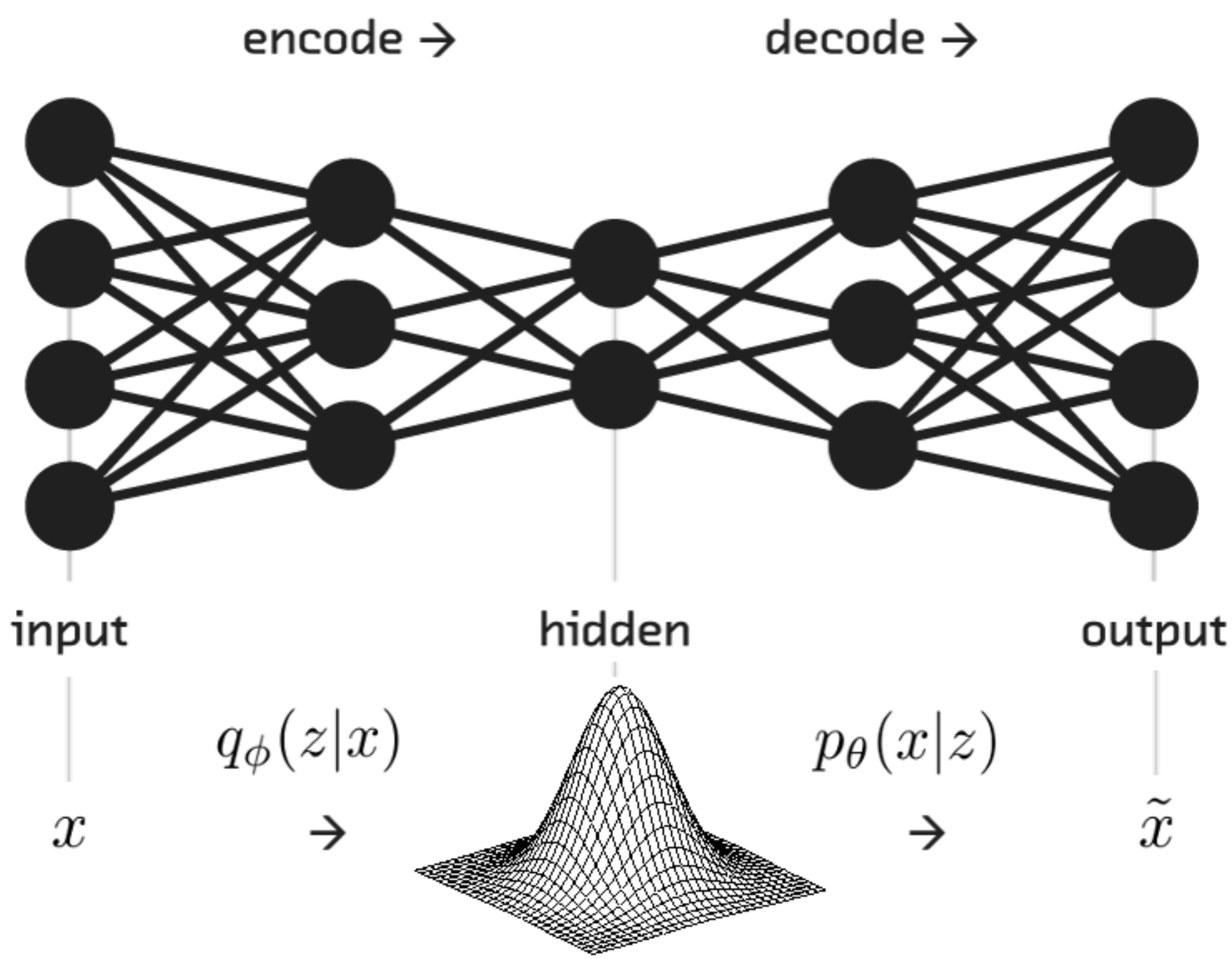
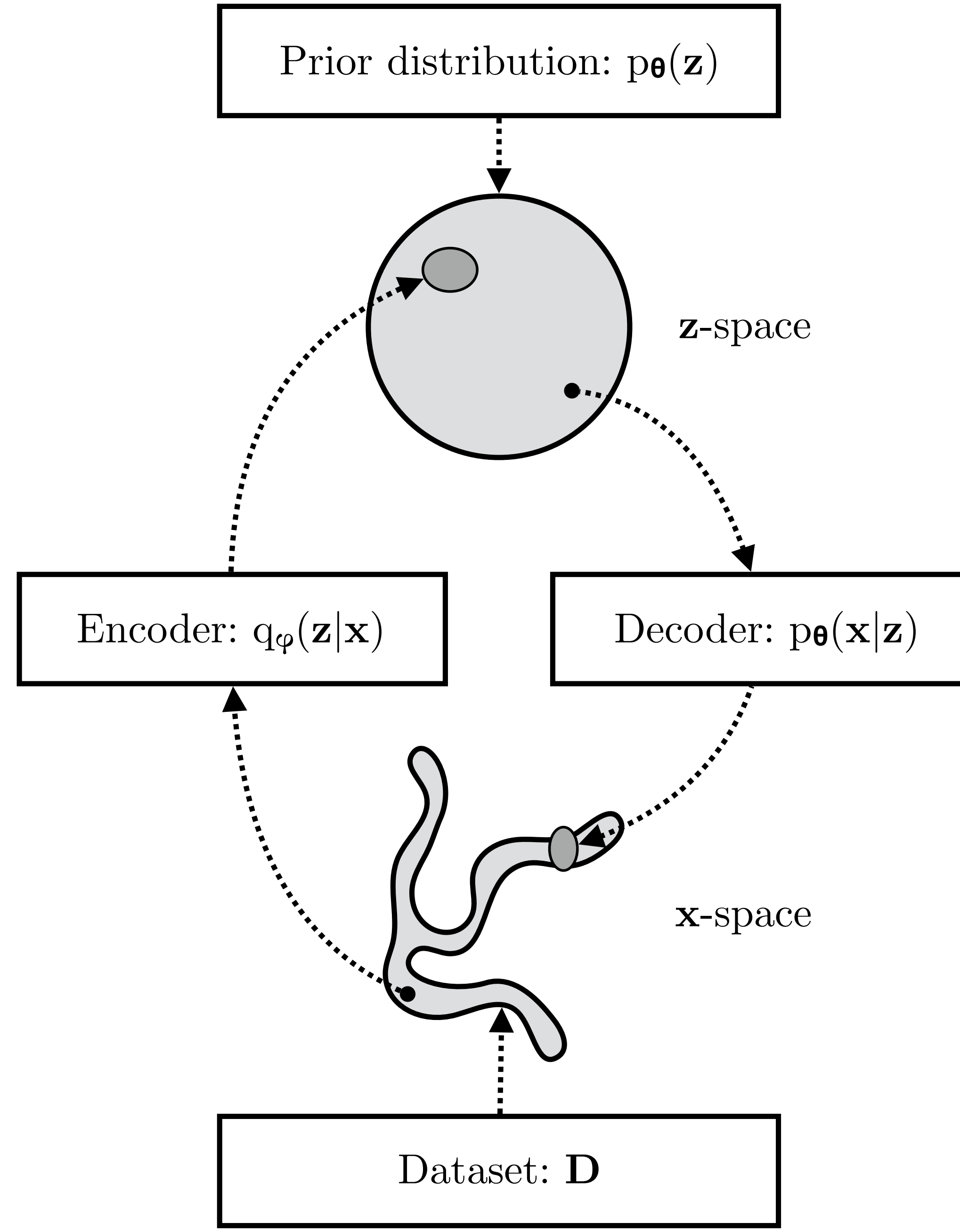
ENM5310: Data-driven modeling and probabilistic scientific computing

Lecture #23: Normalizing Flows

University *of* Pennsylvania

Variational auto-encoders





Kingma, D. P. (2017). Variational inference & deep learning: A new synthesis. (PhD Thesis)

Can we go beyond the mean field approximation for $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$?

Generative Modeling

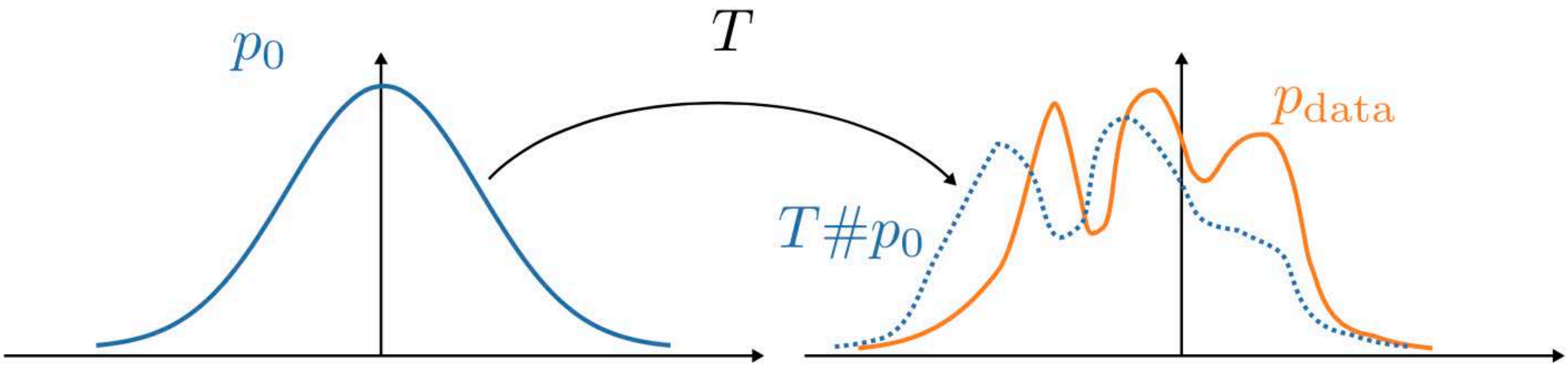


Figure 1. Modern generative modelling principle: trying to find a map $\boldsymbol{T}$ that sends the base distribution $\boldsymbol{p}_0$ as close as possible to the data distribution $\boldsymbol{p}_{\text{data}}$.

$$\boldsymbol{\theta}^* = \underset{\theta}{\operatorname{\mathbf{argmax}}} \sum_{i=1}^{n} \log\left((\boldsymbol{T}_\theta \# \boldsymbol{p}_0)(\boldsymbol{x}^{(i)})\right)$$

Normalizing Flows

In order to compute the log likelihood objective function in (1), if T_θ is a diffeomorphism (and thus has a differentiable inverse T_θ^{-1}), one can rely on the so-called *change-of-variable formula*

$$\log p_1(x) = \log p_0(T_\theta^{-1}(x)) + \log |\det J_{T_\theta^{-1}}(x)| \tag{2}$$

where $J_{T_\theta^{-1}} \in \mathbb{R}^{d\times d}$ is the Jacobian of T_θ^{-1}. Relying on this formula to evaluate the likelihood imposes two constraints on the network:

- T_θ must be invertible; in addition T_θ^{-1} should be easy to compute in order to evaluate the first right-hand side term in (2)
- T_θ^{-1} must be differentiable, and the (log) determinant of the Jacobian of T_θ^{-1} must not be too costly to compute in order to evaluate the second right-hand side term in (2) [5].

Normalizing Flows

The philosophy of Normalizing Flows (NFs) [2, 3, 4] is to design special neural architectures satisfying these two requirements. Normalizing flows are maps $T_\theta = \phi_1 \circ \dots \phi_K$, where each ϕ_k is a simple transformation satisfying the two constraints – and hence so does T_θ. Defining recursively $x_0 = x$ and $x_k = \phi_k(x_{k-1})$ for $k \in \{1, \dots, K\}$, through the chain rule, the likelihood is given by

$$\begin{aligned}\log p_1(x) &= \log p_0(\phi_1^{-1} \circ \dots \circ \phi_K^{-1}(x)) + \log|\det J_{\phi_1^{-1} \circ \dots \circ \phi_K^{-1}}(x)| \\ &= \log p_0(\phi_1^{-1} \circ \dots \circ \phi_K^{-1}(x)) + \sum_{k=1}^{K} \log|\det J_{\phi_k^{-1}}(x_k)|\end{aligned}$$

which is still easy to evaluate provided each ϕ_k satisfies the two constraints.

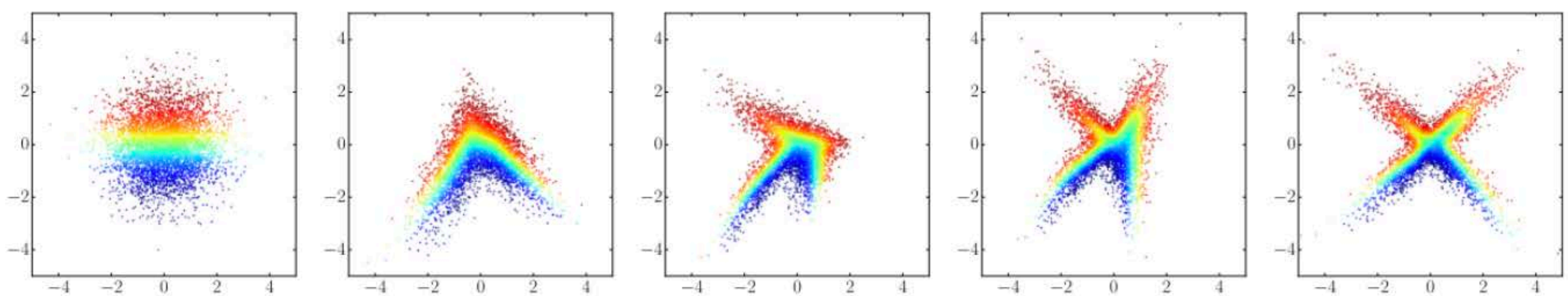


Figure 2. Normalizing flow with $K = 4$, transforming an isotropic Gaussian (leftmost) to a cross shape target distribution (rightmost). Picture from [4]

RealNVP

A more complex example of NF, that satisfies both constraints, is Real NVP [5].

▼ Click here for details about Real NVP

$$\begin{aligned}\phi(x)_{1:d'} &= x_{1:d'} \\ \phi(x)_{d':d} &= x_{d':d} \odot \exp(s(x_{1:d'})) + t(x_{1:d'})\end{aligned} \tag{3}$$

where $d' \leq d$ and the so-called *scale* s and *translation* t are functions from $\mathbb{R}^{d'}$ to $\mathbb{R}^{d-d'}$, parametrized by neural networks. This transformation is invertible in closed-form, and the determinant of its Jacobian is cheap to compute.

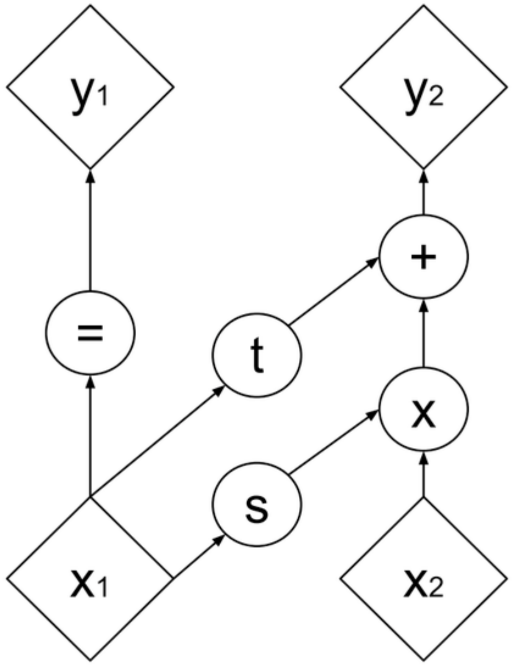
The Jacobian of ϕ defined in (3) is lower triangular:

$$J_\phi(x) = \begin{pmatrix} \mathrm{Id}_{d'} & 0_{d',d-d'} \\ \dots & \mathrm{diag}(\exp(s(x_{1:d}))) \end{pmatrix}$$

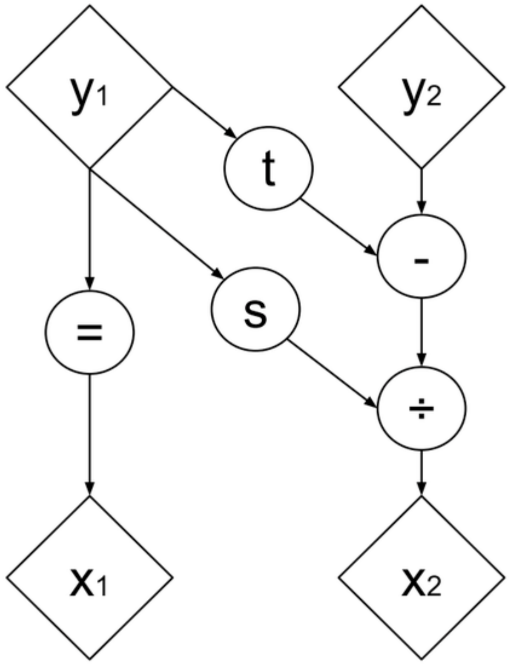
hence its determinant can be computed at a low cost, and in particular without computing the Jacobians of s and t. In addition, ϕ is easily invertible:

$$\begin{aligned}\phi^{-1}(y)_{1:d'} &= y_{1:d'} \\ \phi^{-1}(y)_{d':d} &= (y_{d':d} - t(y_{1:d'})) \odot \exp(-s(y_{1:d'}))\end{aligned}$$

It has met with a huge success in practice and a variety of alternative NFs have been proposed [6, 7, 8]. Unfortunately, the architectural constraints on Normalizing Flows tends to hinder their expressivity [7].



(a) Forward propagation



(b) Inverse propagation

Continuous Normalizing Flows

A successful solution to this expressivity problem is based on an idea similar to that of ResNets, named Continuous Normalizing Flows (CNF) [11]. If we return to the planar normalizing flow, by letting $u_{k-1}(\cdot) \stackrel{\text{def}}{=} K\sigma(b_k^\top \cdot + c)a_k$, we can rewrite the relationship between x_k and x_{k-1} as:

$$\begin{aligned} x_k &= \phi_k(x_{k-1}) \\ &= x_{k-1} + \sigma(b_k^\top x_{k-1} + c)a_k \\ &= x_{k-1} + \frac{1}{K} u_{k-1}(x_{k-1}) \end{aligned}$$

which can be interpreted as a forward Euler discretization, with step $1/K$, of the ODE

$$\begin{cases} x(0) = x_0 \\ \partial_t x(t) = u(x(t), t) \quad \forall t \in [0,1] \end{cases} \tag{4}$$

Note that the mapping defined by the ODE, $T(x_0) := x(1)$ is inherently invertible because one can solve the *reverse-time* ODE (from $t = 1$ to 0) with the initial condition $x(1) = T(x_0)$.

This ODE is called an *initial value problem*, controlled by the **velocity field** $u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$. In addition to u, it is related to two other fundamental objects:

- the **flow** $f^u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$, with $f^u(x,t)$ defined as the solution at time t to the initial value problem driven by u with initial condition $x(0) = x$.
- the **probability path** $(p_t)_{t\in[0,1]}$, defined by $p_t = f^u(\cdot, t)\#p_0$, i.e., p_t is the distribution of $f^u(x,t)$ when $x \sim p_0$.

The Continuity Equation

A fundamental equation linking $\boldsymbol{p_t}$ and $\boldsymbol{u}$ is the *continuity equation*, also called transport equation:

$$\partial_t \boldsymbol{p}_t + \nabla \cdot \boldsymbol{u}_t \boldsymbol{p}_t = 0 \tag{5}$$

Under technical conditions and up to divergence-free vector fields, for a given $\boldsymbol{p_t}$ (resp. $\boldsymbol{u}$) there exists a $\boldsymbol{u}$ (resp. $\boldsymbol{p_t}$) such that the pair $(\boldsymbol{p_t}, \boldsymbol{u})$ solves the continuity equation [8].

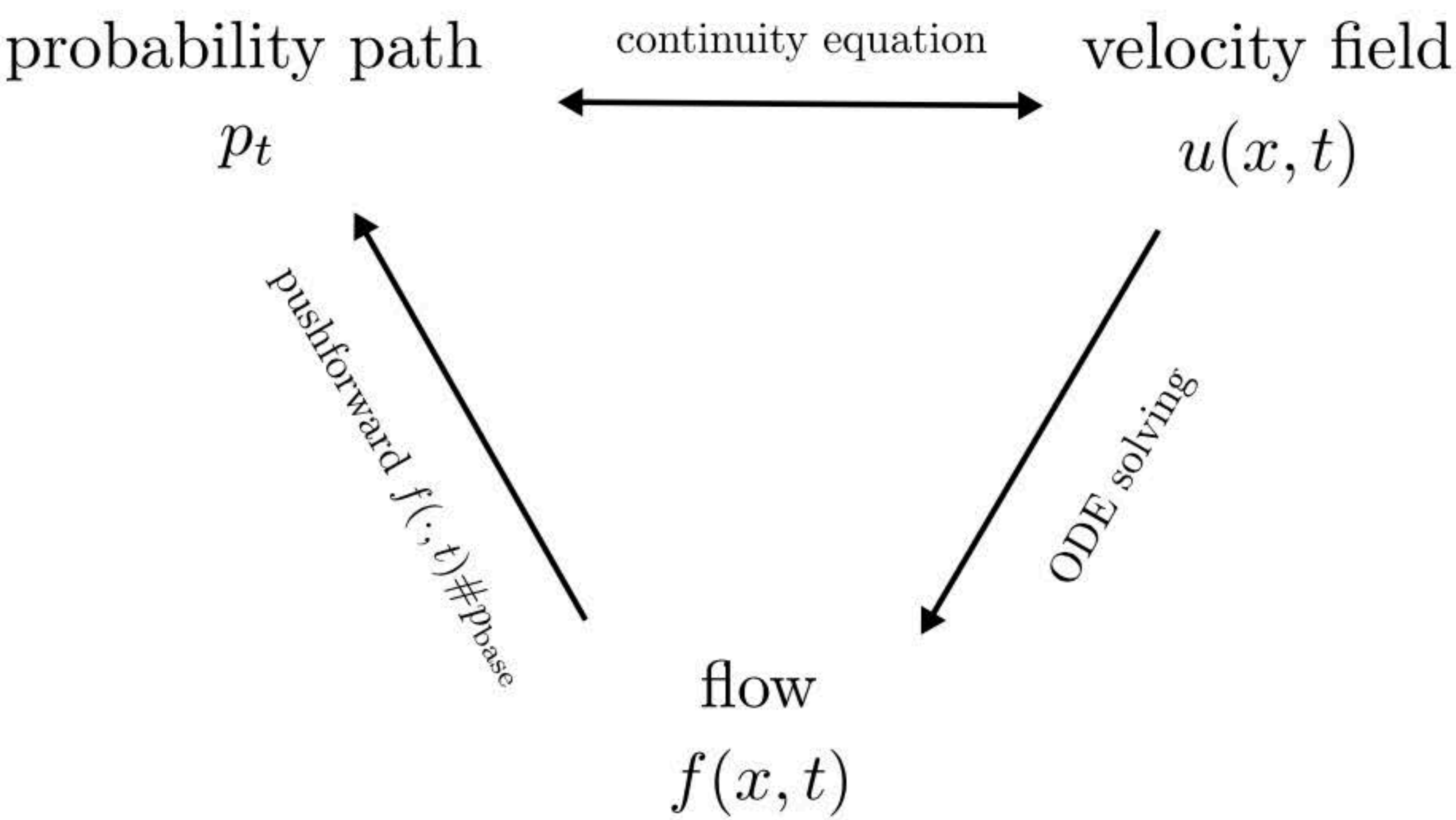


Figure 4. Link between the probability path, the velocity field and the flow.

CNF Pros and Cons

Pros

- The constraints one needs to impose on are much less stringent than in the discrete case: for the solution of the ODE to be unique, one only needs to be Lipschitz continuous in x and continuous in t
- Inverting the flow can be achieved by simply solving the ODE in reverse
- Computing the likelihood does not require inverting the flow, nor to compute a log determinant; only the trace of the Jacobian is required, that can be approximated using the Hutchinson trick.
- Adaptive time-step ODE solvers can automatically choose the number of steps and control the trade-off between sampling speed and approximation error.

Cons

- Training a neural ODE with log-likelihood does not scale well to high-dimensional spaces, and the process tends to be expensive and unstable, likely due to numerical approximations and to the (infinite) number of possible probability paths.