Generative Artificial Intelligence

Flow-based Models

Seungjin Yang

Mon 3 – Wed 5, Aug 2026

Kyung Hee University

Outline

1. Normalizing Flows
2. Continuous Normalizing Flows
3. Flow Matching

Learning Goals

After this lecture, students should be able to:

- explain normalizing flows using the change-of-variables formula;
- interpret continuous normalizing flows as ODE-based transport;
- derive the flow matching objective from conditional probability paths;
- distinguish conditional velocity from marginal velocity;
- understand why coupling and path design affect sampling efficiency.

Central question

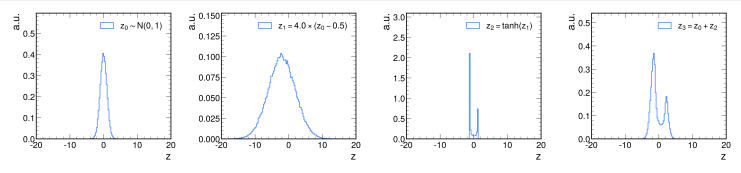
How do we move samples from a simple distribution to the data distribution?

Roadmap

- Normalizing flows define explicit invertible transformations.
- Continuous flows replace finite transformations by a time-dependent ODE.
- Flow matching trains the ODE velocity field without solving ODEs during training.

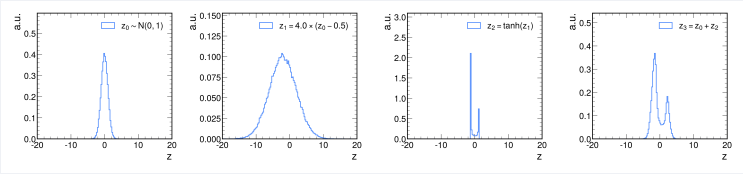
Normalizing Flows

Introduction



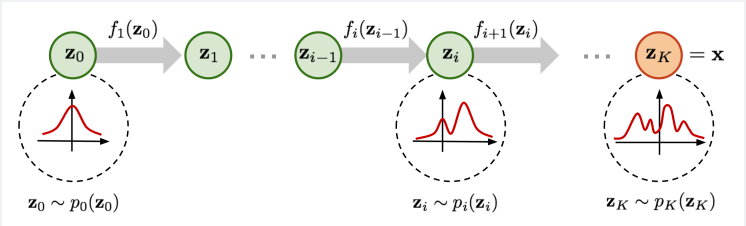
- GenAI aims to model a data distribution $p(\mathbf{x})$ — how do we construct a tractable model of it?
- Latent variable model → VAE
  - Assumes a simple decoding distribution conditioned on a latent variable
  - A decoder network maps the latent to the parameters of the data distribution
- Chain-rule factorization → Autoregressive models
  - Factorize the joint distribution via the chain rule into a product of conditionals, modeling each conditional with a shared network
  - Requires a topological ordering of the variables, which is not always natural
- A third approach: transform a distribution directly — flow-based models.
- Flow-based models rest on the change-of-variables theorem: an invertible map f sends $\mathbf{z} \sim p(\mathbf{z})$ to $\mathbf{x} = f(\mathbf{z})$, with $p(\mathbf{x}) = p(\mathbf{z})\,|\det \partial f/\partial \mathbf{z}|^{-1}$ at $\mathbf{z} = f^{-1}(\mathbf{x})$; composing invertible maps turns a simple distribution into a complex one while keeping the density exactly computable.

Distribution Transformation



- Sample z from a simple base distribution $p_Z(z)$, e.g. a standard Gaussian
- Apply a transformation f to z, yielding $x = f(z) \sim p_X(x)$
  - Probability mass is conserved
  - The map only reshapes the density
- A sufficiently expressive f reshapes a simple $p_Z(z)$ into an approximation of a broad class of continuous densities $p_X(x)$
  - All the complexity is carried by the map, not by the base distribution
  - A bijection preserves dimension and cannot change the topology of the support, so the approximation is not exact for every target
- This reshaping idea is the core mechanism behind normalizing flows, formalized next

Normalizing Flow: Core Definition



- A normalizing flow maps a base random variable to a data-space variable by an invertible function:

$$\mathbf{z} \sim p_Z(\mathbf{z}), \qquad \mathbf{x} = f_\theta(\mathbf{z}). \tag{1}$$

- Because the map is bijective,

$$\mathbf{z} = f_\theta^{-1}(\mathbf{x}). \tag{2}$$

- Generation
  - Sample $\mathbf{z} \sim p_Z$, then compute $\mathbf{x} = f_\theta(\mathbf{z})$
- Density evaluation
  - Given $\mathbf{x}$, compute $\mathbf{z} = f_\theta^{-1}(\mathbf{x})$ and the Jacobian correction

Change of Variables

- For $\mathbf{x} = f_\theta(\mathbf{z})$, probability mass is conserved:

$$p_X(\mathbf{x})\, d\mathbf{x} = p_Z(\mathbf{z})\, d\mathbf{z}. \tag{3}$$

- Therefore,

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right|. \tag{4}$$

- Equivalently,

$$\log p_X(\mathbf{x}) = \log p_Z\left(f_\theta^{-1}(\mathbf{x})\right) - \log \left| \det \frac{\partial f_\theta}{\partial \mathbf{z}} \right|. \tag{5}$$

- The determinant corrects for local volume expansion or compression

Composition of Invertible Maps

- A flow usually composes many simple invertible layers:

$$\mathbf{z}_0 \xrightarrow{f_1} \mathbf{z}_1 \xrightarrow{f_2} \cdots \xrightarrow{f_K} \mathbf{z}_K = \mathbf{x}. \tag{6}$$

  - Each layer is kept simple so that its inverse and Jacobian determinant remain cheap, which limits how much one layer can deform the density
  - Composition builds an expressive map from weak layers: the composition of invertible maps is invertible, its inverse is the reverse composition, and its log-determinant is a sum

- The log density is

$$\log p_X(\mathbf{x}) = \log p_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|. \tag{7}$$

- Practical requirement on each layer:
  - invertible
  - tractable Jacobian determinant

Maximum-Likelihood Training

- Given data $\{\mathbf{x}_i\}_{i=1}^N$, train by negative log likelihood:

$$\mathcal{L}_{\mathrm{NLL}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i). \tag{8}$$

- For each data point,

$$\mathbf{z}_i = f_\theta^{-1}(\mathbf{x}_i), \tag{9}$$

$$\log p_\theta(\mathbf{x}_i) = \log p_Z(\mathbf{z}_i) + \log \left| \det \frac{\partial f_\theta^{-1}}{\partial \mathbf{x}_i} \right|. \tag{10}$$

- An NF is exact likelihood under the model, not an ELBO approximation
- Benefits of exact maximum-likelihood training:
  - A single scalar objective, optimized by plain gradient descent — no adversarial min-max, no ELBO gap
  - Likelihood values are comparable across checkpoints, and across models that share the same data representation, dimension, preprocessing, and dequantization
  - The trained model is usable as a density estimator, e.g. for outlier detection or event reweighting
  - Caveat: a higher likelihood does not guarantee perceptually better samples

Normalizing Flow: Summary

- Take a standard Gaussian base distribution, $\mathbf{z} \sim p_Z = \mathcal{N}(0, I)$
- The maximum-likelihood objective is then explicit:

$$\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \left[\log \mathcal{N}\left(f_\theta^{-1}(\mathbf{x}_i) \,\middle|\, 0, I \right) + \log \left| \det \frac{\partial f_\theta^{-1}}{\partial \mathbf{x}_i} \right| \right]. \tag{11}$$

- Training: map data to latent, $\mathbf{x} \to \mathbf{z} = f_\theta^{-1}(\mathbf{x})$, accumulating the log-determinant of each layer to obtain $\log p_\theta(\mathbf{x})$
- Generation: sample $\mathbf{z} \sim \mathcal{N}(0, I)$ and run the inverse direction, $\mathbf{x} = f_\theta(\mathbf{z})$
- Both directions use the same layers, so the architecture must make both computable — but not necessarily at comparable cost
  - Coupling flows are efficient in both directions
  - Autoregressive flows are parallel in one direction and sequential in the other, so the architecture is chosen according to whether density evaluation or sampling matters more

Why Special Flow Layers Are Needed

- A generic neural network is not automatically a normalizing flow
  - It may not be invertible
  - It may change dimension
  - Its Jacobian determinant may require $O(D^3)$ computation
  - Its inverse may not be available in closed form
- Thus flow layers impose algebraic structure:
  - triangular Jacobians
  - coupling masks
  - autoregressive factorization
  - monotone scalar transformations

Affine Coupling Layer

- Split $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$; define

$$\mathbf{y}_a = \mathbf{x}_a, \qquad \mathbf{y}_b = \mathbf{x}_b \odot \exp s_\theta(\mathbf{x}_a) + t_\theta(\mathbf{x}_a). \tag{12}$$

- The inverse is explicit:

$$\mathbf{x}_a = \mathbf{y}_a, \qquad \mathbf{x}_b = (\mathbf{y}_b - t_\theta(\mathbf{y}_a)) \odot \exp[-s_\theta(\mathbf{y}_a)]. \tag{13}$$

- The Jacobian is triangular, so the log-determinant is a plain sum, at $O(D)$ cost:

$$\log|\det J| = \sum_j s_{\theta,j}(\mathbf{x}_a). \tag{14}$$

- s_θ and t_θ are arbitrary neural networks: invertibility comes from the coupling structure, not from constraining the networks
- $\mathbf{x}_a$ passes through unchanged, so successive layers alternate the partition (or permute dimensions) to transform every coordinate

Examples

NICE [1], RealNVP [2], and Glow [3] are built from coupling-type transformations.

Autoregressive Flows: MAF and IAF

- Autoregressive flows transform each coordinate conditioned on previous coordinates, e.g. in the IAF generative direction:

$$x_i = \mu_i(z_{<i}) + \sigma_i(z_{<i}) z_i. \tag{15}$$

- Each x_i depends only on $z_{\leq i}$, so the Jacobian is triangular by construction:

$$\log |\det J| = \sum_i \log \sigma_i(z_{<i}). \tag{16}$$

- MAF is the inverse parameterization: its conditioner reads the data coordinates $x_{<i}$, so $z_i = (x_i - \mu_i(x_{<i}))/\sigma_i(x_{<i})$
- The coupling layer is a special case: two blocks instead of D sequential steps
- One direction is parallel, the other requires D sequential passes — the two variants choose opposite directions

MAF [4]

Parallel $\mathbf{x} \to \mathbf{z}$: fast likelihood and training; sequential sampling.

IAF [5]

Parallel $\mathbf{z} \to \mathbf{x}$: fast sampling; sequential likelihood evaluation.

Spline Flow

- Affine transformations can be too restrictive
- Spline flows use monotone nonlinear scalar maps:

$$y_i = g_i(x_i; \theta_i), \qquad \frac{\partial g_i}{\partial x_i} > 0. \tag{17}$$

- Common choice: rational-quadratic spline [6]
  - Knot positions and slopes are predicted by the conditioner network, inside a coupling or autoregressive layer
  - Monotonicity guarantees invertibility
  - Log-determinant follows from the scalar derivatives
- **Main lesson:** A flow layer trades architectural freedom for exact density evaluation

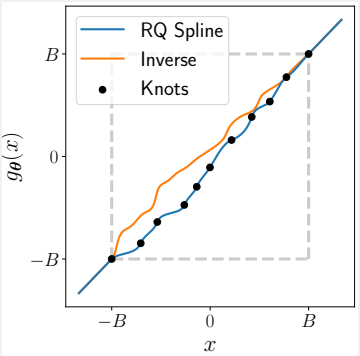


Figure 1: An example of rational-quadratic spline. Source: Ref. [6].

Strengths and Limitations of Normalizing Flows

- Strengths
  - Exact likelihood under the model
  - Exact inverse mapping
  - Maximum-likelihood training
  - Deterministic generation from noise
- Limitations
  - Strong invertibility constraint
  - Equal-dimensional latent and data spaces
  - Restricted architectures
  - Expensive or constrained Jacobians

Continuous Normalizing Flows

From Discrete Layers to Continuous Dynamics

- A discrete flow applies finite transformations:

$$\mathbf{z}_{k+1} = f_k(\mathbf{z}_k). \tag{18}$$

- Now imagine each layer is an infinitesimal update:

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + \mathbf{v}_\theta(\mathbf{z}(t), t)\Delta t. \tag{19}$$

- Taking $\Delta t \to 0$ yields an ODE:

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{v}_\theta(\mathbf{z}(t), t). \tag{20}$$

- Read backwards, K residual layers with step $\Delta t = 1/K$ are the Euler discretization of this ODE, and the ODE is their $K \to \infty$ limit [7]
  - Depth becomes continuous time over a chosen integration interval, e.g. $[0, 1]$, instead of a fixed number of layers; the solver then sets the evaluation points and the number of function evaluations
  - One time-conditioned network $\mathbf{v}_\theta(\cdot, t)$ replaces K separate layers f_k
- A finite composition of maps becomes a continuous trajectory in data space
- Continuous flows relax the layer-by-layer determinant design

Continuous Normalizing Flow

- A continuous normalizing flow (CNF) defines the generative map by solving

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t, t), \qquad \mathbf{x}_0 \sim p_0. \tag{21}$$

- The terminal point is

$$\mathbf{x}_1 = \Phi^\theta_{0\to 1}(\mathbf{x}_0), \tag{22}$$

  where $\Phi^\theta_{0\to 1}$ is the ODE flow map
- The flow map $\Phi^\theta_{s\to t}$ sends an initial condition at time s to the solution of the ODE at time t
  - If $\mathbf{v}_\theta$ is sufficiently regular, e.g. Lipschitz continuous in $\mathbf{x}$, solutions are unique and trajectories never cross: $\Phi^\theta_{s\to t}$ is invertible, with inverse $\Phi^\theta_{t\to s}$ obtained by integrating backwards in time
  - Invertibility therefore comes from the dynamics, not from the architecture: $\mathbf{v}_\theta$ needs no coupling structure or triangular Jacobian, only enough regularity to define a unique flow [7]

Probability Path

- A continuous flow induces a time-indexed family of distributions:

$$p_0 \longrightarrow p_t \longrightarrow p_1. \tag{23}$$

- The point move by

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t, t). \tag{24}$$

- The density evolves as probability mass is transported
- Instead of designing many invertible layers, we learn a vector field that moves the entire distribution over time

Continuity Equation

- If p_t is transported by a velocity field $\mathbf{u}_t(\mathbf{x})$, then

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla_{\mathbf{x}} \cdot [p_t(\mathbf{x})\mathbf{u}_t(\mathbf{x})] = 0. \tag{25}$$

- Probability density changes because probability mass flows in or out of a local region
- For a correct generative model,

$$\mathbf{x}_0 \sim p_0 \implies \mathbf{x}_t \sim p_t \implies \mathbf{x}_1 \sim p_1. \tag{26}$$

Instantaneous Change of Variables

- In a CNF, the log density changes according to

$$\frac{d}{dt} \log p_t(\mathbf{x}_t) = -\nabla_{\mathbf{x}} \cdot \mathbf{v}_\theta(\mathbf{x}_t, t). \tag{27}$$

- Therefore,

$$\log p_1(\mathbf{x}_1) = \log p_0(\mathbf{x}_0) - \int_0^1 \nabla_{\mathbf{x}} \cdot \mathbf{v}_\theta(\mathbf{x}_t, t)\, dt. \tag{28}$$

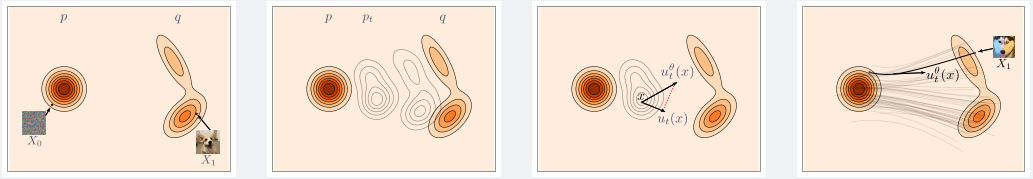
- Jacobian log-determinant $\implies$ integrated divergence

CNF Likelihood Training Is Expensive

- Likelihood-based CNF training may require:
  - solving an ODE during training
  - integrating the log-density equation
  - computing or estimating a divergence term, e.g. by a stochastic trace estimator [8]
  - differentiating through the ODE solver
- Can we train the velocity field by direct regression, without solving ODEs during training?

Flow Matching

The Flow Matching Idea



(a) Data (b) Path design (c) Training (d) Sampling

Figure 2: The Flow Matching blueprint. Source: Ref. [9].

- Flow Matching (FM) answers the CNF question: train the velocity field by regression, without solving ODEs during training
- The recipe:
  - (b) design a probability path p_t from a source $p_0 = p$ (e.g. Gaussian noise) to the data distribution $p_1 = q$
  - (c) train a network u_t^θ to match the velocity field known to generate p_t
  - (d) sample by solving the ODE with u_t^θ from $t = 0$ to $t = 1$

The Flow Matching Problem

Goal

Find a velocity field u_t^θ generating a probability path p_t with $p_0 = p$ and $p_1 = q$.

- u_t^θ is the neural network v_θ of the previous section, in FM notation
- Suppose we knew a target velocity field u_t that generates p_t; then we could regress onto it:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim U[0,1],\, X_t\sim p_t} \left\| u_t^\theta(X_t) - u_t(X_t) \right\|^2 . \tag{29}$$

- A plain regression loss: no ODE solving, no divergence, no Jacobian
- Two problems remain:
  - How do we design the path p_t?
  - The target u_t transports the whole distribution p to q — we do not know it
- FM resolves both with one idea: conditioning on a single data point

Conditional Probability Path

- Fix one training example x_1; design a simple path toward it:

$$p_{t|1}(x|x_1) = \mathcal{N}\big(x \,\big|\, tx_1,\, (1-t)^2 I\big)\,. \tag{30}$$

- Boundary behavior:
  - $t = 0$: $\mathcal{N}(0, I)$ — the source p
  - $t \to 1$: shrinks onto the point x_1 (a delta measure δ_{x_1})
- A Gaussian blob that moves to x_1 while narrowing
- Designing a path to one point is easy; the data distribution enters next by averaging

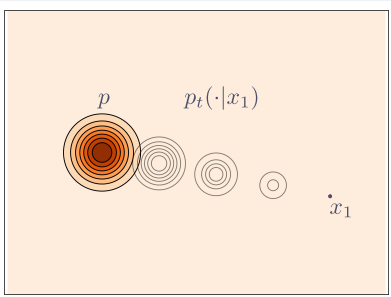


Figure 3: Conditional path $p_{t|1}(x|x_1)$. Source: Ref. [9].

Marginal Path by Aggregation

- Average the conditional paths over the data distribution:

$$p_t(x) = \int p_{t|1}(x|x_1)\, q(x_1)\, dx_1. \tag{31}$$

- Boundary conditions are inherited:

$$p_0 = p, \qquad p_1 = q. \tag{32}$$

- Sampling $X_t \sim p_t$ is trivial — pick a data point, pick noise, interpolate:

$$X_t = tX_1 + (1-t)X_0, \quad X_0 \sim p,\ X_1 \sim q. \tag{33}$$

- Sampling training states needs no network and no ODE — just a random mixture of straight lines; generation after training still requires both

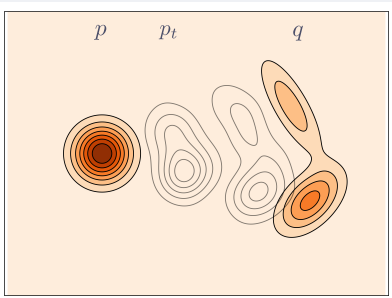


Figure 4: Marginal path $p_t(x)$. Source: Ref. [9].

Conditional Velocity Field

- Which velocity field moves samples along the conditional path?
- The conditional samples follow

$$X_{t|1} = tx_1 + (1-t)X_0. \tag{34}$$

- Differentiate in t and eliminate X_0:

$$u_t(x|x_1) = \frac{x_1 - x}{1-t}. \tag{35}$$

- Interpretation: from position x, head straight to x_1, arriving at $t = 1$
- Known in closed form — this is the regression target

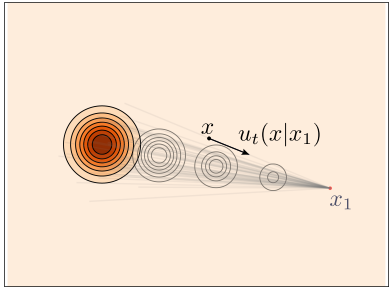


Figure 5: Conditional velocity $u_t(x|x_1)$. Source: Ref. [9].

Marginal Velocity Field

- The velocity that transports the marginal path averages the conditional velocities, weighted by the posterior of x_1 given the current position:

$$u_t(x) = \int u_t(x|x_1)\, p_{1|t}(x_1|x)\, dx_1 . \tag{36}$$

- Equivalently, a conditional expectation:

$$u_t(x) = \mathbb{E}[\, u_t(X_t|X_1) \,|\, X_t = x\,] . \tag{37}$$

- "Given that I am at x at time t, where is the data pulling me on average?"
- Still intractable to compute — but perfect as a regression target, as we will see

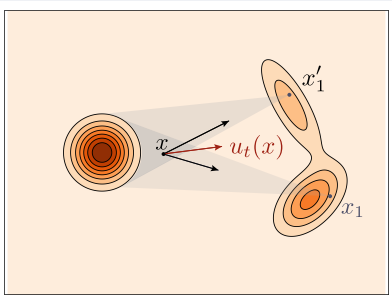


Figure 6: Marginal velocity $u_t(x)$. Source: Ref. [9].

The Marginalization Trick

Theorem (Marginalization Trick [9])

If each conditional velocity $u_t(\cdot|x_1)$ generates its conditional path $p_{t|1}(\cdot|x_1)$, then (under mild regularity assumptions) the marginal velocity u_t generates the marginal path p_t.

- Proof idea: each conditional pair satisfies the continuity equation; integrating over $q(x_1)$ and exchanging integral and divergence gives the continuity equation for the marginal pair
- Consequence: a hard global problem (transport p to q) decomposes into easy per-example problems (transport noise to one x_1)
- We now know a valid regression target u_t exists and what it is — the remaining question is how to regress onto it without computing it

Conditional Flow Matching Loss

- Replace the intractable marginal target by the closed-form conditional target:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,\, X_1,\, X_t} \left\| u_t^\theta(X_t) - u_t(X_t|X_1) \right\|^2. \tag{38}$$

Theorem [10]

The two losses have identical gradients: $\nabla_\theta \mathcal{L}_{\mathrm{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\mathrm{CFM}}(\theta)$. Under the population loss, the minimizer of $\mathcal{L}_{\mathrm{CFM}}$ over all measurable functions is the marginal velocity $u_t(x)$.

- Why: least-squares regression onto noisy per-example targets converges to their conditional mean — which is exactly $u_t(x) = \mathbb{E}[u_t(X_t|X_1) \mid X_t = x]$
- The network sees only simple straight-line targets, yet learns the full transport field
- A finite network learns the closest approximation to u_t within its model class, not u_t itself

The Simplest Algorithm

- For the linear path, $u_t(X_t|X_1) = X_1 - X_0$, so the loss becomes

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,\, X_0,\, X_1} \left\| u_t^\theta\big(tX_1 + (1-t)X_0\big) - (X_1 - X_0) \right\|^2. \tag{39}$$

```
for x_1 in dataloader:                          # data batch, x_1 ~ q
    x_0 = torch.randn_like(x_1)                 # noise batch, x_0 ~ N(0, I)
    t = torch.rand(batch_size)                  # t ~ U[0, 1]
    x_t = t * x_1 + (1 - t) * x_0               # sample the path
    loss = ((model(x_t, t) - (x_1 - x_0)) ** 2).mean()
    loss.backward(); optimizer.step(); optimizer.zero_grad()
```

- Each training step: two samples, one interpolation, one regression — no ODE
- This linear path is the conditional optimal transport path: between each paired endpoint it is straight and constant-speed, and optimal under squared Euclidean transport cost
- The induced marginal flow need not be straight — with independent coupling the conditional lines cross, and better couplings or reflow are needed to straighten the trajectories [11]

Sampling

- After training, generation is just the CNF sampler:
  - draw $X_0 \sim \mathcal{N}(0, I)$
  - solve $\frac{dx_t}{dt} = u_t^\theta(x_t)$ from $t = 0$ to $t = 1$
- The simplest solver is the Euler method:

$$x_{t+h} = x_t + h\, u_t^\theta(x_t). \tag{40}$$

- Midpoint, Heun, or adaptive solvers are also used; cost is measured in the number of function evaluations (NFE)
- The required NFE depends on the curvature of the marginal trajectory and the target accuracy, which coupling and path schedule control
- Cost moved from training (CNF likelihood) to a solve at inference

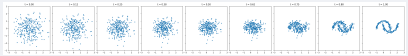


Figure 7: FM trained on 2D data: noise (left) flows to data (right). Source: Ref. [9].

Summary and Outlook

- The arc of this lecture:
  - **NF**: invertible layers, exact likelihood, constrained architectures
  - **CNF**: one free-form ODE instead of many layers, but expensive likelihood training
  - **FM**: same ODE model, trained by simulation-free regression on conditional velocities
- Key concepts: conditional path $\rightarrow$ marginal path; conditional velocity $\rightarrow$ marginal velocity; Marginalization Trick; CFM loss
- Beyond this lecture [9]:
  - affine paths $X_t = \alpha_t X_1 + \sigma_t X_0$ generalize the linear choice; different schedulers (α_t, σ_t) recover diffusion-style paths
  - diffusion models: the stochastic sibling — an SDE instead of an ODE, trained by an analogous conditional regression (denoising score matching); on Gaussian paths the two are related by a change of prediction parameterization, and a diffusion SDE has a probability-flow ODE with the same marginals
  - extensions: couplings, guidance, flows on manifolds, discrete data
- FM is the training paradigm behind recent large-scale image, video, and audio generators

Questions?

Further Reading

- A broad taxonomy of normalizing flows: Papamakarios et al., Normalizing Flows for Probabilistic Modeling and Inference [12]
- The original continuous-flow formulation: Chen et al., Neural Ordinary Differential Equations [7]
- A unified treatment of flow matching, with code: Lipman et al., Flow Matching Guide and Code [9]

References i

Laurent Dinh, David Krueger, and Yoshua Bengio.
NICE: non-linear independent components estimation.
In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio.
Density estimation using real NVP.
In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Diederik P. Kingma and Prafulla Dhariwal.
Glow: Generative flow with invertible 1×1 convolutions.
In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10236–10245, 2018.


George Papamakarios, Iain Murray, and Theo Pavlakou.
Masked autoregressive flow for density estimation.
In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2338–2347, 2017.


Diederik P. Kingma, Tim Salimans, Rafal Józefowicz, Xi Chen, Ilya Sutskever, and Max Welling.
Improving variational autoencoders with inverse autoregressive flow.
In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4736–4744, 2016.


Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. **Neural spline flows.** In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7509–7520, 2019.


Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. **Neural ordinary differential equations.** In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud.
FFJORD: free-form continuous dynamics for scalable reversible generative models.
In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat.
Flow matching guide and code.
CoRR, abs/2412.06264, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le.
Flow matching for generative modeling.
In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu.
Flow straight and fast: Learning to generate and transfer data with rectified flow.
In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.


George Papamakarios, Eric T. Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan.
Normalizing flows for probabilistic modeling and inference.
J. Mach. Learn. Res., 22:57:1–57:64, 2021.