Normalising Flows and Flow Matching

Tutorial

Aryan Goyal

IIT Bombay

March 21, 2026

Overview

1. Normalising Flows

2. Flow Matching

Motivation

Real-NVP flow



Glow flow

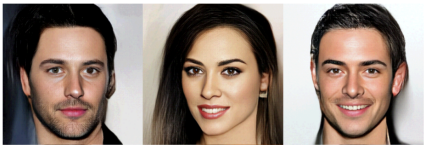
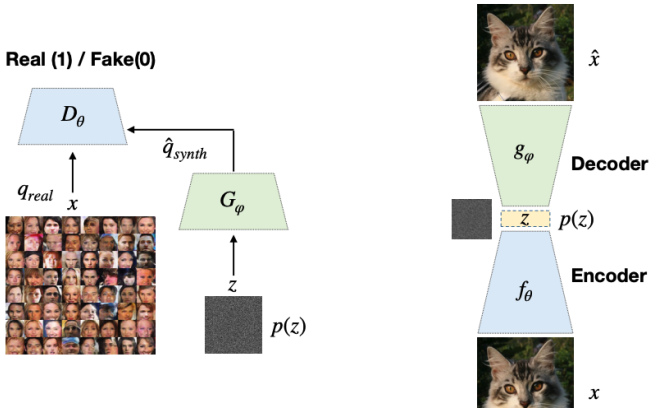


Figure: Images generated using Normalising Flow models

Why not GANs and VAEs



Why Normalizing Flows? (Motivation)

- **Exact likelihood:** Can compute the probability of any data point exactly.
- **Invertibility:** Efficient mapping between data and latent space in both directions.

Typical Properties Across Generative Models

Model	Exact Likelihood	Invertible	Fast Sampling
Normalizing Flows	Yes	Yes	Yes
VAEs	Approximate	No	Yes
GANs	No	No	Yes
Diffusion Models	Approximate	No	No

We'll see these concepts in detail next!

- Model complex distribution over $x \in \mathbb{R}^D$ via transformation of a simple variable:

$$x = T(u), \quad u \sim p_u(u)$$

- $p_u(u)$: base distribution (e.g., Gaussian)
- T: invertible, differentiable transformation
- Parameters: $\{\phi, \psi\}$ for T and p_u

- **Key idea:** Learn a flexible distribution over x by warping a simple one.

Change of Variable Theorem (1D Case)

- Given a random variable $z \sim p(z)$ and a one-to-one transformation $x = f(z)$
- Since f is invertible, we have $z = f^{-1}(x)$
- Goal: Find the density function $p(x)$ of the transformed variable x

- By definition of probability:

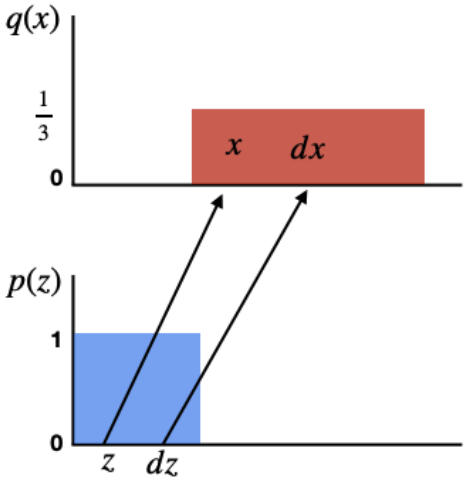
$$\int_x p(x)\, dx = \int_z p(z)\, dz = 1$$

- Hence,

$$p(x) = p(z) \left| \frac{dz}{dx} \right| = p(f^{-1}(x)) \left| \frac{df^{-1}}{dx} \right|$$

- $\left| \frac{df^{-1}}{dx} \right|$ accounts for the scaling between coordinates

Change of Variable Theorem (1D Case)



Change of Variable Theorem (Multivariable Case)

- Suppose $z \sim p(z)$ and $x = f(z)$ is an invertible mapping
- Then $z = f^{-1}(x)$ and the density transforms as:

$$p(x) = p(f^{-1}(x)) \cdot \left| \det \left(\frac{\partial f^{-1}}{\partial x} \right) \right|$$

- $\left| \det \left(\frac{\partial f^{-1}}{\partial x} \right) \right|$ is the Jacobian determinant
- It captures how volume elements change under the transformation from z-space to x-space

- Given an invertible, differentiable transformation $x = T(u)$, with $u \sim p_u(u)$:

$$p_x(x) = p_u(u) \left|\det J_T(u)\right|^{-1}, \quad \text{where } u = T^{-1}(x)$$

- Or equivalently:

$$p_x(x) = p_u(T^{-1}(x)) \left|\det J_{T^{-1}}(x)\right|$$

- $J_T(u)$: Jacobian matrix of T

- This change-of-variables rule enables **exact density evaluation** — a core strength of normalizing flows.
- The transformation T is often learned via a **sequence of simpler invertible mappings** (a "flow").

- Given two invertible, differentiable functions T_1 and T_2:

$$T = T_2 \circ T_1 \quad \Rightarrow \quad T^{-1} = T_1^{-1} \circ T_2^{-1}$$

- The Jacobian determinant of the composition:

$$\det J_{T_2 \circ T_1}(u) = \det J_{T_2}(T_1(u)) \cdot \det J_{T_1}(u)$$

- This composition preserves **invertibility and differentiability**.
- Enables **exact density computation** even for complex transformations.

- In practice, we chain K transformations:

$$T = T_K \circ \cdots \circ T_1, \quad \text{with } z_0 = u,\ z_K = x$$

- Each step gradually warps the base distribution into the target.
- The **"flow"** refers to this sequence of transformations applied to samples.
- The **"normalizing"** part comes from the invertible mapping back to a known density.

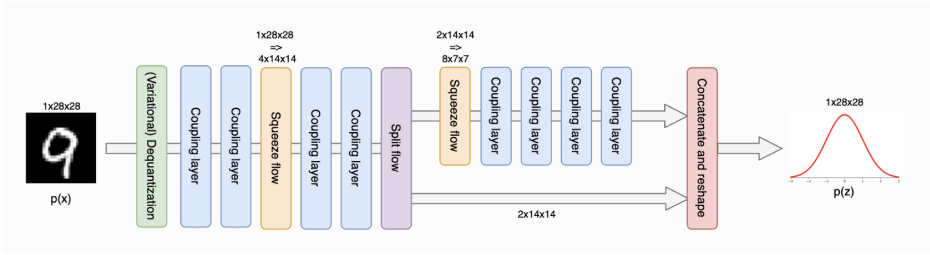


Figure: Composition of flows

Recipe for Learning with Normalizing Flows

Given:

- Dataset: $\mathcal{D} = \{x_1, x_2, ..., x_n\} \sim q(x)$
- Goal: Learn the unknown density $q(x)$ with a flow model

Step-by-Step Recipe:

1. **Choose a simple source density** $p(z)$ (e.g., standard Gaussian)
2. **Learn an invertible transformation** T such that $x = T(z)$
3. **Train via Maximum Likelihood:**

$$\max_T \sum_{i=1}^{n} \log p(z_i) - \log \det(\nabla_z T(z_i))$$

where $z_i = T^{-1}(x_i)$

Designing flow models is about balancing flexibility with tractable computation for both inversion and Jacobians.

Recipe for Learning with Normalizing Flows

Key Points and Practicalities:

- We need to invert T: compute $z_i = T^{-1}(x_i)$ for every x_i
- Need to compute Jacobian determinant $\det(\nabla_z T(z_i))$ for each data point
- **Challenge:** Both inversion and determinant can be expensive!
- **Solution:** Use architectures (e.g., *triangular maps*) where inverse and Jacobian determinant are fast and tractable

Designing flow models is about balancing flexibility with tractable computation for both inversion and Jacobians.

Triangular Maps: Efficient Flows

$$
\begin{aligned}
x_1 &= T_1(z_1) \\
x_2 &= T_2(z_1, z_2) \\
x_3 &= T_3(z_1, z_2, z_3) \\
&\vdots \\
x_d &= T_d(z_1, z_2, \ldots, z_d)
\end{aligned}
$$

Jacobian Structure (Lower Triangular):

$$
\nabla_z T = \begin{bmatrix}
\frac{\partial T_1}{\partial z_1} & 0 & \cdots & 0 \\
\frac{\partial T_2}{\partial z_1} & \frac{\partial T_2}{\partial z_2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial T_d}{\partial z_1} & \cdots & \cdots & \frac{\partial T_d}{\partial z_d}
\end{bmatrix}
$$

Triangular Maps: Efficient Flows

Theory:
(Bogachev et al., *Triangular Transformation of Measures*, Sbornik Math, 2005):
There always exists a unique increasing triangular map that transforms a source density to a target density.

- To fit a flow-based model $p_x(x;\theta)$ to a target distribution $p_x^*(x)$, we minimize a divergence between them.
- Let $\theta = \{\phi, \psi\}$ where:
  - ϕ: parameters of the transformation T
  - ψ: parameters of the base distribution $p_u(u)$
- We use the forward KL divergence:

$$\mathcal{L}(\theta) = D_{\mathrm{KL}}[p_x^*(x) \,\|\, p_x(x;\theta)] = -\mathbb{E}_{p_x^*}[\log p_x(x;\theta)] + \text{const.}$$

- Using the change of variables:

$$\log p_x(x;\theta) = \log p_u(T^{-1}(x;\phi);\psi) + \log|\det J_{T^{-1}}(x;\phi)|$$

- So the loss becomes:

$$\mathcal{L}(\theta) = -\mathbb{E}_{p_x^*}\left[\log p_u(T^{-1}(x;\phi);\psi) + \log|\det J_{T^{-1}}(x;\phi)|\right] + \text{const.}$$

- This objective is optimized via gradient descent using samples from $p_x^*(x)$.

Core Operations of a Trained Normalizing Flow

1. **Sampling**
   - Generate data samples by mapping from the base distribution:

$$u \sim p_u(u) \quad \Rightarrow \quad x = T(u)$$

   - Enables generation of realistic samples from $p_x(x)$
   - Similar to VAEs and GANs, but **exact and efficient**

2. **Density Evaluation**
   - Compute likelihood of a sample x using the inverse transformation:

$$p_x(x) = p_u(T^{-1}(x)) \cdot |\det J_{T^{-1}}(x)|$$

   - Enables **exact log-likelihood** computation
   - Useful for **model comparison**, **anomaly detection**, and **inference**

Flow Models

Introducing Ordinary Differential Equations (ODEs)

- We define a trajectory as a function:

$$X : [0,1] \to \mathbb{R}^d, \quad t \mapsto X_t$$

- Describes how a point moves through space $\mathbb{R}^d$ over time $t \in [0,1]$

- An ODE is defined by a vector field:

$$u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d, \quad (x,t) \mapsto u_t(x)$$

- For each point x and time t, $u_t(x)$ specifies a velocity vector
- Think of u_t as "telling the particle where to go"

- A trajectory X_t follows the vector field if it satisfies:

$$\frac{d}{dt}X_t = u_t(X_t) \quad \text{(ODE)}, \qquad X_0 = x_0 \quad \text{(initial condition)}$$

- This defines how the position evolves over time starting at x_0

Flow example

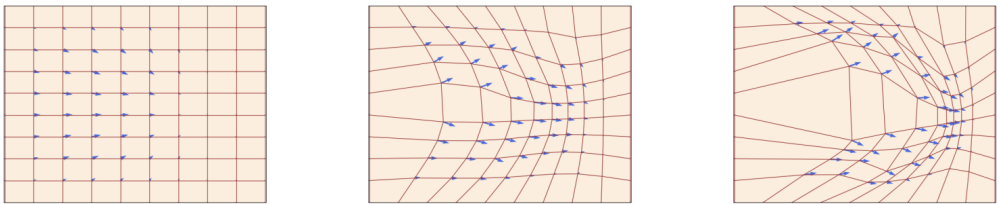


Figure: A flow $\psi t : R\ d \to R\ d$ (red square grid) is defined by a velocity field $ut : R\ d \to R\ d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations (here, $d = 2$). We show three different times t. As one can see, a flow is a diffeomorphism that "warps" space.

Generative Modeling with ODEs

- Goal: Transform a simple distribution p_{init} into a complex target distribution p_{data}
- Define a flow using an ODE:

$$X_0 \sim p_{\text{init}} \quad \text{(random initialization)}$$

$$\frac{d}{dt} X_t = u_t^\theta(X_t) \quad \text{(ODE with neural vector field)}$$

- $u_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ is a neural network parameterized by θ
- The ODE solution defines a flow ψ_t^θ:

$$X_1 = \psi_1^\theta(X_0) \sim p_{\text{data}}$$

- **Important:** The vector field u_t^θ, not the flow ψ_t^θ, is learned
- Simulating the ODE transports p_{init} to match p_{data}

Learning a Flow Model via ODEs

- Define the flow model:

$$X_0 \sim p_{\text{init}}, \quad \frac{dX_t}{dt} = u_t^\theta(X_t)$$

- u_t^θ: neural network (vector field) parameterized by θ
- Simulating this ODE transports samples from p_{init} toward p_{data}
- But a randomly initialized u_t^θ produces meaningless outputs
- We train the model by minimizing the loss:

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2$$

- $u_t^{\text{target}}(x)$: desired vector field that guides samples toward the data distribution
- **Goal:** Derive a formula for this target vector field

Defining a Probability Path

- To construct the training target u_t^{target}, we define a **probability path** — a smooth interpolation between noise and data.
- Use a **conditional probability path**:

$$p_t(x \mid z), \quad \text{with} \quad p_0(x \mid z) = p_{\text{init}}, \quad p_1(x \mid z) = \delta_z$$

- δ_z: Dirac delta centered at data point $z \in \mathbb{R}^d$
- Behavior over time:
  - At $t = 0$: full noise (randomness)
  - At $t = 1$: becomes exactly the data point z
- This describes how a sample z **"emerges" from noise** over time

Induced Marginal Path: Intuition and Construction

- The conditional path $p_t(x \mid z)$ (how x "emerges" from z at time t) induces a **marginal path** over x:
- **Intuitive process:**
  Pick a data point z at random from your dataset ($z \sim p_{\text{data}}(z)$).
  For this z, generate x by adding the right amount of noise ($x \sim p_t(x|z)$).
  Repeat many times: you get a "cloud" of x's—this is the distribution $p_t(x)$.
- **Mathematically:**

$$p_t(x) = \int p_t(x \mid z)\, p_{\text{data}}(z)\, dz$$

  (Average over all possible z after noising each z at time t)
- This creates a **trajectory in distribution space** from noise to data:

$$p_0(x) = p_{\text{init}} \quad \rightarrow \quad p_1(x) = p_{\text{data}}$$

- As t increases from 0 to 1, $p_t(x)$ evolves smoothly from pure noise to the real data distribution.

Sampling from the Marginal Path

- To sample from the marginal distribution $p_t(x)$:
  1. Sample $z \sim p_{\text{data}}$
  2. Sample $x \sim p_t(x \mid z)$
- This gives $x \sim p_t(x)$ — a sample from the intermediate distribution at time t
- **Limitation:** The exact density $p_t(x)$ is unknown
  - The integral

$$p_t(x) = \int p_t(x \mid z)\, p_{\text{data}}(z)\, dz$$

    is **intractable** in practice

Theorem: Marginalization Trick

Conditional Vector Field

For every data point $z \in \mathbb{R}^d$, let $u_t^{\text{target}}(\cdot \mid z)$ denote a conditional vector field such that the following ODE produces the conditional probability path $p_t(\cdot \mid z)$:

$$X_0 \sim p_{\text{init}}, \quad \frac{d}{dt} X_t = u_t^{\text{target}}(X_t \mid z) \quad \Rightarrow \quad X_t \sim p_t(\cdot \mid z), \quad 0 \leq t \leq 1$$

Marginal Vector Field

The marginal vector field $u_t^{\text{target}}(x)$ is then defined by:

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x \mid z) \, \frac{p_t(x \mid z) \, p_{\text{data}}(z)}{p_t(x)} \, dz$$

Implication of the Marginalization Trick

- **Key idea:** The marginalization trick lets us compute the **overall vector field** (which moves the whole data distribution from noise to data) simply by averaging the much easier **conditional vector fields** (which move single samples).
- This simplifies the learning process:
- We don't need to find a complicated formula for the motion of every x in the population.
- Instead, we just figure out the motion for each possible z (a data point) and average these in the right way.

Practical advantage:

- Conditional vector fields are often **analytically tractable**: that means we can write them down with basic calculus, instead of estimating them with expensive or noisy computation.
- The trick turns a potentially intractable global problem into a tractable local one.

Gaussian Probability Path — Example

- Define a conditional probability path $p_t(x \mid z)$ to smoothly transition from noise to data:

$$p_t(x \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I)$$

- **Mean:** $\alpha_t z$ — linearly pulls the sample toward the data point z
- **Variance:** $\beta_t^2 I$ — controls the level of noise

- Choose schedulers to satisfy:

$$\alpha_0 = 0, \quad \beta_0 = 1 \quad \Rightarrow \quad p_0(x \mid z) = \mathcal{N}(0, I) = p_{\text{init}} \quad \text{(pure noise)}$$

$$\alpha_1 = 1, \quad \beta_1 = 0 \quad \Rightarrow \quad p_1(x \mid z) = \mathcal{N}(z, 0) = \delta_z \quad \text{(pure data)}$$

- **Interpretation:** $p_t(x \mid z)$ transitions smoothly:
  - From standard Gaussian at $t = 0$
  - To a delta spike at z at $t = 1$

Sampling from the Marginal Path

- **Step 1: Define the Marginal Path**

$$p_t(x) = \int p_t(x \mid z)\, p_{\text{data}}(z)\, dz$$

  Intuition: For each $z \sim p_{\text{data}}$, add Gaussian noise $\Rightarrow x \sim p_t(x \mid z) \Rightarrow$ Smooth blend of data and noise

- **Step 2: Sampling Procedure**
  - Sample $z \sim p_{\text{data}}, \quad \epsilon \sim \mathcal{N}(0, I)$
  - Compute $x = \alpha_t z + \beta_t \epsilon \Rightarrow x \sim p_t(x)$

- **Why it works:**
  - At $t = 1$: $\beta_t = 0 \Rightarrow x = z$ (pure data)
  - At $t = 0$: $\alpha_t = 0 \Rightarrow x \sim \mathcal{N}(0, I)$ (pure noise)

- **Intuition:**
  - α_t: scales influence of data
  - β_t: scales injected noise
  - As $t \to 0$: noise dominates; as $t \to 1$: clean data recovered

Differentiate the Flow

- Define the flow trajectory:
$$\psi_t(x \mid z) = \alpha_t z + \beta_t x$$
- Differentiate w.r.t. time t:
$$\frac{d}{dt}\psi_t(x \mid z) = \dot{\alpha}_t z + \dot{\beta}_t x$$
- Express in terms of current position $x_t = \psi_t(x \mid z)$ (not original x):
$$x = \frac{x_t - \alpha_t z}{\beta_t}$$
- Substitute back:
$$\frac{d}{dt}\psi_t(x \mid z) = \dot{\alpha}_t z + \dot{\beta}_t \left(\frac{x_t - \alpha_t z}{\beta_t} \right)$$
- **Final Expression for Target Vector Field:**
$$u_t^{\text{target}}(x_t \mid z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t \right) z + \frac{\dot{\beta}_t}{\beta_t} x_t$$

Conditional Flow Matching (CFM)

- Recall the marginal vector field:

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x \mid z) \cdot \frac{p_t(x \mid z)\, p_{\text{data}}(z)}{p_t(x)}\, dz$$

- This integral is **intractable** in practice

- Instead, use the **tractable** conditional velocity field $u_t^{\text{target}}(x \mid z)$
- Define the **Conditional Flow Matching (CFM) loss**:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)} \left[\left\| u_\theta^t(x) - u_t^{\text{target}}(x \mid z) \right\|^2 \right]$$

- **Key idea:**
  - Although we care about the marginal vector field $u_t^{\text{target}}(x)$
  - We regress against $u_t^{\text{target}}(x \mid z)$, which is available in closed form
- Does it make sense to regress against the conditional vector field if it's the marginal vector field we care about?

Theorem: Equivalence of Flow Matching Losses

Statement

The marginal and conditional flow matching losses differ only by a constant:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathcal{L}_{\text{CFM}}(\theta) + C$$

where C is independent of θ.

- Their gradients are equal:

$$\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta)$$

- So minimizing $\mathcal{L}_{\text{CFM}}(\theta)$ via SGD also minimizes $\mathcal{L}_{\text{FM}}(\theta)$
- At the minimizer θ^*:

$$u^t_{\theta^*} = u^{\text{target}}_t \quad \text{(assuming an infinitely expressive model)}$$

Algorithm: Flow Matching Training (Gaussian CondOT Path)

Algorithm 3 Flow Matching Training Procedure (here for Gaussian CondOT path $p_t(x|z) = \mathcal{N}(tz, (1-t)^2)$)

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x = tz + (1-t)\epsilon$ (General case: $x \sim p_t(\cdot|z)$)
6: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - (z - \epsilon)\|^2 \qquad \text{(General case: } = \|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2)$$

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

Figure:

Suggested Readings and Resources

- **Generative AI with Stochastic Differential Equations**
  MIT IAP 2025, "The Missing Semester" project.
  Comprehensive class site with notes and labs on diffusion and flow models.
  diffusion.csail.mit.edu

- **Normalizing Flows Tutorial, Part 1: Distributions and Determinants**
  Blog post by Eric Jang, 2018.
  Clear and visual introduction to normalizing flow building blocks.
  blog.evjang.com/2018/01/nf1.html

- **Flow Matching Guide and Code**
  Yaron Lipman, Máté Havasi, Paul Holderrieth et al., 2024.
  Recent arXiv paper—a practical guide with open-source implementation.
  arxiv.org/abs/2412.06264

The End