GENERATIVE MODELS

FROM GAUSSIAN SAMPLING TO DIFFUSION MODELS

Alexandre Vérine,
Research Fellow, ENS-PSL
Université PSL

IASD
Université Dauphine-PSL

September 21, 2025

WHO AM I?

- Alexandre Vérine
- Research Fellow, ENS-PSL
- Université PSL
- Previously: PhD student at Dauphine (2021-2024)
- Research interests: generative models, evaluation methods, quality diversity trade-off
- All my slides and code are available on `www.alexverine.com`

About this course

Outline of the course:

- Lecture 1 (22/09, 9h–12h15): From sampling to the first generative model
- Practical 1 (29/09, 9h–12h15): Building and training a VAE
- Lecture 2 (03/11, 9h–12h15): GANs (from the first GAN to f-GAN, WGAN, discriminator rejection sampling)
- Practical 2 (08/12, 9h–12h15): Training a GAN
- Lecture 3 (15/12, 9h–12h15): Diffusion Models (DDPM to EDM, score-based and classifier guidance)
- Practical 3 (26/01, 9h–12h15): Comparing ODE and SDE in small dimensions
- Lecture 4 (02/02, 9h–12h15): Evaluating generative models (IS, Precision and Recall) + Project presentation
- Last session (10/02, 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 (10/02/2024). 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.

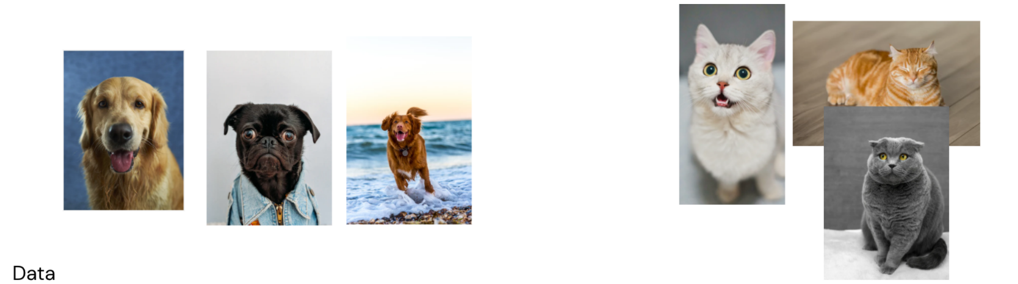
AI 101: From Fundamentals to Deep Learning

Part I

Introduction to Generative Models

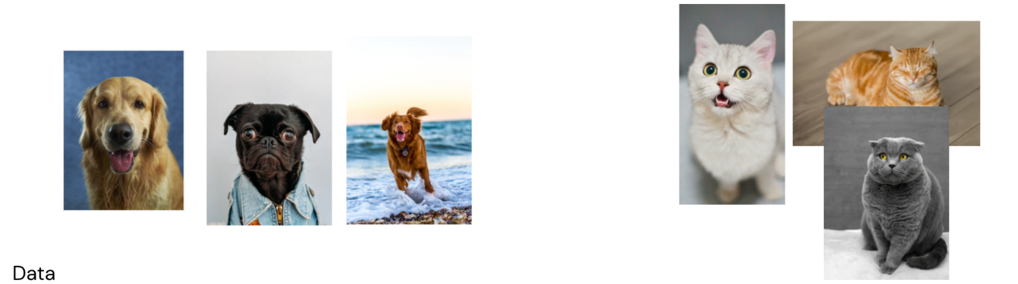
WHAT IS SAMPLING?
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P:



What is Sampling?
Objective of Sampling Images

Images distributed under a given distribution P:



What constraints should a generated image satisfy?

WHAT IS SAMPLING?
OBJECTIVE OF SAMPLING IMAGES

Images distributed under a given distribution P:

What constraints should a generated image satisfy?

What is Sampling?

Objective of Sampling Images

Images distributed under a given distribution P:

How do we ensure resemblance to real images?

What is Sampling?
Objective of Sampling Images

Images distributed under a given distribution P:

How do we ensure resemblance to real images?

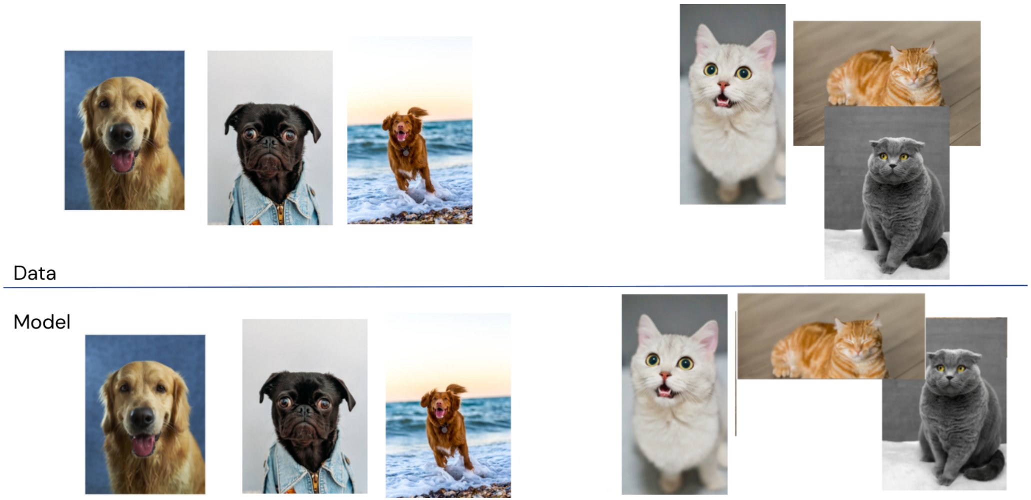
What is Sampling?
Objective of Sampling Images

Images distributed under a given distribution P:

How to cover the diversity of the dataset?

What is Sampling?
Objective of Sampling Images

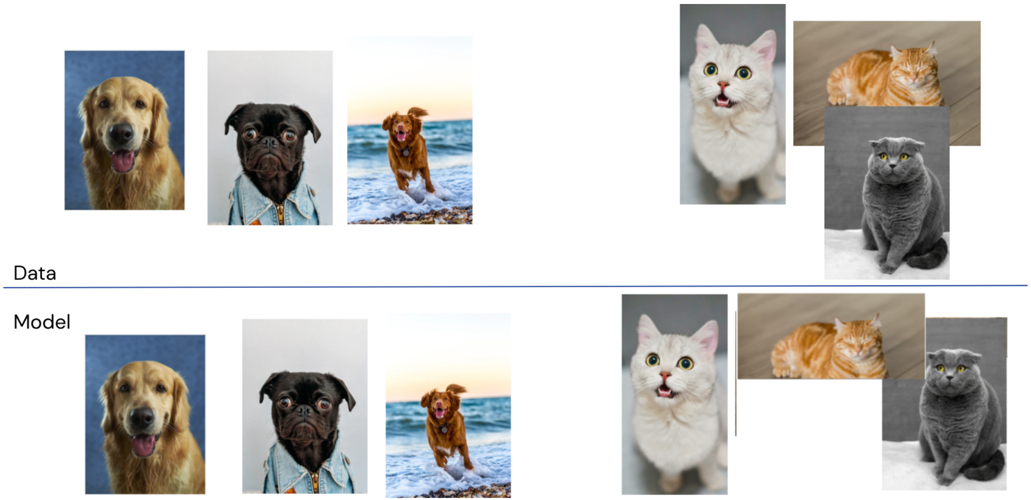
Images distributed under a given distribution P:



How to cover the diversity of the dataset?

What is Sampling?
Objective of Sampling Images

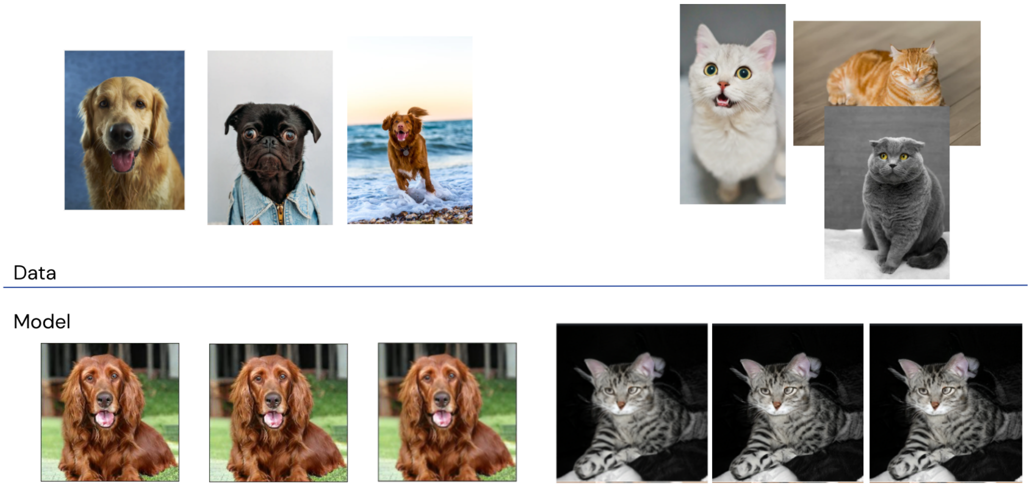
Images distributed under a given distribution P:



What about likelihood and novelty?

What is Sampling?
Objective of Sampling Images

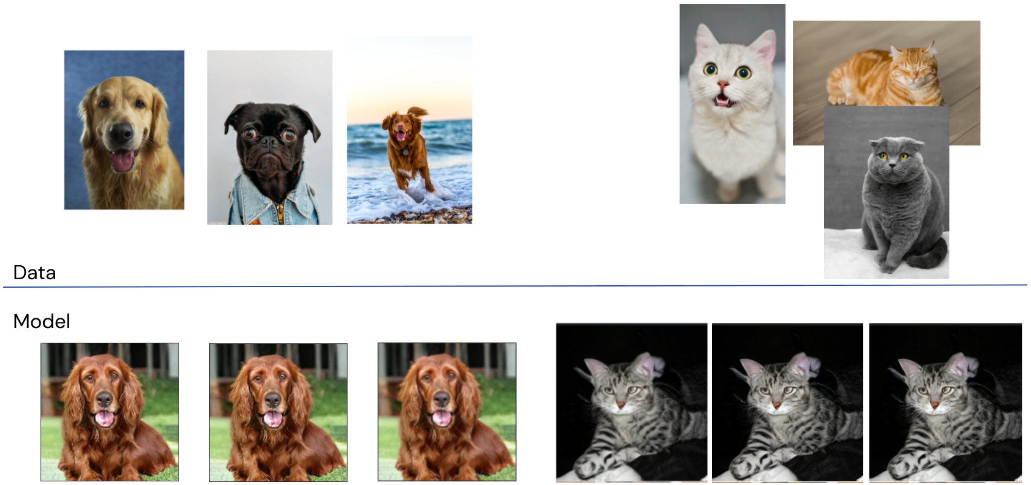
Images distributed under a given distribution P:



What about likelihood and novelty?

What is Sampling?
Objective of Sampling Images

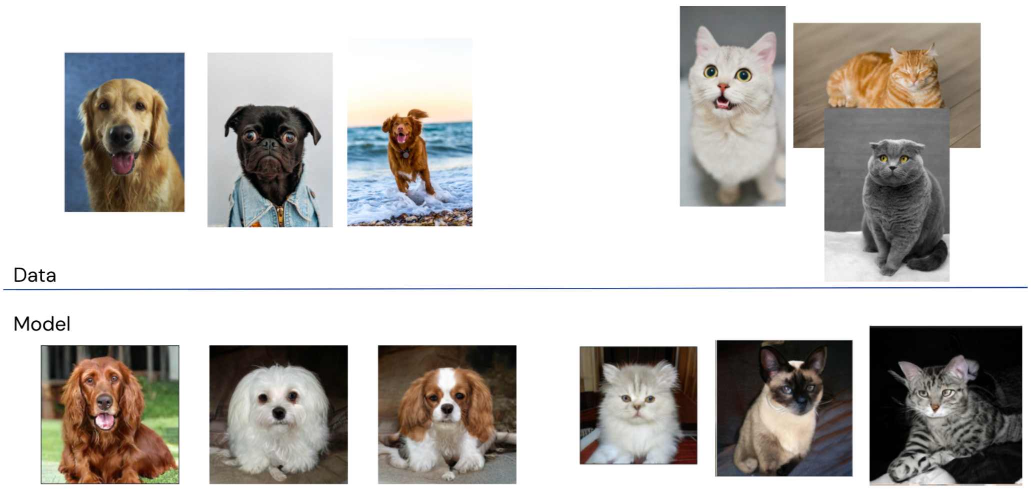
Images distributed under a given distribution P:



How to incorporate stochasticity in generation?

What is Sampling?
Objective of Sampling Images

Images distributed under a given distribution P:



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 colorized 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.

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):	**Output (French):**
A scenic view of a mountain during sunset.	Une vue pittoresque d'une montagne au coucher du soleil.
A bustling city street at night.	Une rue animée de la ville la nuit.
A serene beach with palm trees.	Une plage sereine avec des palmiers.

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

Example: Text-to-Text Math Problem Solving

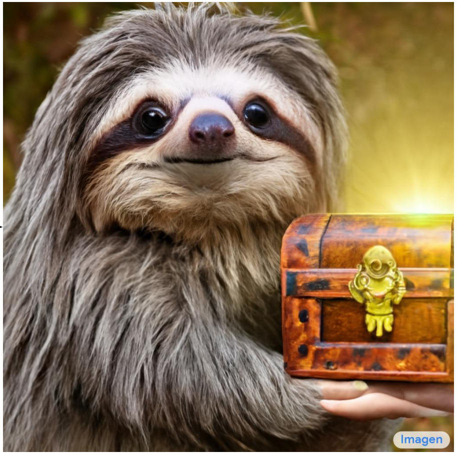
Input (Problem):	**Output (Solution):**
If a car travels at 60 mph for 2 hours, how far does it go?	The car travels 120 miles.
What is the derivative of $x^2 + 3x + 2$?	The derivative is $2x + 3$.
Solve for x: $2x + 5 = 15$.	The solution is $x = 5$.

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

What is Sampling?
How do we actually generate an image?

Prompt:
"A cute sloth holding a small treasure chest.
A bright golden glow is coming from the chest."

?



How should we generate something random corresponding to this?

What is Sampling?

How do we actually generate an image?

Prompt:
"A cute sloth holding a small treasure chest.
A bright golden glow is coming from the chest."

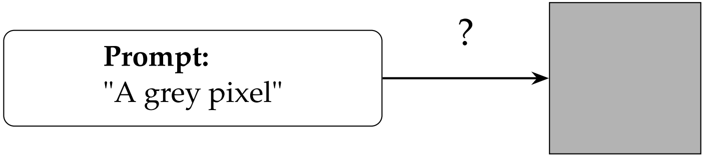
→ ?

How should we generate something random corresponding to this?

It's very complex.

From complex to simple

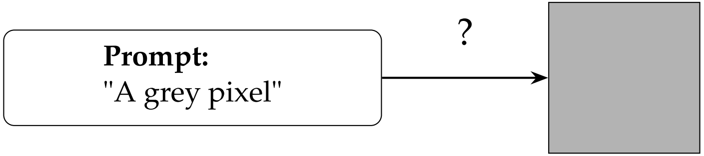
Simplifying the problem



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)

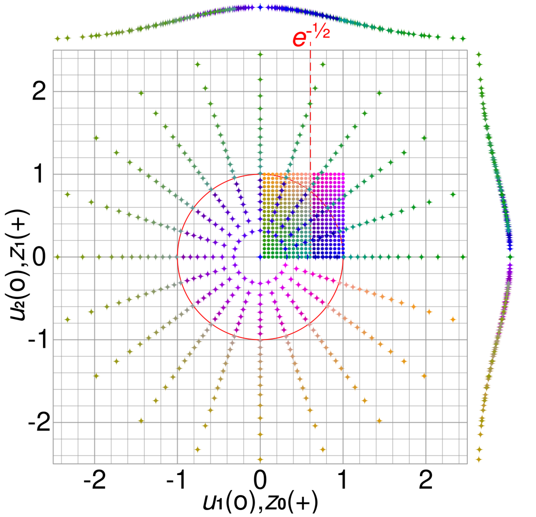
$$U \sim \text{Unif}(0,1) \stackrel{\text{Box–Muller}}{\Longrightarrow} Z \sim \mathcal{N}(0,1)$$

Sampling Transformations

Box–Muller (Uniform → 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.



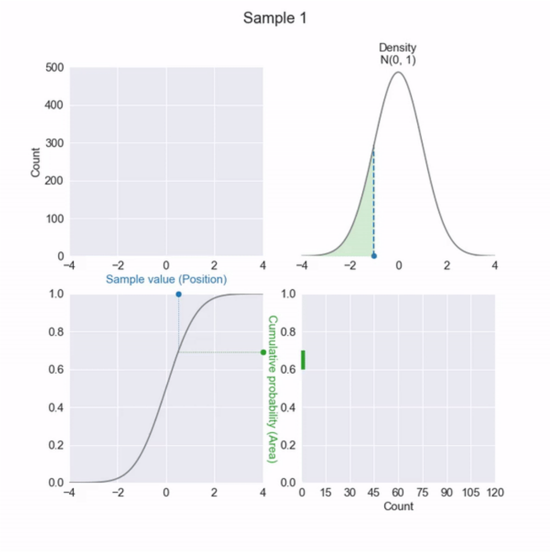
Algorithm 2: Box–Muller

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

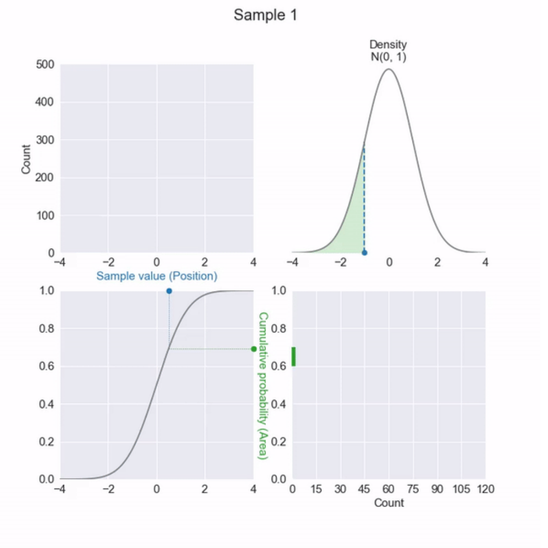
- If $U \sim \text{Unif}(0,1)$ and F is a CDF, then $X = F^{-1}(U)$ has CDF F.
- Requires F to be strictly increasing / invertible; otherwise use generalized inverse.
- In practice: precompute/discretize F, interpolate F^{-1}, or use spline/numerical root-finding.



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 $p(x_1, \ldots, x_N)$.
Special cases where it becomes easy(er):

- **Independence:** $p(x_1, \ldots, x_N) = \prod_{i=1}^{N} p_i(x_i)$ — sample each coordinate independently.
- **Markov (autoregressive) chain:** $p(x_{1:N}) = p(x_1) \prod_{i=2}^{N} p(x_i \mid x_{i-1})$ — 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 $\sigma > 0$,

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

Parameters:

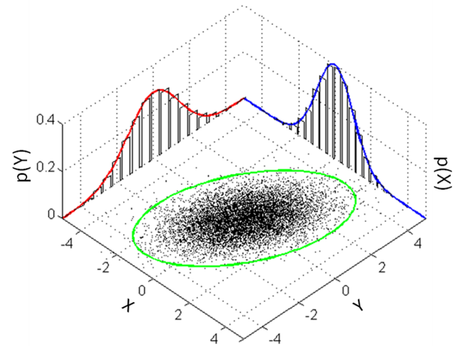
- μ (location): shifts the center of mass of the density.
- σ (scale): controls spread (variance σ^2).

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

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

Parameters:

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



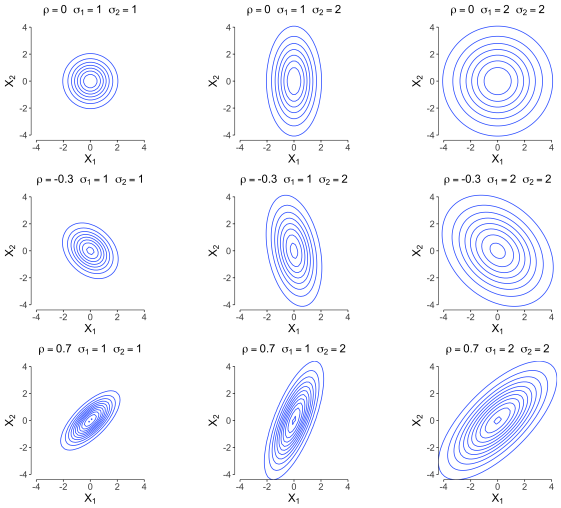
Elliptical contours illustrating Σ in $\mathbb{R}^d$.

From Univariate to Multivariate Gaussian

Covariance types

Special cases.

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



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

From $\mathcal{N}(0, \mathbf{I})$ to $\mathcal{N}(\mu, \Sigma)$

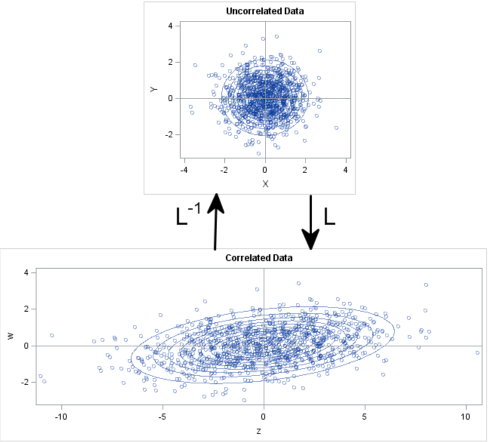
Cholesky Factorization and Sampling

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 $x \leftarrow \mu + Lz \quad \Rightarrow$ then $x \sim \mathcal{N}(\mu, \Sigma)$.

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



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

Gaussian Mixtures

Definition and Intuition

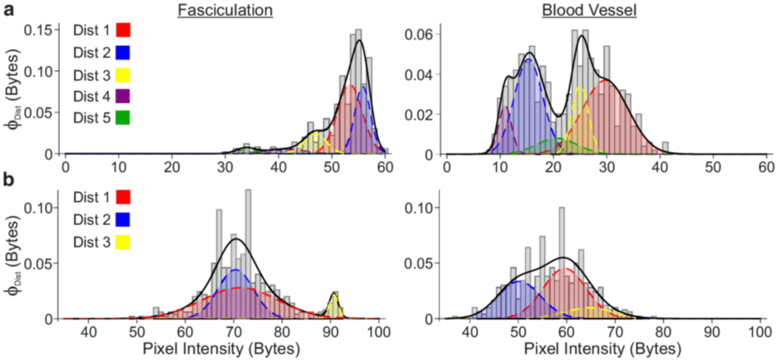
Motivation. To increase the model complexity beyond a single Gaussian, we can *mix* several Gaussians; this yields a flexible, multi-modal density.
Definition. A Gaussian Mixture Model (GMM) with K components on $\mathbb{R}^d$ is

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

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

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

Example from [1]:



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 $K \sim \text{Categorical}(\pi_1, \ldots, \pi_K)$
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:**

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

- **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 $\{x_n\}_{n=1}^{N}$):

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

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

- 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:

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

Training Gaussian Mixtures
Gradient Descent Approach

Parameterization for constraints:

- Weights: raw logits w_k with $\pi_k = \text{softmax}(w)_k$.
- Covariance: diagonal with $\sigma^2_{k,j} = \exp(\alpha_{k,j})$; 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)
$J \leftarrow -\frac{1}{|\mathcal{B}|} \sum_b \ell_b$

`// 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).

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

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

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

- 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
```

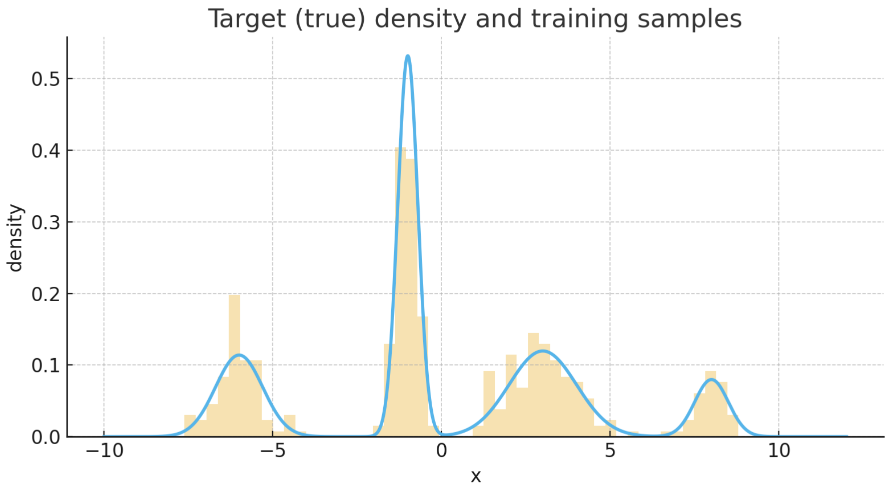
  for $n = 1$ **to** N **do**
    for $k = 1$ **to** K **do**
      $r_{nk} \leftarrow \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$
    end
    $\gamma_{nk} \leftarrow r_{nk} / \sum_{j=1}^{K} r_{nj}$
  end

```
    // M-step: parameter updates
```

  $N_k \leftarrow \sum_{n=1}^{N} \gamma_{nk}$ for $k = 1, \ldots, K$
  $\pi_k \leftarrow N_k / N$
  $\mu_k \leftarrow \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} x_n$
  $\Sigma_k \leftarrow \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^\top + \epsilon \mathbf{I}$

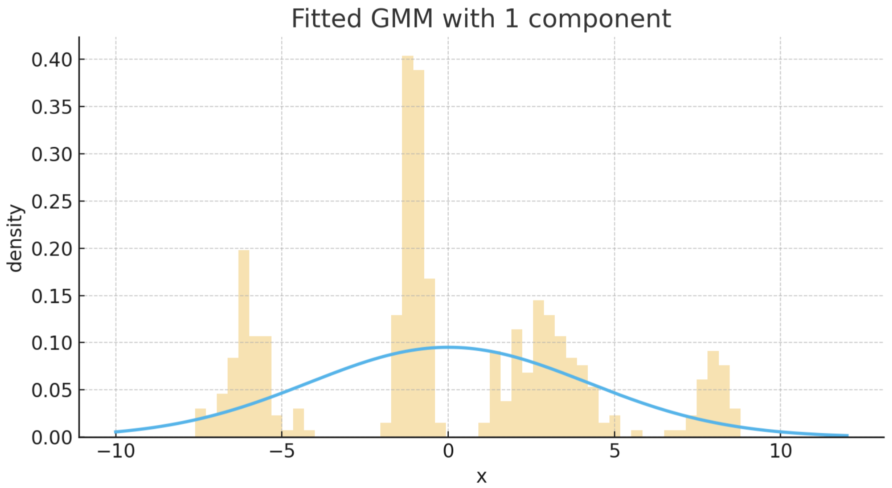
until *convergence*

Gaussian Mixtures
Example



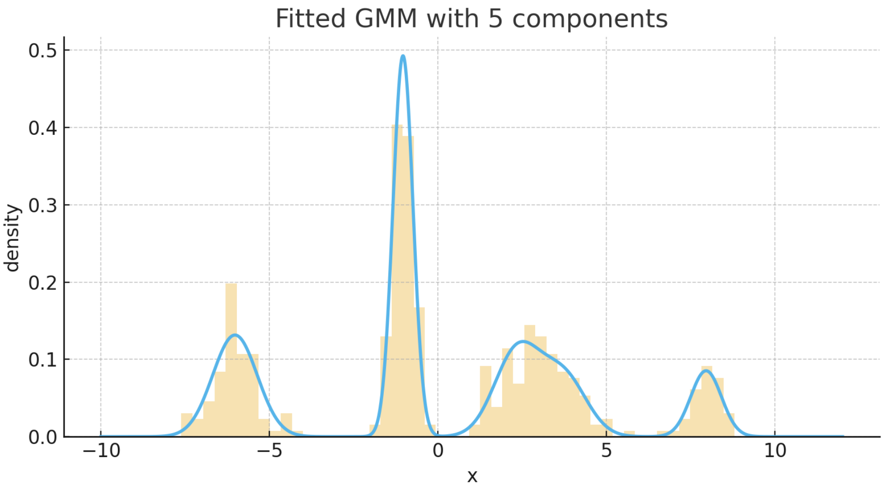
Fitting a GMM to 1D data points. In blue the density, in orange the histogram of data points.

Gaussian Mixtures
Example



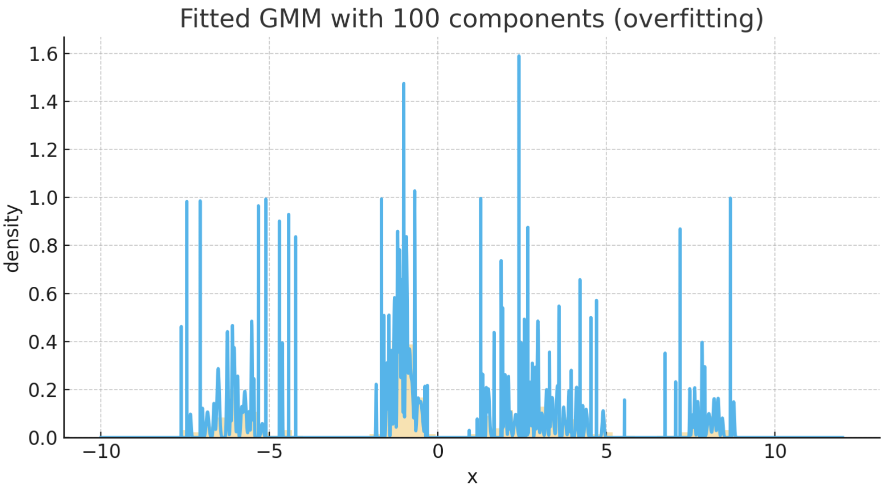
Fitting a GMM to 1D data points. $K = 1$.

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 $\rightarrow$ 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.

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 $\widehat{P}_\theta$

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

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

Types of generative models.

- **Explicit models:** provide a tractable density $\widehat{p}_\theta(x)$ (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 $\widehat{p}_\theta(x)$ (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 $F : \mathcal{X} \to \mathcal{Z}$ 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:

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

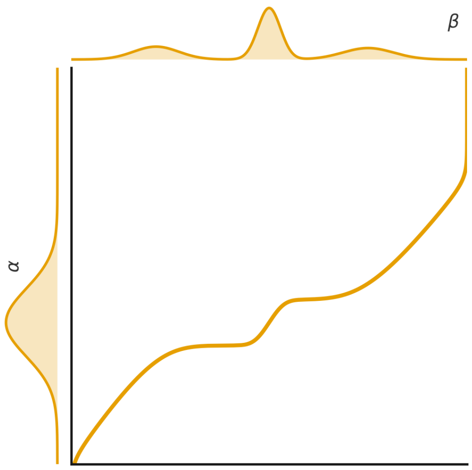
where $\mathrm{Jac}_F(x)$ is the Jacobian matrix of F at x.

Conditions:

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

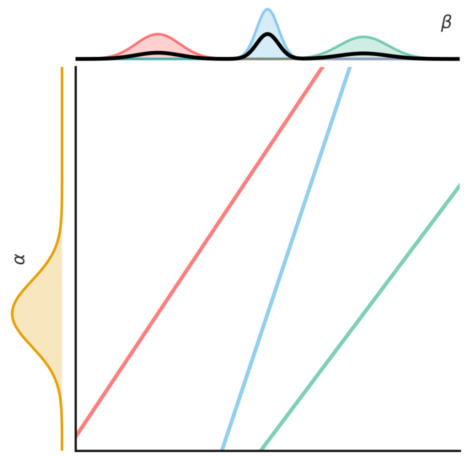
Implicit vs Explicit Models

Change of Variables Formula



Deterministic Monge transport (T = Fβ⁻¹∘Fα)

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 $F : \mathcal{Z} \to \mathcal{X}$ 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.

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{D}\big(P \,\|\, \widehat{P}_\theta\big).$$

Examples next: KL, Total Variation, Wasserstein.

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

DIVERGENCES

KULLBACK–LEIBLER (FORWARD KL)

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

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

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

DIVERGENCES
TOTAL VARIATION (TV)

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

- 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 $\Pi(P, Q)$,

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

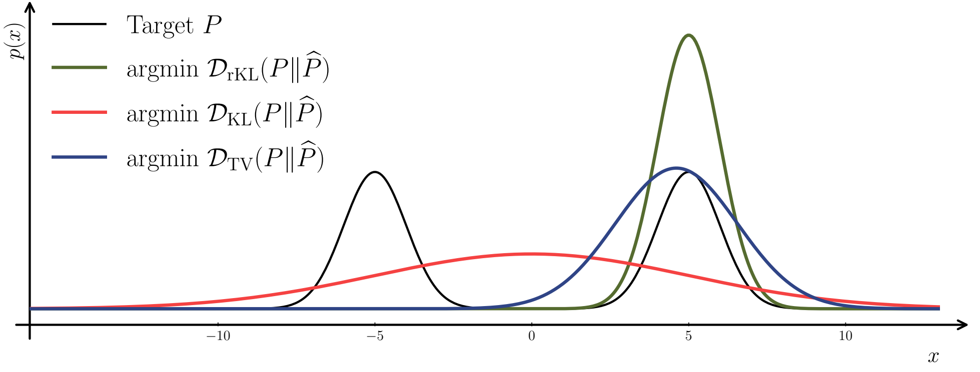
- Sensitive to the geometry of $\mathcal{X}$; finite even with disjoint supports.
- Dual (Kantorovich–Rubinstein): $\mathcal{W}_1 = \sup_{\|f\|_{\text{Lip}} \leq 1} \big(\mathbb{E}_P[f] - \mathbb{E}_Q[f]\big)$.
- Basis for WGAN objectives.

Divergences

KL vs Reverse KL

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

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

Therefore,

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

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

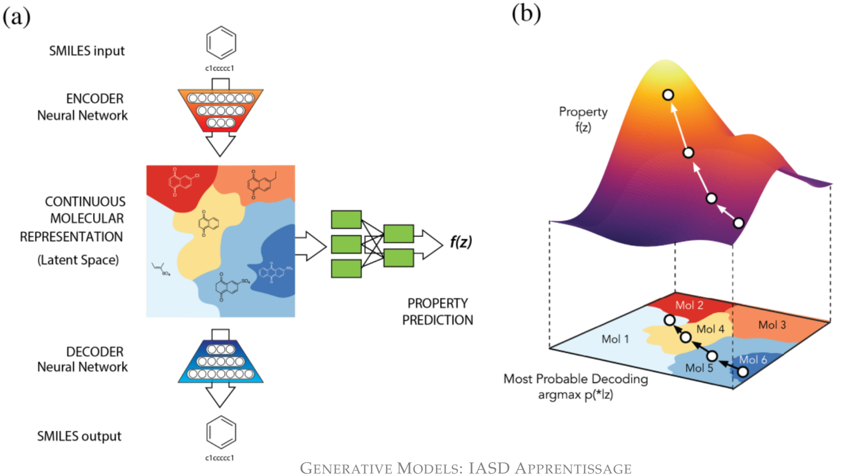
$$\theta^* \;=\; \arg\max_\theta\; \mathbb{E}_P[\log q_\theta(X)] \;\equiv\; \arg\min_\theta\; \mathcal{D}_{\mathrm{KL}}\big(P\|Q_\theta\big).$$

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 $x \sim p_\theta(x \mid z)$ (decoder/model).
- **Inference:** approximate the posterior with $q_\phi(z \mid x)$ (encoder) since $p_\theta(z \mid x)$ 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 $q_\phi(z \mid x)$,

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

Define the **ELBO**:

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

How to Compute a Divergence?

Approximate MLE and ELBO

ELBO in KL form.

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

And the exact evidence decomposes as

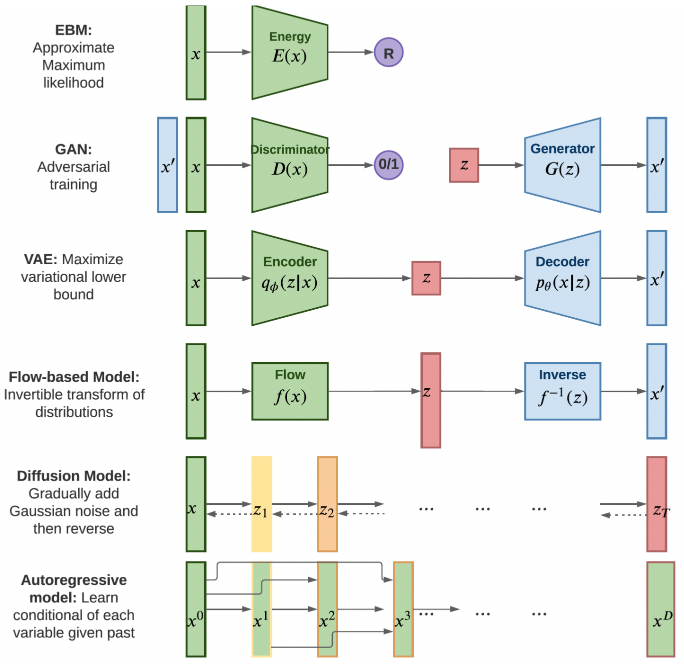
$$\log p_\theta(x) \;\geq\; \mathrm{ELBO}(\theta, \phi; x) \; - \; \mathcal{D}_{\mathrm{KL}}\big(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)\big),$$

so maximizing ELBO *minimizes* the posterior KL.

What to optimize:

- **w.r.t. θ (decoder/model):** increase the reconstruction term $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$; use Monte Carlo gradients with reparameterized z.
- **w.r.t. ϕ (encoder/inference):** tighten the bound by driving $q_\phi(z \mid x)$ toward $p_\theta(z \mid x)$ (reducing the posterior KL above), while also respecting the regularizer $\mathcal{D}_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z))$.
- **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



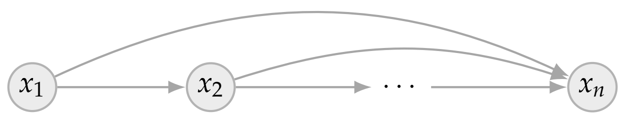
Model Landscape
At a Glance

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	$\mathbb{R}^m$	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

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

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

- 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.

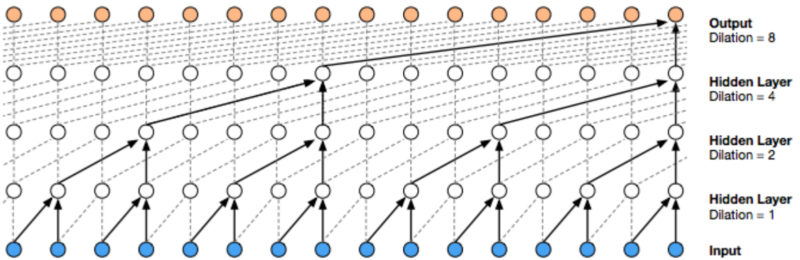


Autoregressive Models
Training Objective

Maximum likelihood = sum of conditional cross-entropies.

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

- Empirical objective: $-\frac{1}{N}\sum_n \sum_i \log p_\theta\big(x_{n,i} \mid x_{n,<i}\big)$.
- *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 $x = (x_1, \ldots, x_d)$
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).

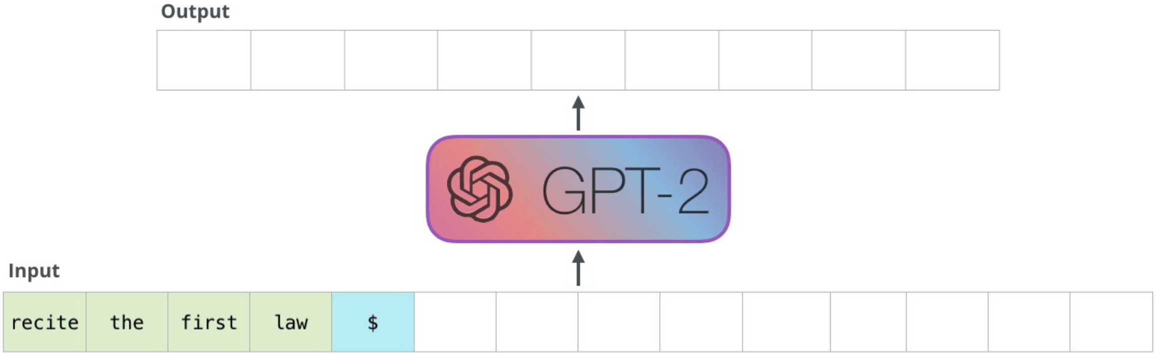
Autoregressive Models

LLMs as Autoregressive Next-Token Predictors

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

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

- 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.



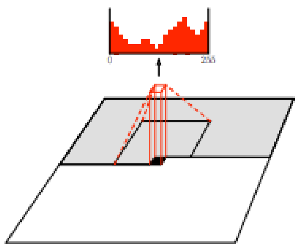
Autoregressive Models

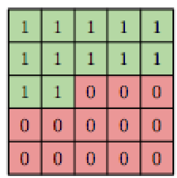
PixelCNN / PixelRNN

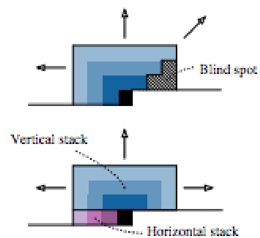
Image factorization. Raster-scan ordering over pixels (and optionally channels):

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

- **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

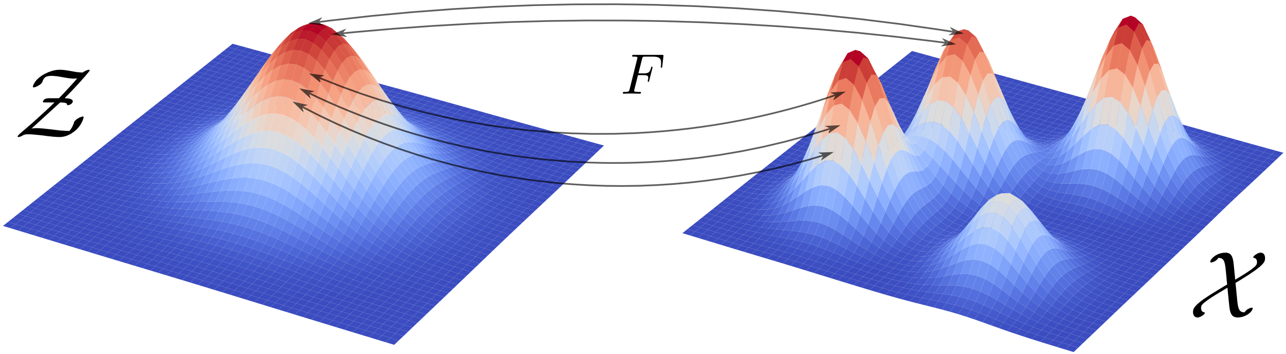


Figure. A mapping between two probability distributions
Point to point

Normalizing Flows

Mapping Between Distributions — Subset to Subset

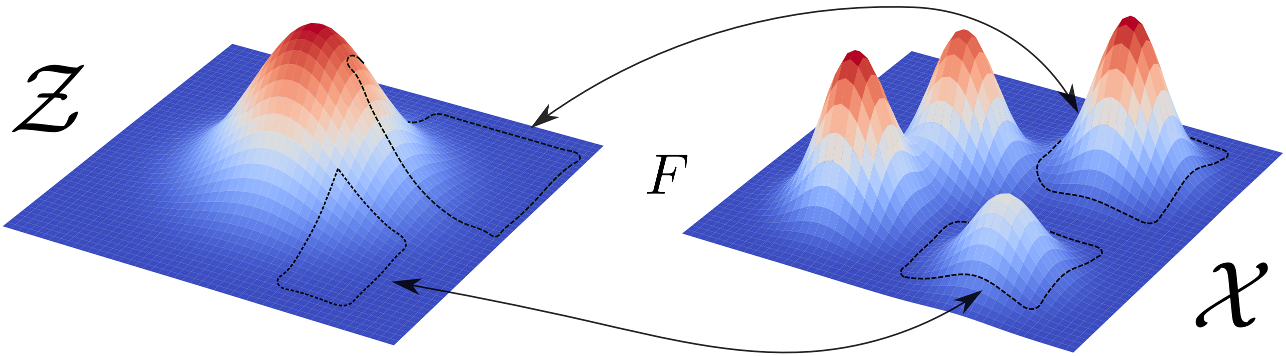


Figure. A mapping between two probability distributions
Subset to subset

Normalizing Flow

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

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

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 $\widehat{P}$ 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.

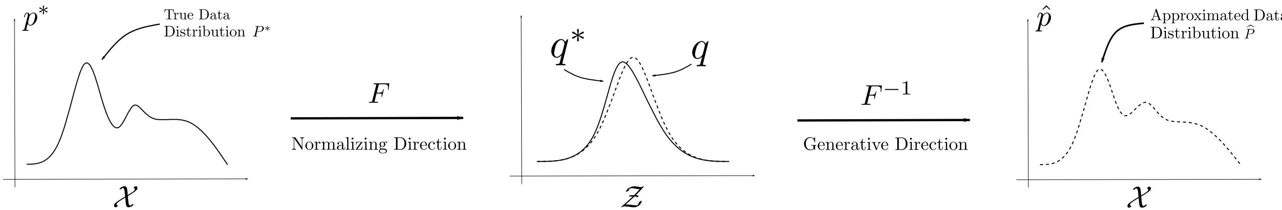


Figure. 1D Normalizing Flow process.

Change of Variables Formula

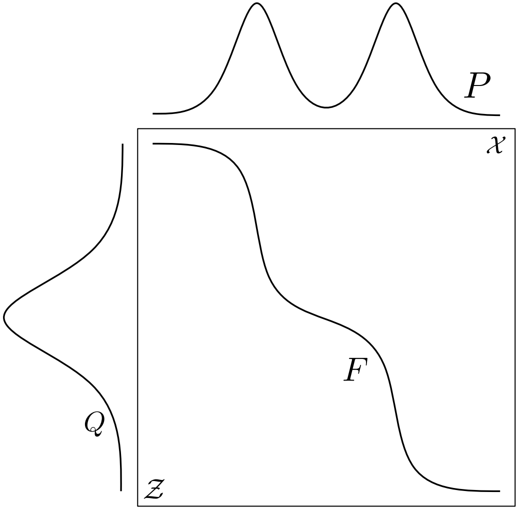
For a bijective and continuous function F and a latent distribution Q, the distribution induced by Q and F is defined by:

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

Normalizing Flows

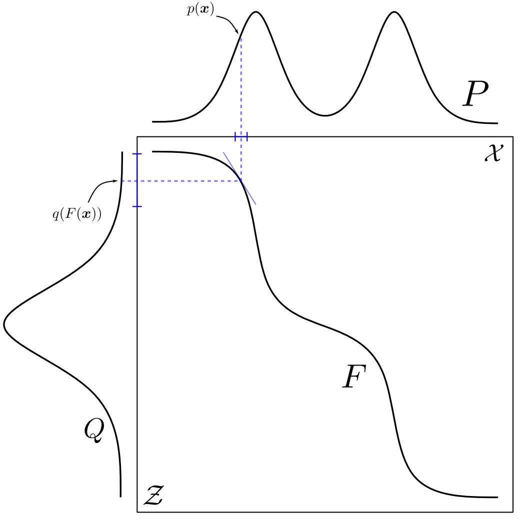
Change of Variables — Visual Intuition

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



Normalizing Flows
Change of Variables — Visual Intuition

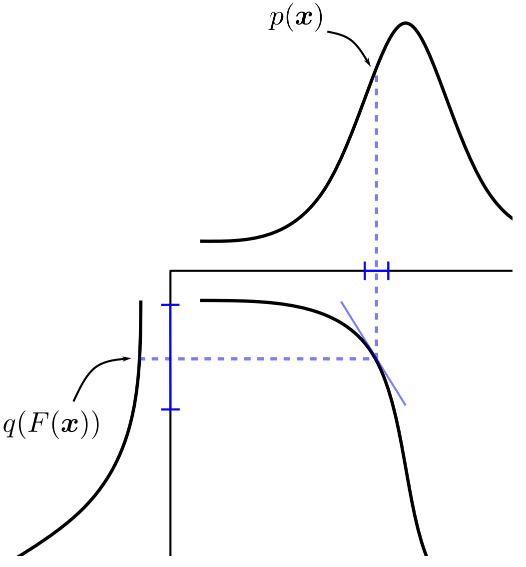
$$\forall x \in \mathcal{X}, \quad \hat{p}(x) = \left| \det \mathrm{Jac}_F(x) \right| q(F(x)).$$



Normalizing Flows

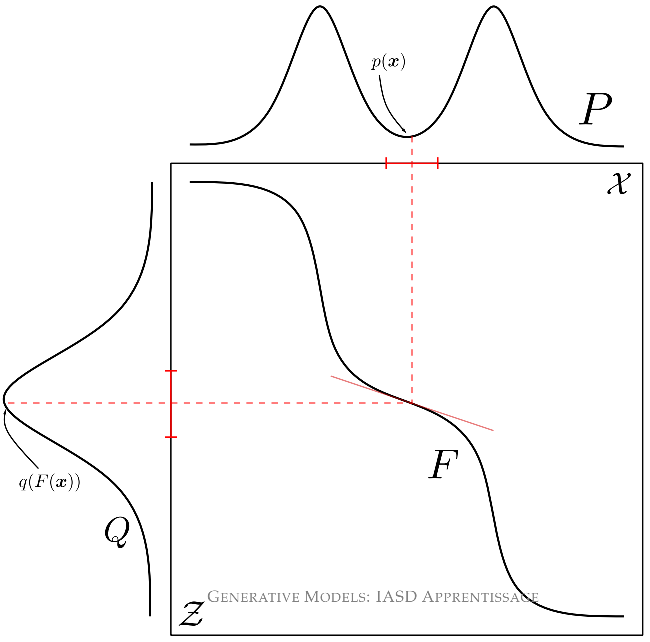
Change of Variables — Visual Intuition

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



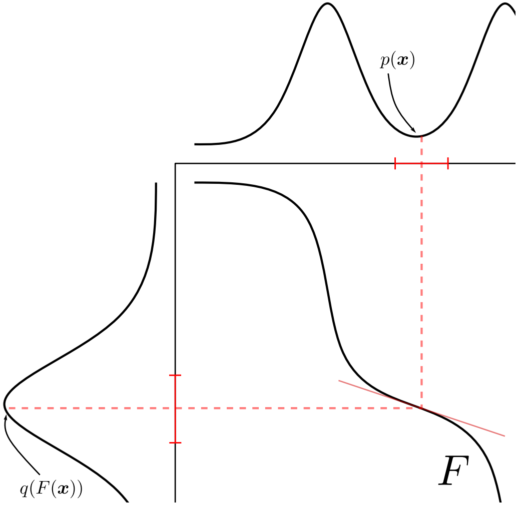
CHANGE OF VARIABLES — VISUAL INTUITION

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



Normalizing Flows
Change of Variables — Visual Intuition

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



Normalizing Flows

Density Estimation

To perform density estimation:

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

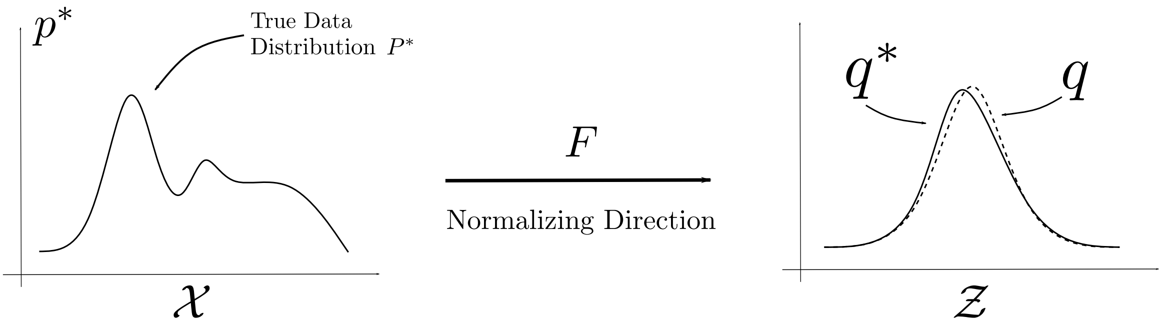


Figure. 1D Normalizing Flow process of density estimation.

To perform data generation:

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

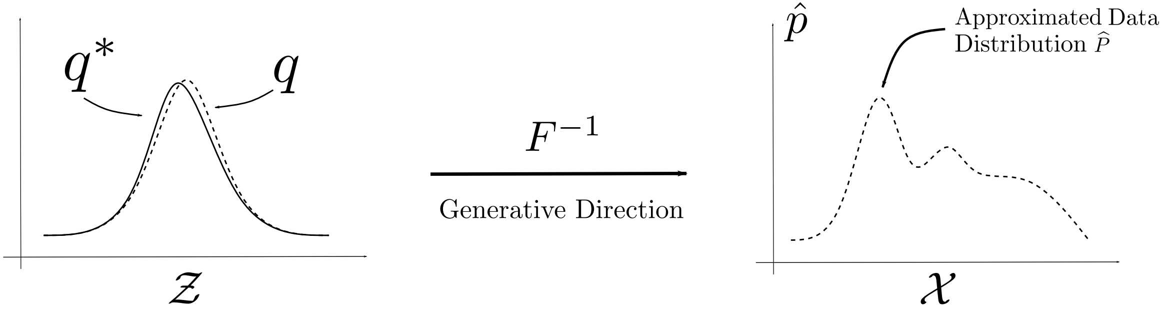


Figure. 1D Normalizing Flow process of generation.

Normalizing Flows
Learning Steps

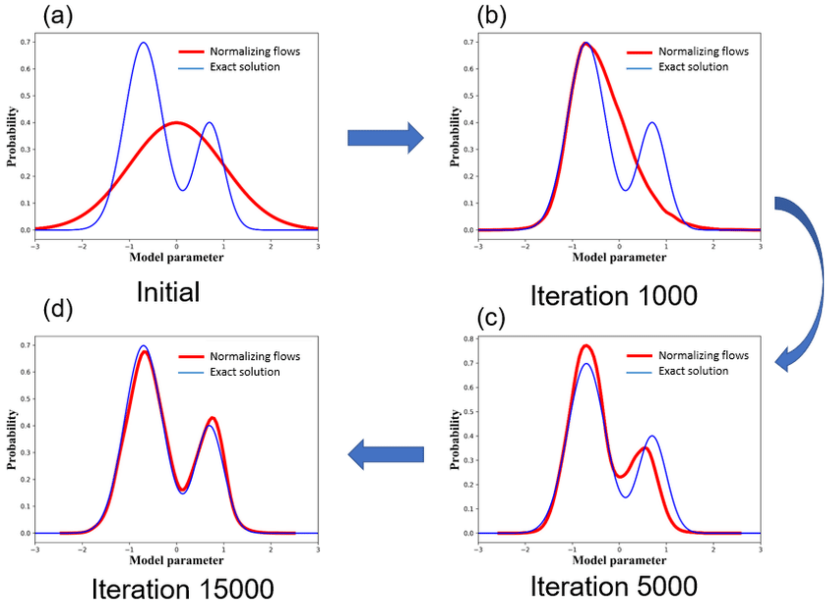


Figure. Learning process for a 1D Normalizing Flow.

Energy-Based Models
Partition Function Challenge

Definition. An Energy-Based Model (EBM) defines

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

- $E_\theta(x)$ 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.

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:

$$Z(\theta) = \int e^{-E_\theta(u)}\, du, \qquad \nabla_\theta Z(\theta) = \int e^{-E_\theta(u)} \big(-\nabla_\theta E_\theta(u)\big)\, du,$$

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

Substitute back:

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

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

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

Gradient step (schematic):

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

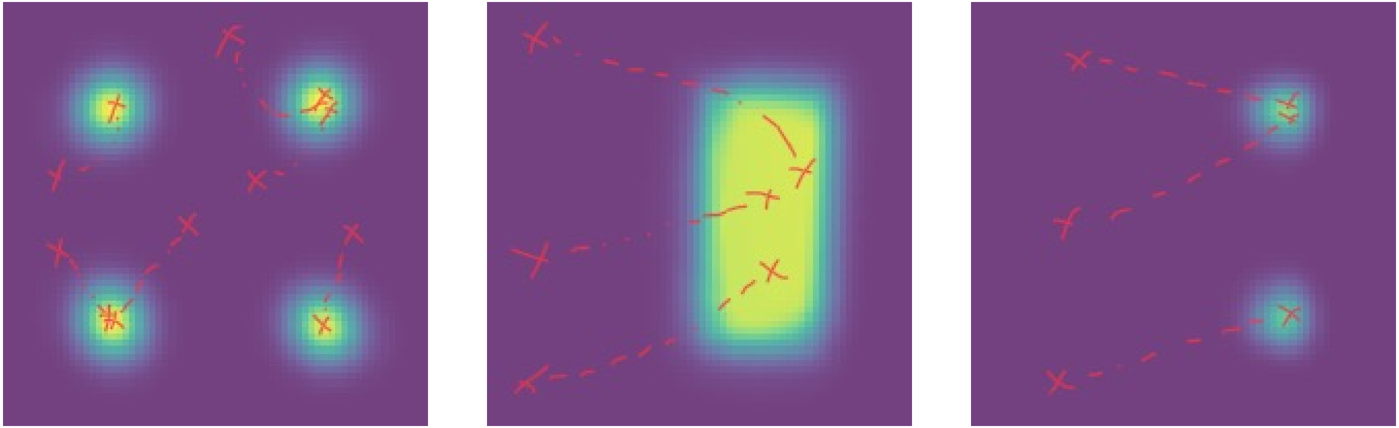
- *Pull down* energy on data (first term), *push up* on model samples (second term).
- Trade-offs: bias vs. mixing time; stability tricks (noise scale, step size, gradient clipping, spectral norm).

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:

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

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.

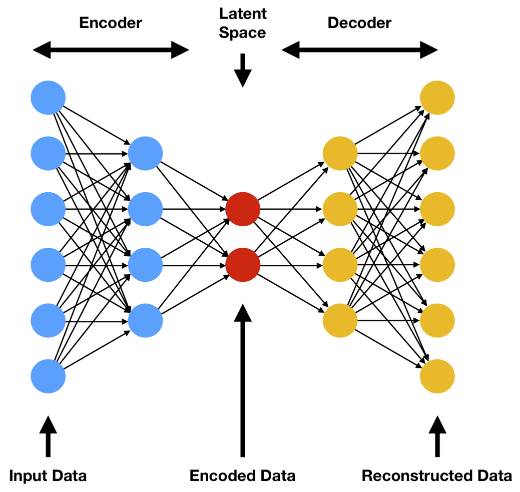
AUTOENCODER
ILLUSTRATION AND EXAMPLE

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

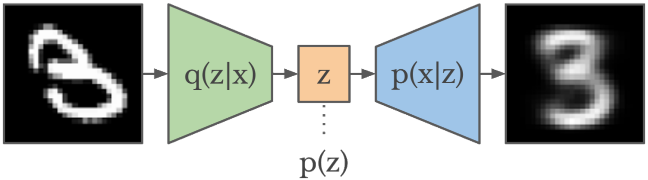


Variational Autoencoder
Intuition Behind VAE

Key ideas:

- **Probabilistic encoder:** Instead of mapping $x \to z$ 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 $\mathcal{D}_{\mathrm{KL}}(q_\phi(z|x)\|p(z))$ 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)

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

- **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:

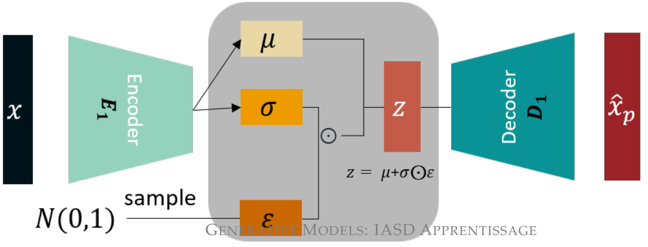
- For $q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x)^2)$,
- Sample z as:

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

- 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:

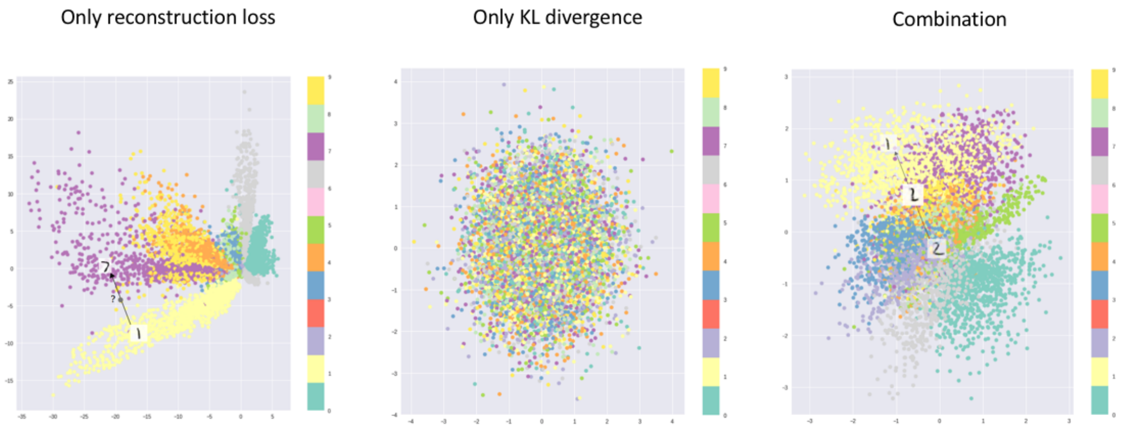
$$\mathcal{L}_{\text{VAE}}(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[-\log p_\theta(x|z)]}_{\text{Reconstruction loss}} + \underbrace{\mathcal{D}_{\text{KL}}(q_\phi(z|x)\|p(z))}_{\text{KL divergence (regularization)}}$$

- **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 [5].
- β**-VAE:** Introduces a hyperparameter β to scale the KL term, encouraging disentangled latent representations [2].
- **Other notable extensions:**
  - *VampPrior*: Learnable mixture prior for more flexible latent space.
  - *Vector Quantized VAE (VQ-VAE)* [4]: Discrete latent variables via vector quantization.
  - *Hierarchical VAE*: Multiple layers of latent variables.
  - *FactorVAE*, *InfoVAE*, *WAE* (Wasserstein Autoencoder), etc.

References:

- Original VAE paper: [3]
- Conditional VAE: [5]
- β-VAE : [2]
- VQ-VAE: [4]

VQ-VAE (van den Oord et al., 2017)

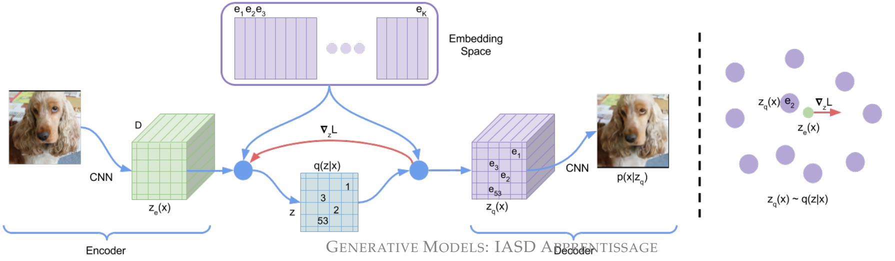
Vector Quantized VAE

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.



References I

[1] 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.

[2] 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.

[3] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022. URL `http://arxiv.org/abs/1312.6114`. arXiv:1312.6114 [cs, stat].

[4] 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].

[5] 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`.