$

Stochastic gradient descent:

- 1 Draw $(x_0, x_1) \leftarrow \pi$ and $t \leftarrow \mathcal{U}[0, 1]$
- 2 $x_t = (1 - t)x_0 + tx_1$
- 3 $\ell(\theta) = \|v_t^\theta(x_t) - (x_1 - x_0)\|^2$
- 4 $\theta \leftarrow \theta - \eta \nabla_{\theta} \ell(\theta)$

$$u_t^\pi(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x]$$

$$\min_{\theta} \mathbb{E}_{(X_0, X_1) \sim \pi} [\|v_t^\theta(X_t) - (X_1 - X_0)\|^2]$$

Stochastic gradient descent:

- 1 Draw $(x_0, x_1) \leftarrow \pi$ and $t \leftarrow \mathcal{U}[0, 1]$
- 2 $x_t = (1 - t)x_0 + tx_1$
- 3 $\ell(\theta) = \|v_t^\theta(x_t) - (x_1 - x_0)\|^2$
- 4 $\theta \leftarrow \theta - \eta \nabla_{\theta} \ell(\theta)$
- 5 Repeat.

FROM DIFFUSION TO FLOW MATCHING

MOTIVATION

Recap (Part 3). The probability-flow ODE

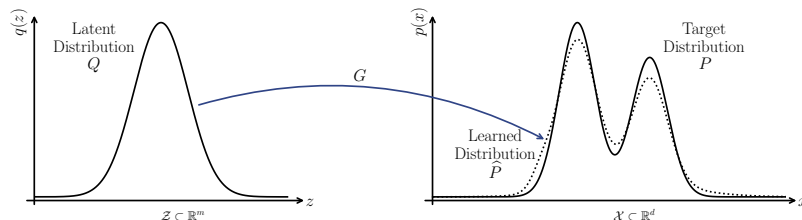
$$\frac{dx}{dt} = f(x, t) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x)$$

is a *deterministic* generator, but it was derived *after the fact* from a forward noising SDE and a learned score.

Question

Can we directly design a time-dependent velocity field v_t that transports noise to data, and train it with a simple regression loss — without ever simulating an SDE?

This is the idea of **Flow Matching** [17] and **Rectified Flow** [18]: simulation-free training of continuous normalizing flows.



SETUP AND NOTATION

CAREFUL: REVERSED TIME CONVENTION W.R.T. PART 3

- ▶ Source $X_0 \sim p_0$, **easy to sample** (e.g. $p_0 = \mathcal{N}(0, \mathbf{I}_d)$).
- ▶ Target $X_1 \sim p_1$, the **data distribution** (samples only, unknown density).

Convention flip

Here $t = 0$ is **noise** and $t = 1$ is **data** — the opposite of the DDPM/SDE convention used in Part 3, where x_0 was data and x_T was noise. This is the standard convention in the Flow Matching literature.

Goal. Find $v_t : \mathbb{R}^d \rightarrow \mathbb{R}^d, t \in [0, 1]$, such that the ODE

$$\dot{z}(t) = v_t(z(t)), \quad z(0) \sim p_0$$

satisfies $z(1) \sim p_1$. The law of $z(t)$ traces out a **probability path** $(p_t)_{t \in [0, 1]}$ with p_0, p_1 as endpoints, generated by v_t through the continuity equation $\partial_t p_t + \nabla \cdot (p_t v_t) = 0$.

CONDITIONAL PROBABILITY PATHS

FROM A SINGLE PAIR TO A GLOBAL PATH

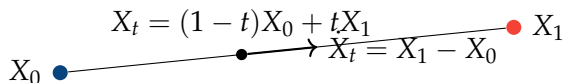
Directly targeting the marginal path p_t is intractable: it requires already knowing p_t . Instead, build it from simple **conditional** paths.

1. Draw a **coupling** $(X_0, X_1) \sim \pi$ of p_0 and p_1 (default: independent, $\pi = p_0 \otimes p_1$).
2. Connect the pair with a simple interpolant, e.g. the **linear (OT) path**:

$$X_t = (1 - t)X_0 + tX_1, \quad t \in [0, 1].$$

3. Its conditional velocity is constant along the segment:

$$\dot{X}_t = X_1 - X_0.$$



THE MARGINAL VECTOR FIELD

WHAT WE ACTUALLY WANT TO LEARN

Marginalizing the conditional path over the coupling gives the true marginal path

$$p_t(x) = \int p_t(x \mid x_0, x_1) \mathrm{d}\pi(x_0, x_1),$$

and it is generated by the **marginal velocity field**

$$u_t(x) = \mathbb{E}_{(X_0, X_1) \sim \pi} [X_1 - X_0 \mid X_t = x].$$

Marginalization trick [17]

The ODE driven by u_t generates *exactly* p_t : a mixture of conditional flows is itself generated by the (conditional-expectation) mixture of their velocity fields.

Problem: $u_t(x)$ is itself an intractable conditional expectation over every pair consistent with $X_t = x$ — the same obstacle as the score $\nabla_x \log p_t(x)$ in Part 3.

CONDITIONAL FLOW MATCHING (CFM)

REGRESSION INSTEAD OF AN INTRACTABLE TARGET

Ideal, intractable objective (FM):

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}[0,1], X_t \sim p_t} [\|v_\theta(X_t, t) - u_t(X_t)\|^2].$$

Conditional Flow Matching: replace the marginal target $u_t(X_t)$ by the conditional velocity $X_1 - X_0$, which is tractable:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}[0,1], (X_0, X_1) \sim \pi} [\|v_\theta(X_t, t) - (X_1 - X_0)\|^2].$$

Theorem 1.1 (17)

$\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta)$: both objectives share the same minimizer $v_\theta = u_t$, yet CFM only needs samples (x_0, x_1, t) — *simulation-free*.

Proof idea (L^2 projection / Pythagoras). $u_t(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x]$ is the orthogonal projection of $X_1 - X_0$ onto functions of X_t ; expanding $\|v_\theta(X_t) - (X_1 - X_0)\|^2$ around $u_t(X_t)$ leaves a cross term that vanishes in expectation and a θ -independent constant.

TRAINING ALGORITHM

Algorithm 11: Conditional Flow Matching (SGD)

Input: coupling π of p_0, p_1 (e.g. independent)

repeat

 Draw $(x_0, x_1) \sim \pi$ and $t \sim \mathcal{U}[0, 1]$

$x_t \leftarrow (1 - t)x_0 + tx_1$

$\ell(\theta) \leftarrow \|v_\theta(x_t, t) - (x_1 - x_0)\|^2$

$\theta \leftarrow \theta - \eta \nabla_\theta \ell(\theta)$

until *convergence*

Notes

- ▶ No forward diffusion, no noise schedule β_t or $\sigma(t)$ to tune.
- ▶ One regression pass per step: as cheap as denoising-score matching.
- ▶ v_θ plays the role of s_θ in Part 3.

SAMPLING: INTEGRATING THE LEARNED FLOW

EULER DISCRETIZATION

Algorithm 12: ODE Sampling (Flow Matching)

Input: time grid $0 = t_0 < \dots < t_K = 1$, trained v_θ

Sample $z_{t_0} \sim p_0 = \mathcal{N}(0, \mathbf{I}_d)$

for $k \leftarrow 0; k < K; k \leftarrow k + 1$ **do**

$\Delta t \leftarrow t_{k+1} - t_k$

$z_{t_{k+1}} \leftarrow z_{t_k} + \Delta t v_\theta(z_{t_k}, t_k)$

end

return z_{t_K} // approx. sample from p_1

Notes

- ▶ Deterministic given the seed z_{t_0} .
- ▶ Same trade-off as Part 3: fewer steps $\Rightarrow$ faster but noisier samples.
- ▶ Higher-order solvers (Heun, RK) apply unchanged.

CHOICE OF COUPLING π

INDEPENDENT VS. OPTIMAL TRANSPORT

Independent coupling $\pi = p_0 \otimes p_1$

- ▶ Simple, unbiased, always available.
- ▶ Paths X_t from unrelated pairs **cross**, so u_t is hard to fit and trajectories curve $\Rightarrow$ many integration steps needed.

Rectified Flow [18]: iteratively *reflow* — resample pairs along the current learned flow and retrain — to straighten trajectories further, in the limit approaching the OT map.

Minibatch OT coupling [23, 28]

- ▶ Within a batch of size k , pair source/target samples by the OT assignment

$$\sigma^* \in \arg \min_{\sigma \in \mathfrak{S}_k} \sum_i \|x_i - y_{\sigma(i)}\|^2.$$

- ▶ Still simulation-free; **straighter** paths, fewer function evaluations at inference.

FLOW MATCHING VS. DIFFUSION

WRAP-UP

- ▶ **Diffusion (Part 3):** a *fixed* Gaussian noising process defines the path; a score network is learned to reverse it (SDE or probability-flow ODE).
- ▶ **Flow Matching:** *any* conditional path and coupling can be chosen (linear/OT interpolant, learned reflow, ...); the velocity field is regressed directly, with no reverse-time SDE derivation needed.
- ▶ **Unifying view** [1]: the probability-flow ODE of a Gaussian diffusion is itself a special case of a conditional-path flow — diffusion and flow matching are two instances of the same simulation-free recipe.
- ▶ **In practice:** simpler loss, no noise-schedule tuning, same ODE sampler toolbox as Part 3; coupling choice (independent vs. OT) is the main lever on sampling speed.

Part V

EVALUATION OF GENERATIVE MODELS & PROJECT PRESENTATIONS

THE VARIOUS PERFORMANCES OF GENERATIVE MODELS

MOTIVATION

As the generation becomes better, the evaluation becomes more challenging.

DALL-E 2 (2023)



Midjourney v5 (2023)



?

Prompt: *A dog playing with a child.*

THE VARIOUS PERFORMANCES OF GENERATIVE MODELS

MOTIVATION

As the generation becomes better, the evaluation becomes more challenging.

DALL-E 2 (2023)



Low Quality
High Diversity

≠

Midjourney v5 (2023)



High Quality
Low Diversity

Prompt: *A dog playing with a child.*

WHAT IS A *good* GENERATIVE MODEL?

THREE AXES + PRACTICAL CONSTRAINTS

A generative model is judged by its samples and its behavior.

- ▶ **Fidelity / Quality:** samples look real (or read well).
- ▶ **Diversity / Coverage:** captures many modes; avoids collapse.
- ▶ **Conditioning / Usefulness:** follows labels/prompts; controllable.

Other important goals (often forgotten)

- ▶ **Novelty:** not memorizing the training set.
- ▶ **Efficiency:** sampling speed, compute, memory.
- ▶ **Robustness:** stable across seeds / prompts / datasets.

WHY EVALUATION IS HARD

NO SINGLE NUMBER CAPTURES EVERYTHING

- ▶ **Quality vs diversity** is a fundamental trade-off.
- ▶ Most metrics are **imperfect indicators** of human judgement (and can be fooled).
- ▶ In high dimensions, density comparison is difficult:
 - likelihood may be unavailable or misleading,
 - perceptual similarity is not pixel-wise.
- ▶ Conditioning adds a third axis: **“looks good”** vs **“matches the prompt”**.

EVALUATION TOOLBOX

THREE COMPLEMENTARY FAMILIES

- ▶ **Human evaluation:** pairwise preference, rating, error analysis.
- ▶ **Automatic distribution metrics:** compare samples vs real data.
- ▶ **Task-based evaluation:** downstream utility (retrieval, captioning, etc.).

Good practice: report several metrics and show failure cases.

PROTOCOL MATTERS

MAKE COMPARISONS MEANINGFUL

- ▶ Fix **data split**, preprocessing, and resolution.
- ▶ Use enough samples (FID/KID are sample-size sensitive).
- ▶ Report **mean** $\pm$ **std** across seeds when possible.
- ▶ Avoid cherry-picking prompts/samples; disclose prompt set.

REMINDER: ENTROPY

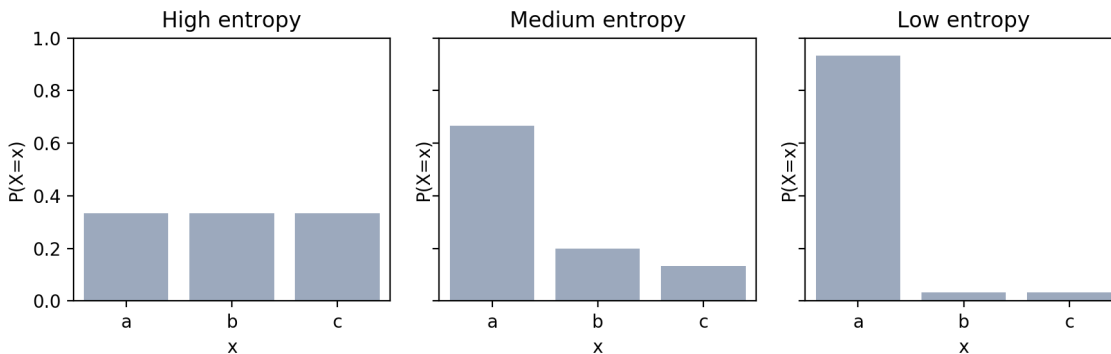
A 30-SECOND RECAP

Entropy (discrete)

$$H(P) = - \sum_y P(y) \log P(y).$$

Interpretation

- ▶ Low entropy: probability mass concentrated ("confident" / "peaked").
- ▶ High entropy: spread out ("uncertain" / "uniform").



TRADITIONAL METRICS

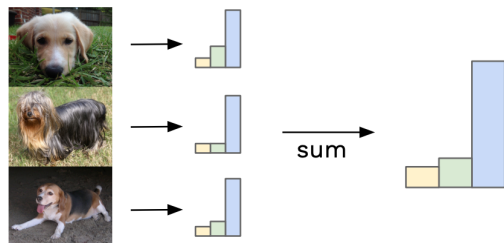
INCEPTION SCORE

$$\log \left(\text{IS}(\hat{P}) \right) = H \left(\mathbb{E}_{x \sim \hat{P}} [\mathbb{P}(Y|x)] \right) - \mathbb{E}_{x \sim \hat{P}} [H(\mathbb{P}(Y|x))] .$$

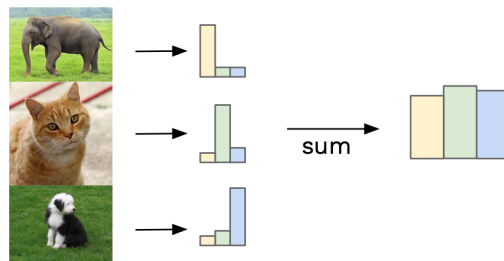
Inception Score is maximized :

- ▶ if for every $x \sim \hat{P}$, $H_Y(\mathbb{P}(Y|x))$ is minimized. This suggests that the classification model is highly confident in predicting a singular label per image, which implies that the images are clearly recognizable and of high quality.
- ▶ if the entropy of $\mathbb{E}_{x \sim \hat{P}} [\mathbb{P}(Y|x)]$ is maximized. The label predictions must be uniformly distributed over all possible labels. This indicates that the generative model generates heterogeneous labels, thereby ensuring diversity.

Similar labels sum to give focussed distribution



Different labels sum to give uniform distribution



FRÉCHET INCEPTION DISTANCE (FID)

2-WASSERSTEIN BETWEEN GAUSSIANS IN FEATURE SPACE

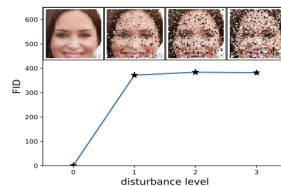
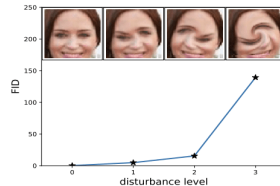
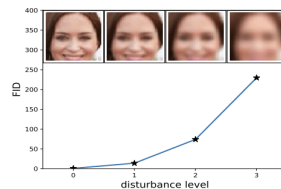
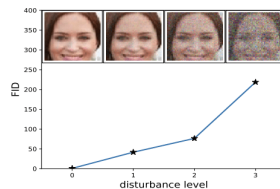
Definition (Heusel et al., 2017)

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

with (μ, Σ) computed from Inception features.

Pros / Cons

- ▶ (+) compares to **real data**; sensitive to mode dropping.
- ▶ (-) Gaussian assumption; biased with small sample sizes.
- ▶ (-) still collapses quality+diversity into one number.



TRADITIONAL METRICS

FID FALLS SHORT

For example, Kynkäänniemi et al. [16] highlights the limitations of FID with samples drawn from StyleGAN:



(a) Set A - FID= 91.7

(b) Set B - FID= 16.9

(c) Set C - FID= 4.5

(d) Set D - FID= 16.7

PRECISION AND RECALL FOR GENERATIVE MODELS

METRICS TO EVALUATE QUALITY AND DIVERSITY

To assess models, we use the notion of Precision and Recall, inspired from Information Retrieval:

Quality

Diversity

PRECISION AND RECALL FOR GENERATIVE MODELS

METRICS TO EVALUATE QUALITY AND DIVERSITY

To assess models, we use the notion of Precision and Recall, inspired from Information Retrieval:

Quality

Diversity



Precision

=

What proportion of **generated** samples are
realistic?

PRECISION AND RECALL FOR GENERATIVE MODELS

METRICS TO EVALUATE QUALITY AND DIVERSITY

To assess models, we use the notion of Precision and Recall, inspired from Information Retrieval:

Quality



Precision

=

What proportion of **generated** samples are **realistic**?

Diversity



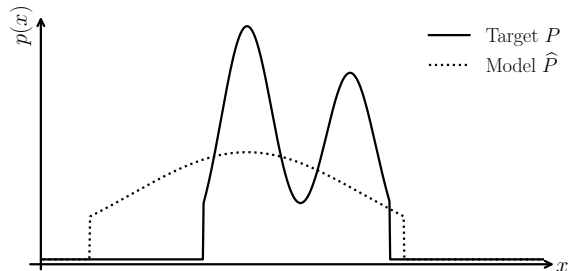
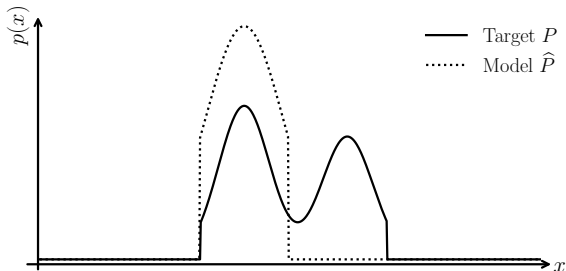
Recall

=

What proportion of **real** samples can be **generated**?

PRECISION AND RECALL FOR GENERATIVE MODELS

FOR FINITE SUPPORT



PRECISION AND RECALL FOR GENERATIVE MODELS

FOR FINITE SUPPORT

Definition 1.1 (Support-Based Precision and Recall - [16].)

For any distributions $P \in \mathcal{P}(\mathcal{X})$ and $\hat{P} \in \mathcal{P}(\mathcal{X})$, we say that the distribution P has precision $\bar{\alpha}$ at recall $\bar{\beta}$ with respect to $\hat{P}$ if

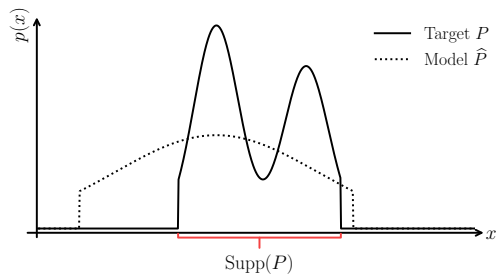
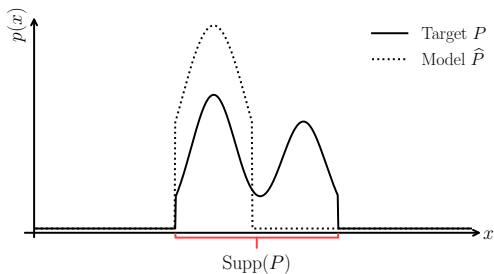
$$\bar{\alpha} := \hat{P}(\text{Supp}(P)) \quad \text{and} \quad \bar{\beta} := P(\text{Supp}(\hat{P})).$$

PRECISION AND RECALL FOR GENERATIVE MODELS

FOR FINITE SUPPORT

Precision for finite support is the proportion of generated data that lies on the support of the real data:

$$\bar{\alpha} = \hat{P}(\text{Supp}(P)).$$

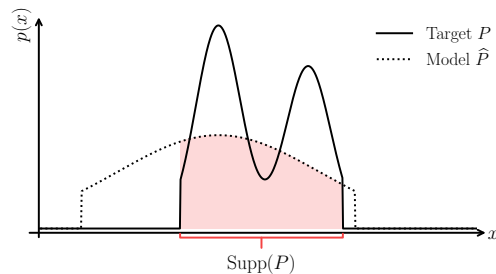
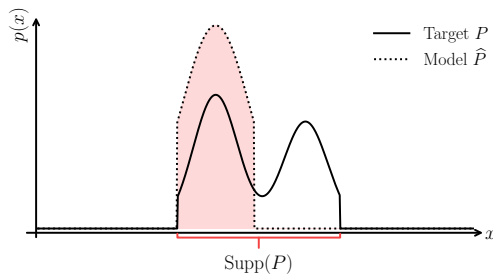


PRECISION AND RECALL FOR GENERATIVE MODELS

FOR FINITE SUPPORT

Precision for finite support is the proportion of generated data that lies on the support of the real data:

$$\bar{\alpha} = \hat{P}(\text{Supp}(P)).$$

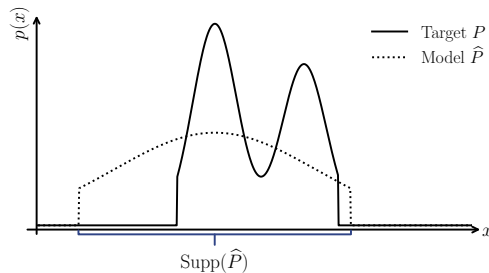
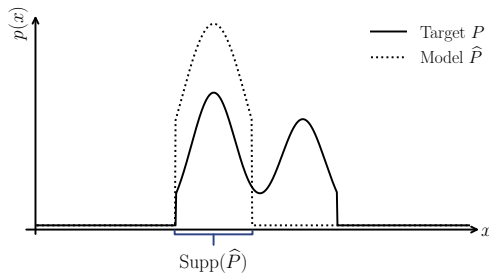


PRECISION AND RECALL FOR GENERATIVE MODELS

FOR FINITE SUPPORT

Recall for finite support is the proportion of the support of the real data that is covered by the generated data:

$$\bar{\beta} = P(\text{Supp}(\hat{P})).$$

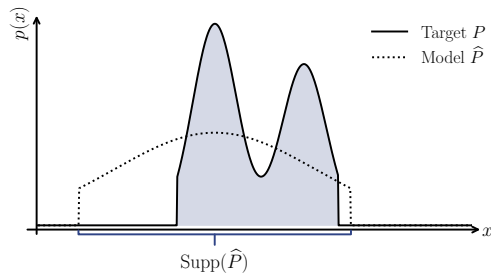
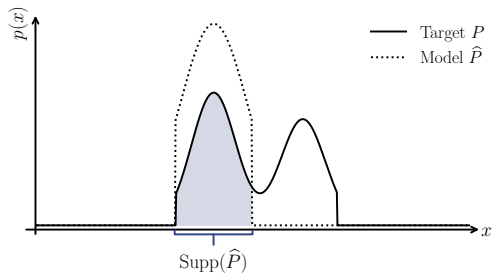


PRECISION AND RECALL FOR GENERATIVE MODELS

FOR FINITE SUPPORT

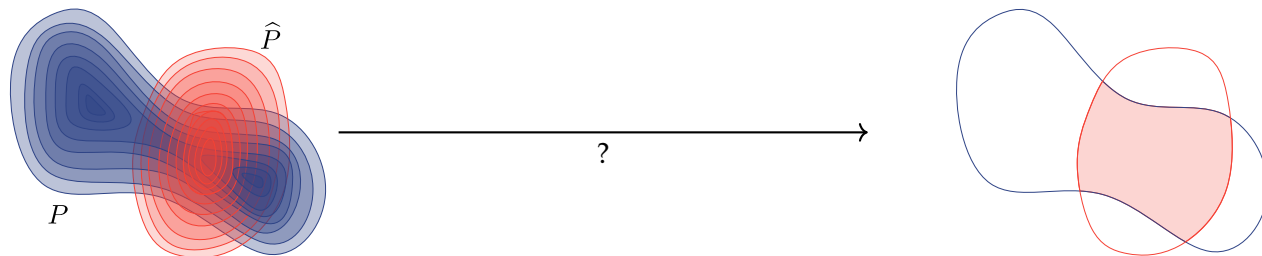
Recall for finite support is the proportion of the support of the real data that is covered by the generated data:

$$\bar{\beta} = P(\text{Supp}(\hat{P})).$$



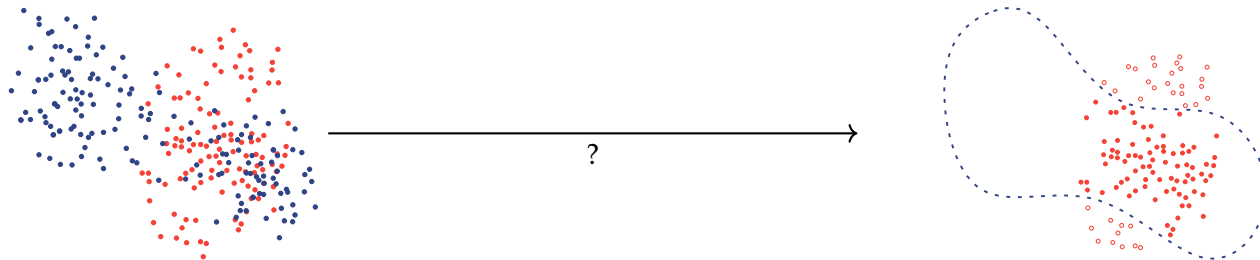
PRECISION AND RECALL FOR GENERATIVE MODELS

ESTIMATING THE PRECISION AND RECALL



PRECISION AND RECALL FOR GENERATIVE MODELS

ESTIMATING THE PRECISION AND RECALL



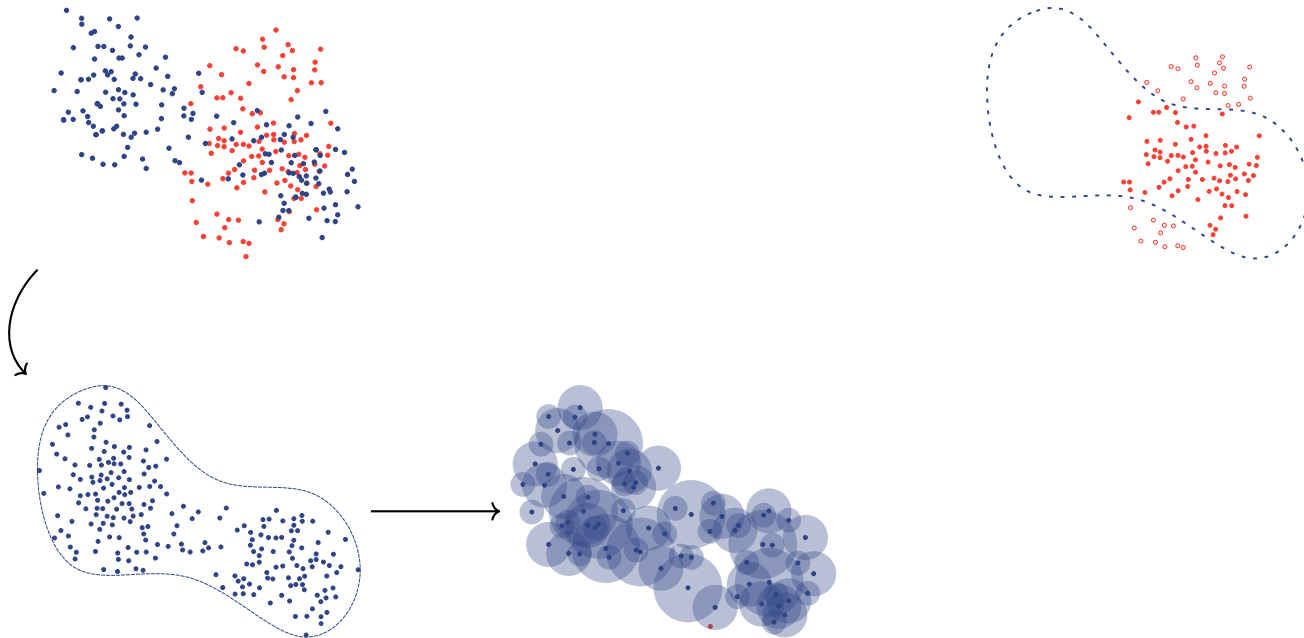
PRECISION AND RECALL FOR GENERATIVE MODELS

ESTIMATING THE PRECISION AND RECALL



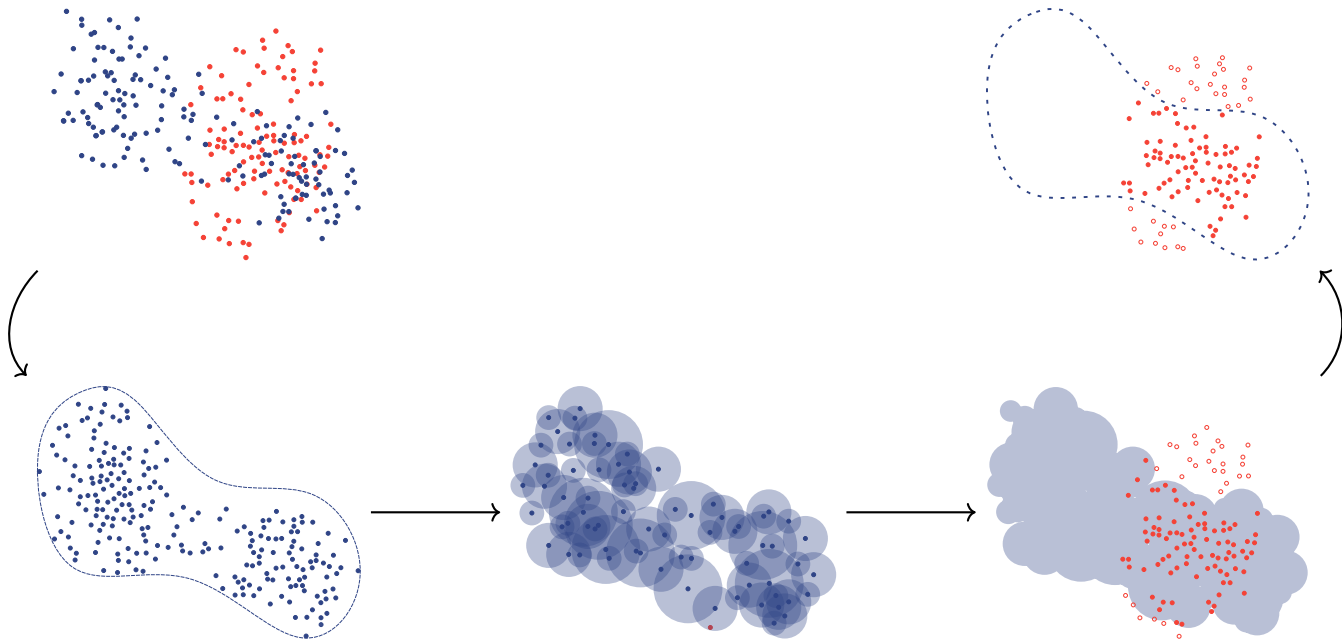
PRECISION AND RECALL FOR GENERATIVE MODELS

ESTIMATING THE PRECISION AND RECALL



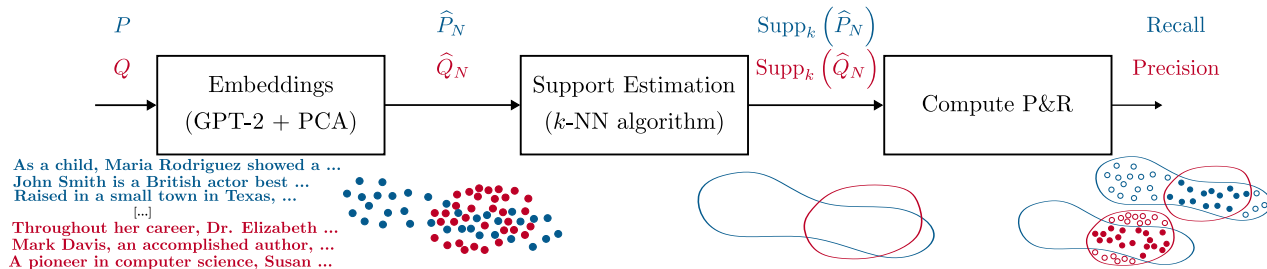
PRECISION AND RECALL FOR GENERATIVE MODELS

ESTIMATING THE PRECISION AND RECALL



PRECISION AND RECALL FOR GENERATIVE MODELS

FOR LLMs



PRECISION AND RECALL FOR GENERATIVE MODELS

IN PRACTICE

MNIST Dataset [29]



PRECISION AND RECALL FOR GENERATIVE MODELS

IN PRACTICE

Low Precision High Recall



Precision: 0.54 Recall: 0.91

MNIST Dataset [29]



High Precision Low Recall

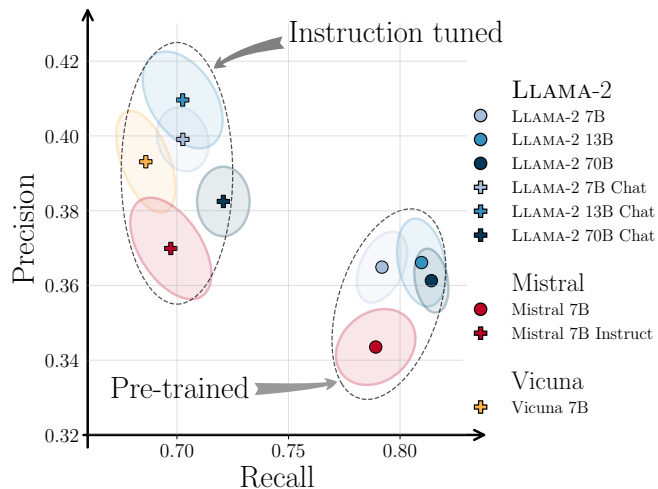


Precision: 0.80 Recall: 0.70

PRECISION AND RECALL FOR GENERATIVE MODELS

FOR LLMs

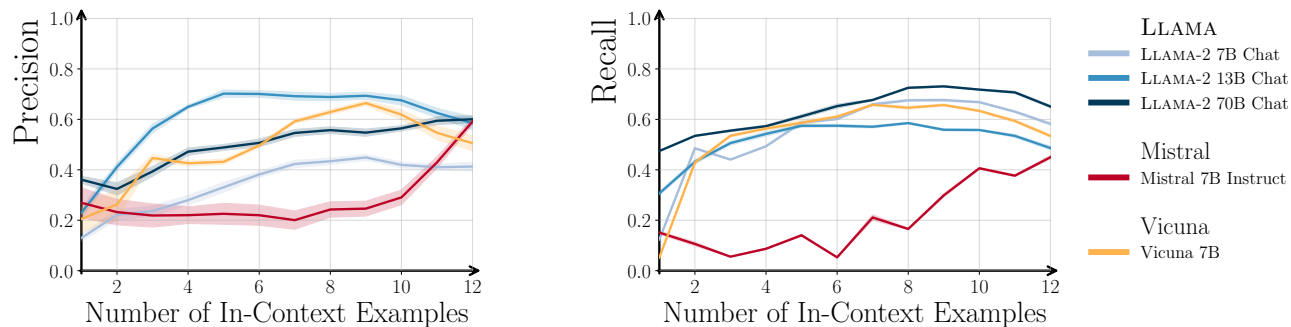
On open-ended generation, the quality and diversity of LLMs can also be evaluated using Precision and Recall: Bronnec et al. [5]



PRECISION AND RECALL FOR GENERATIVE MODELS

FOR LLMs

We can also evaluate the quality and diversity of LLMs on Chatbot open-ended generation. We can for instance check the impact of In-Context examples on the quality and diversity of the generated text. For instance on Wikipedia Biographies generation:



DIVERSITY / COLLAPSE CHECKS

COMPLEMENTARY HEURISTICS

Why: a model can have good FID but still be repetitive.

- ▶ **MS-SSIM (or LPIPS):** average similarity between generated images.
 - higher similarity $\Rightarrow$ lower diversity.
- ▶ **Per-class metrics:** compute FID/PR per class (when labels exist).
- ▶ **Nearest-neighbor visualization:** check for duplicates.

MEMORIZATION AND OVERFITTING

A NECESSARY SANITY CHECK

Question: is the model generating *new* samples?

- ▶ **Feature-space nearest neighbor:** for each generated sample, retrieve closest training image.
- ▶ **Distance statistics:** distribution of NN distances (train vs test).
- ▶ **Advanced tests:** can we tell if an image was in the training set?

PROMPT ADHERENCE METRICS

QUALITY IS NOT ALIGNMENT

Text-to-image: a sample can look great but not match the prompt.

CLIP-based alignment

- ▶ Compute similarity between image and prompt embeddings.
- ▶ Often reported as **CLIPScore**-like metrics.

Caveats

- ▶ CLIP can be fooled (it can reward strange shortcuts).
- ▶ High CLIP similarity does not guarantee realism (or correctness).
- ▶ Always complement with FID/PR + human inspection.

CLIPSCORE EXAMPLE

PROMPT: "A DOG PLAYING WITH A CHILD."

Prompt Adherence vs. Realism

Prompt: "A dog playing with a child."



CLIP Similarity (illustrative): **High**

Prompt: "A dog playing with a child."



CLIP Similarity (illustrative): **Medium**

Prompt: "A dog playing with a child."



CLIP Similarity (illustrative): **Medium**

Prompt: "A dog playing with a child."



CLIP Similarity (illustrative): **Low**

CLIPSCORE EXAMPLE

PROMPT: "A DOG PLAYING WITH A CHILD."

CLIP Can Be Fooled

Prompt: "A dog playing with a child."



CLIP Similarity (illustrative): High

Prompt: "A dog playing with a child."



CLIP Similarity (illustrative): High

HUMAN EVALUATION FOR IMAGES

SIMPLE DESIGNS THAT WORK

- ▶ **Pairwise preference** (A/B): “Which image is better for this prompt?”
- ▶ **Likert ratings**: realism, prompt match, artifacts (separate axes).
- ▶ **Inter-rater agreement**: show that evaluators are consistent.

Practical tips

- ▶ Pre-register prompts; randomize order; blind the model identity.
- ▶ Keep evaluation short; provide calibration examples.

PERPLEXITY

LIKELIHOOD-BASED FLUENCY METRIC

Definition

$$\text{PPL} = \exp \left(- \frac{1}{N} \sum_{i=1}^N \log p_{\theta}(w_i \mid w_{<i}) \right)$$

Interpretation

- ▶ lower PPL $\Rightarrow$ the text is more probable under the model.
- ▶ good for language modeling; weaker for instruction-following preference.

Caveats

- ▶ depends on tokenizer / domain; not always comparable.
- ▶ a model can have low PPL and still be unhelpful.

REFERENCE-BASED OVERLAP METRICS

BLEU, ROUGE (WHEN YOU HAVE GOLD ANSWERS)

Use cases: translation, summarization, QA with references.

- ▶ **BLEU:** n-gram precision with brevity penalty.
- ▶ **ROUGE:** recall-oriented overlap (often ROUGE-L).

Caveats

- ▶ penalizes valid paraphrases; sensitive to phrasing.
- ▶ should be complemented with semantic metrics and human eval.

CHOOSING METRICS

MATCH THE METRIC TO THE QUESTION

A minimal “responsible” report (images)

- ▶ Quality: FID & Precision
- ▶ Coverage : Recall
- ▶ Memorization: nearest-neighbor check
- ▶ Conditioning: CLIP-style alignment + human A/B

A minimal “responsible” report (text)

- ▶ Fluency: perplexity (if meaningful)
- ▶ Diversity: Distinct-n
- ▶ Preference: human (or calibrated) pairwise win-rate

TAKEAWAYS

EVALUATION IS MULTI-DIMENSIONAL

- ▶ There is **no universal metric** for generative modeling.
- ▶ Report **multiple axes**: quality, diversity, alignment, memorization.
- ▶ Always include **qualitative examples** + failure cases.
- ▶ Protocol and transparency matter as much as the metric.

Rule of thumb

If one number is enough to summarize your model,
you probably missed something important.

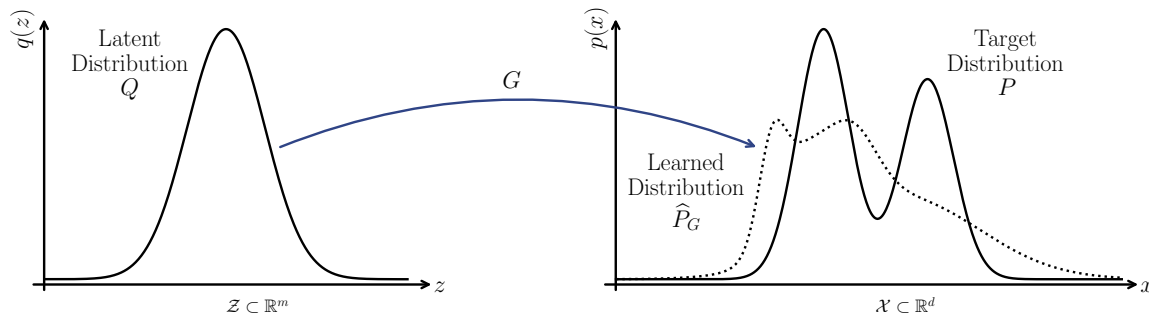
GENERATIVE ADVERSARIAL NETWORKS

ORIGINAL FRAMEWORK

- ▶ Let $G : \mathcal{Z} \rightarrow \mathcal{X}$ be a generator model parameterized by a neural network.
- ▶ Let $D : \mathcal{X} \rightarrow [0, 1]$ be a discriminator model parameterized by a neural network.

The original GAN framework [6] is defined by the following optimization problem:

$$\min_G \max_D \mathbb{E}_{x \sim P} [\log D(x)] + \mathbb{E}_{x \sim \hat{P}_G} [\log(1 - D(x))] . \quad (7)$$



GENERATIVE ADVERSARIAL NETWORKS

ORIGINAL FRAMEWORK

- ▶ Let $G : \mathcal{Z} \rightarrow \mathcal{X}$ be a generator model parameterized by a neural network.
- ▶ Let $D : \mathcal{X} \rightarrow [0, 1]$ be a discriminator model parameterized by a neural network.

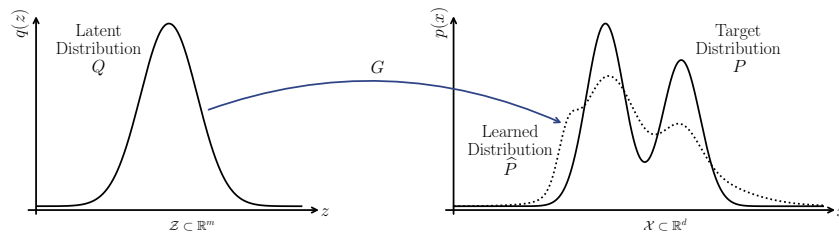
The original GAN framework [6] is defined by the following optimization problem:

$$\min_G \max_D \mathbb{E}_{x \sim P} [\log D(x)] + \mathbb{E}_{x \sim \hat{P}_G} [\log(1 - D(x))] . \quad (7)$$

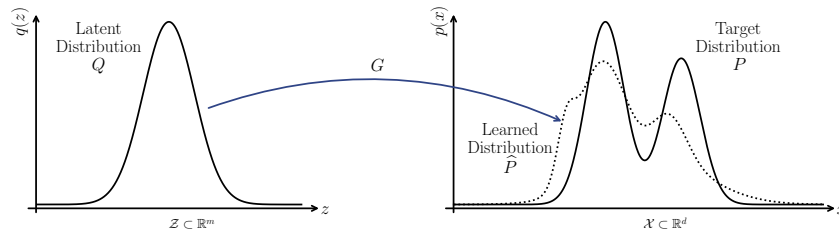
TUNING PRECISION AND RECALL IN GENERATIVE MODELS

TRUNCATION

Hard Truncation
Karras et al. [12]



Soft Truncation
Kingma and Dhariwal
[15]



PROJECT PRESENTATIONS

HARD TRUNCATION



Figure. From left to right: $\psi = 0.0$, $\psi = 0.3$
 $\psi = 0.7$ $\psi = 1.0$.

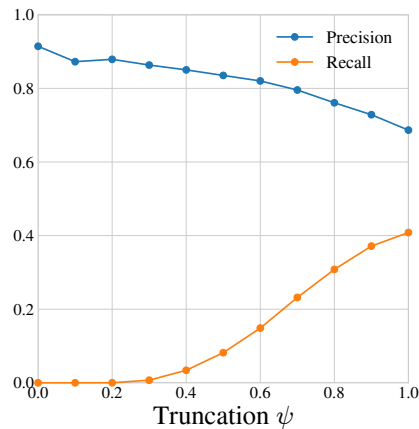


Figure. Source: [16]

PROJECT PRESENTATIONS

SOFT TRUNCATION

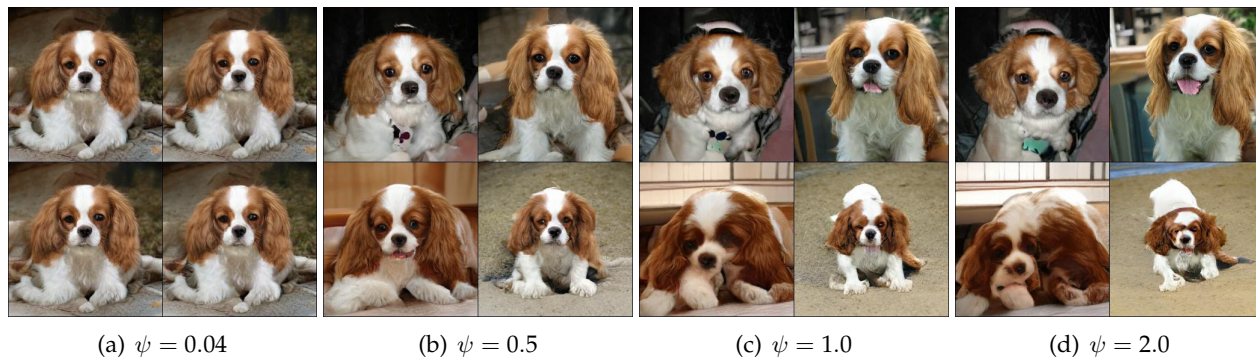


Figure. Soft-Truncation on BigGAN. Source:[4].

FINAL PROJECT

TUNING QUALITY AND DIVERSITY IN GANS

- ▶ **Goal:** Implement a GAN architecture to generate samples from a given dataset.
- ▶ **Dataset:** MNIST
- ▶ **Code:** Use the repository given in the Project
`https://github.com/AlexVerine/DataLabAssignement2`
- ▶ **Enrolling:** GitHub Classroom `https://classroom.github.com/a/irBR3g3S`
- ▶ **Unifying yourself:** Choose different methods than your class mates
`https://docs.google.com/spreadsheets/d/1V8cfjwByJdssL-fupBpbHqmrZpZcHXBAIvMhKSepIG0/edit?usp=sharing`

POSSIBLE IMPROVEMENTS

- ▶ f-GANs
 1. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
 2. Precision-Recall Divergence Optimization for Generative Modeling with GANs and Normalizing Flows
- ▶ WGAN
 1. Wasserstein GAN
- ▶ Rejection Sampling
 1. Discriminator Rejection Sampling
 2. Metropolis-Hastings Generative Adversarial Networks
 3. Optimal Budgeted Rejection Sampling for Generative Models
- ▶ Latent Rejection sampling
 1. Latent reweighting, an almost free improvement for GANs
- ▶ Gradient ascent
 1. Discriminator optimal transport
 2. Refining Deep Generative Models via Discriminator Gradient Flow
 3. Your GAN is Secretly an Energy-based Model and You Should use Discriminator Driven Latent Sampling
- ▶ Classifier guidance generation
 1. MMGAN: Generative Adversarial Networks for Multi-Modal Distributions

FAIR COMPARISON

IMPORTANT DETAILS

You will have to compare your method with:

- ▶ The original GAN implementation given in the repository.
- ▶ The Hard Truncation method.
- ▶ The Soft Truncation method.
- ▶ A paper of your choice (that is not already chosen by your class mates).

One constraint for every one:

- ▶ You **HAVE TO USE THE SAME GENERATOR ARCHITECTURE** as the one given in the repository.

You will have to use the following metrics:

- ▶ FID
- ▶ Precision and Recall

The code is evaluated on:

- ▶ <https://www.lamsade.dauphine.fr/testplatform/iasdapp/results/last.html>

PROJECT PRESENTATIONS

FINAL PROJECT: PRESENTATION, CODE AND REPORT

Specifics:

- ▶ For fair comparison, use the same *generator* architecture as the one in the project.
- ▶ Your code will be pulled and test every 24 hours (at 02:00 AM).
- ▶ You have to choose one of the methods (or any other method that makes sense to you) and implement it.

During the presentation, you will have to:

- ▶ Present the paper you chose.
- ▶ Present how you adapted the method to the project.
- ▶ Explain your results.
- ▶ Explain the limitations of your method.

You will be graded:

- ▶ on the quality of the presentation.
- ▶ on how well you understood the method.
- ▶ on how well you adapted the method to the project.

You will not be graded on:

- ▶ Your rank on the platform.
- ▶ The quality of your code.

PROJECT PRESENTATIONS

REPORT AND PRESENTATION

The work is:

- ▶ Groups of 2 students.
- ▶ Due on the 16th of March at 14:00 PM.

The report and the presentation:

- ▶ The report should be a PDF file to push on the Github repository.
- ▶ The report should be named `report.pdf`. (2pts penalty if not respected)
- ▶ The report should be at most 5 pages long. (excluding references and title page)
- ▶ The report should be written in English.
- ▶ The presentation should be a PDF file to push on the Github repository.
- ▶ The presentation should be named `slides.pdf`. (2pts penalty if not respected)
- ▶ The presentation must be 10 minutes long (hard limit) + 2 minutes for questions.
- ▶ The presentation should be in English or French.

WHAT IS SLURM?



- ▶ Open-source workload manager for high-performance clusters.
- ▶ Schedules and queues jobs; matches jobs to available nodes.
- ▶ Allocates requested resources (CPU, GPU, memory) reliably.
- ▶ Provides monitoring and accounting tools (e.g., `squeue`, `sacct`).

TRADITIONAL SLURM WORKFLOW

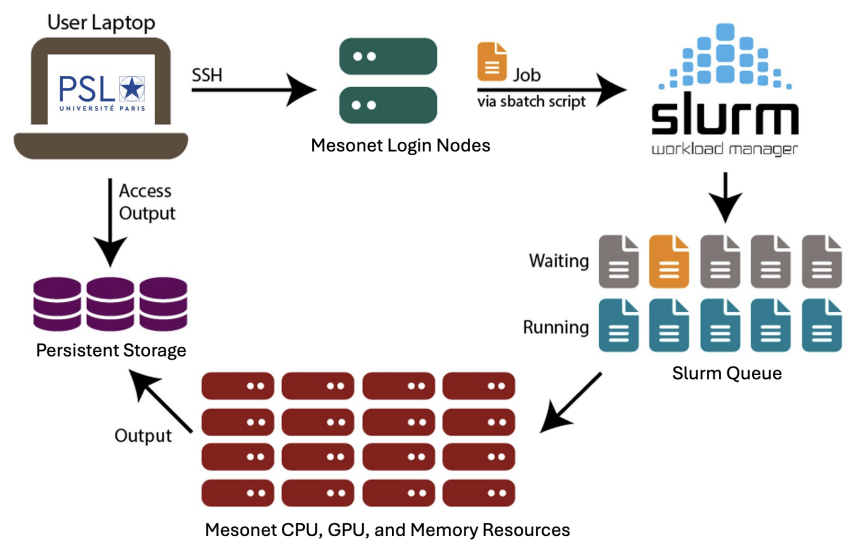
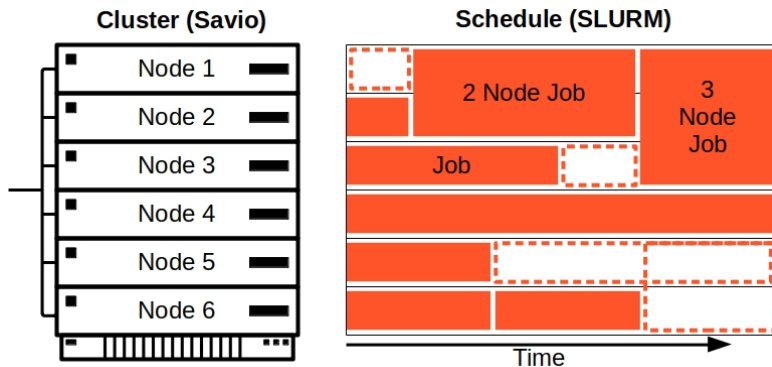


Figure. Typical SLURM job submission and execution workflow.

SLURM BACKFILL

- ▶ Jobs at the top of the queue have highest priority.
- ▶ Slurm can run lower-priority jobs **without delaying higher-priority jobs**.
- ▶ Helps smaller jobs run and prevents large jobs from blocking the system.
- ▶ Especially useful for multi-node job scheduling.



SUBMITTING A SLURM JOB

```
name="GAN_TRAINING"
outdir="outputs"

echo "Launching job $name"

sbatch <<EOT
#!/bin/bash
#SBATCH -p mesonet
#SBATCH -N 1
#SBATCH -c 28
#SBATCH --gres=gpu:${n_gpu}
#SBATCH --time=00:20:00
#SBATCH --mem=256G
#SBATCH --account=m25146
#SBATCH --job-name=${name}
#SBATCH --cpus-per-task=4
#SBATCH --output=$outdir/path.txt

source activate venv/bin/activate
python my_program.py
EOT
```


BASIC COMMANDS

Submit a job:

```
sbatch myscript.sh
```

```
chmod +x myscript.sh  
./myscript.sh
```

Check your jobs:

```
squeue -u $USER
```

Cancel a job:

```
scancel JOB_ID
```

CONNECTING TO MESONET

- ▶ Create an account on MesoNET:
`https://www.mesonet.fr/documentation/user-documentation/acces/portail`
- ▶ Use EduGain
- ▶ Mention that you are a student in the **IASD** course.
- ▶ You will receive an email with your MesoNET ID.
- ▶ Once you have received the confirmation email of your account, you can connect to MesoNET.
- ▶ Fill your account information in the MesoNET portal.
- ▶ Once you have filled your account information, you can send a mail to
`alexandre.verine@dauphine.psl.eu`
- ▶ You will be added to the MesoNET group for the IASD course.

GENERATING AN SSH KEY FOR MESONET

Step 1: Generate a new SSH key

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

- ▶ Press Enter to accept the default file location. (.ssh/id_ed25519)
- ▶ Enter a passphrase if you want extra security (optional).

Step 2: Copy the public key

```
cat .ssh/id_ed25519.pub
```

This prints your public key in the terminal.

Step 3: Add your key to MesoNET

1. Log in to your MesoNET account.
2. Go to the **SSH Keys** section and paste the public key you copied from the terminal.
3. Save your changes.

CONNECTING TO JULIET (MESONET)

Prerequisites:

- ▶ A valid **MesoNET** account
- ▶ An **SSH** key associated with your account

Connect via SSH:

```
ssh username@juliet.mesonet.fr
```

Replace username with your MesoNET ID. **Optional SSH configuration:**

```
Host juliet
  Hostname juliet.mesonet.fr
  User username
  IdentityFile [PathToYourSSHKey]
  IdentitiesOnly yes
```

Replace username and [PathToYourSSHKey] with your information.

CONCLUSION

Good luck with your project!

REFERENCES I

- [1] Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic Interpolants: A Unifying Framework for Flows and Diffusions. *arXiv preprint*, 2023. URL <http://arxiv.org/abs/2303.08797>. arXiv: 2303.08797.
- [2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. In *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia,, December 2017*. URL <http://arxiv.org/abs/1701.07875>. arXiv: 1701.07875.
- [3] Kate Bibbings, Peter J. Harding, Ian D. Loram, Nicholas Combes, and Emma F. Hodson-Tole. Foreground Detection Analysis of Ultrasound Image Sequences Identifies Markers of Motor Neurone Disease across Diagnostically Relevant Skeletal Muscles. *Ultrasound in Medicine & Biology*, 45(5):1164–1175, May 2019. ISSN 1879-291X. doi: 10.1016/j.ultrasmedbio.2019.01.018.
- [4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis, February 2019. URL <http://arxiv.org/abs/1809.11096>. arXiv:1809.11096 [cs, stat].
- [5] Florian Le Bronnec, Alexandre Verine, Benjamin Negrevertgne, Yann Chevaleyre, and Alexandre Allauzen. Exploring Precision and Recall to assess the quality and diversity of LLMs. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, February 2024. URL <http://arxiv.org/abs/2402.10693>. arXiv:2402.10693 [cs].

REFERENCES II

- [6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. In *27th Conference on Neural Information Processing Systems (NeurIPS 2014)*, June 2014. URL <http://arxiv.org/abs/1406.2661>. arXiv: 1406.2661.
- [7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs, December 2017. URL <http://arxiv.org/abs/1704.00028>. arXiv:1704.00028 [cs].
- [8] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK. 2017.
- [9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. arXiv, December 2020. URL <http://arxiv.org/abs/2006.11239>. arXiv:2006.11239, NeurIPS 2020.
- [10] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation, February 2018. URL <http://arxiv.org/abs/1710.10196>. arXiv:1710.10196 [cs, stat].

REFERENCES III

- [11] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks, March 2019. URL <http://arxiv.org/abs/1812.04948>. arXiv:1812.04948 [cs, stat].
- [12] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN, March 2020. URL <http://arxiv.org/abs/1912.04958>. arXiv:1912.04958 [cs, eess, stat].
- [13] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models, October 2022. URL <http://arxiv.org/abs/2206.00364>. arXiv:2206.00364, NeurIPS 2022.
- [14] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022. URL <http://arxiv.org/abs/1312.6114>. arXiv:1312.6114 [cs, stat].
- [15] Durk P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montréal, Canada., volume 31, 2018.
- [16] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved Precision and Recall Metric for Assessing Generative Models. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada., October 2019. arXiv: 1904.06991.

REFERENCES IV

- [17] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. In *International Conference on Learning Representations (ICLR)*, 2023. URL <https://openreview.net/forum?id=PqvMRDCJT9t>. arXiv: 2210.02747.
- [18] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. In *International Conference on Learning Representations (ICLR)*, 2023. URL <https://openreview.net/forum?id=XVjTT1nw5z>. arXiv: 2209.03003.
- [19] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks, February 2018. URL <http://arxiv.org/abs/1802.05957>. arXiv:1802.05957 [cs, stat].
- [20] XuanLong Nguyen, Martin J. Wainwright, and Michael I. Jordan. On surrogate loss functions and f -divergences. *The Annals of Statistics*, 37(2), April 2009. ISSN 0090-5364. doi: 10.1214/08-AOS595. URL <http://arxiv.org/abs/math/0510521>. arXiv:math/0510521.
- [21] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization, June 2016. URL <http://arxiv.org/abs/1606.00709>. arXiv:1606.00709 [cs, stat].
- [22] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural Discrete Representation Learning, May 2018. URL <http://arxiv.org/abs/1711.00937>. arXiv:1711.00937 [cs].

REFERENCES V

- [23] Aram-Alexandre Pooladian, Heli Ben-Hamu, Carles Domingo-Enrich, Brandon Amos, Yaron Lipman, and Ricky T. Q. Chen. Multisample Flow Matching: Straightening Flows with Minibatch Couplings. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. URL <https://proceedings.mlr.press/v202/pooladian23a.html>. arXiv: 2304.14772.
- [24] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, January 2016. URL <http://arxiv.org/abs/1511.06434>. arXiv:1511.06434 [cs].
- [25] Axel Sauer, Katja Schwarz, and Andreas Geiger. StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, pages 1–10, Vancouver BC Canada, August 2022. ACM. ISBN 978-1-4503-9337-9. doi: 10.1145/3528233.3530738. URL <https://dl.acm.org/doi/10.1145/3528233.3530738>.
- [26] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://papers.nips.cc/paper_files/paper/2015/hash/8d55a249e6baa5c06772297520da2051-Abstract.html.

REFERENCES VI

- [27] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. SCORE-BASED GENERATIVE MODELING THROUGH STOCHASTIC DIFFERENTIAL EQUATIONS. 2021.
- [28] Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrod Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport. In *Transactions on Machine Learning Research*, 2024. URL <https://openreview.net/forum?id=CD9Snc73AW>. arXiv: 2302.00482.
- [29] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs*, 2, 2010. URL <http://yann.lecun.com/exdb/mnist>.