IV. Continuous Normalizing flows (CNF)

- For NF are based on the change-of-variables.

$$\begin{cases} x_N = \phi(z_0), \quad x_0 \xrightarrow{\phi_0} x_1 \xrightarrow{\phi_1} \cdots \xrightarrow{\phi_N} x_N \\[2ex] P_i(x_i) = P_{i-1}(x_{i-1}) \left| \operatorname{Det} \dfrac{\partial \phi_i}{\partial x_{i-1}} \right|^{-1} \Rightarrow P_N(x_N) = P_0(x_0) \displaystyle\prod_{i=1}^{N} \left| \operatorname{Det} \dfrac{\partial \phi_i}{\partial x_{i-1}} \right|^{-1} \end{cases}$$

P_N density is the "push forward" of P_0 under $f = f_N \circ \cdots \circ f_1$ "finite" flow

<ins>Drawbacks</ins>:
1) f has to be <ins>invertible</ins> bijection. this may be a limitation
2) full Jacobian needs to be computed $O(d^3)$: slow
3) can be solved by restricting Jacobian matrix $\left(0 \right) \sim O(d)$
this reduces expressibility...

we will see that continuous Normalizing flows (CNF) solve issues 2, 3)

[R. Chen et al. "Neural Ordinary differential equations" NeurIPS 2018.]

- consider "infinitesimal flows"

$$\phi = \phi_N \circ \cdots \circ \phi_1 \longrightarrow \phi = \phi_t, \quad t \in [0, 1]$$

"continuous time" replaces the flow index

auxiliary time

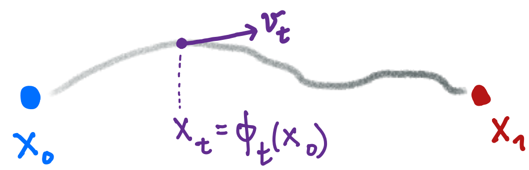
the point of CNF is to learn how to continuously evolve a simple distribution $q(z)$ into a complex one $p(z)$

- ϕ_t is a "trajectory" or "path" in time described by some dynamics

we can write down an ODE

$$\begin{cases} \dfrac{d\phi_t(x)}{dt} = \underbrace{v_t}_{\text{velocity field}}(\phi_t(x)) \\[2ex] \phi_0(x) = x \end{cases}$$

$\hookrightarrow \phi_0 = id$



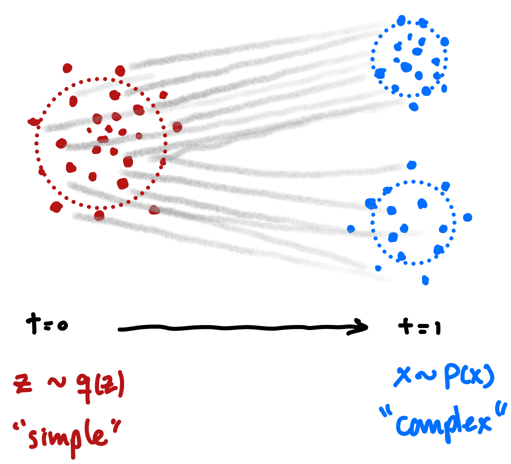
the solution to this ODE is straight forward:

$$x_t = x_0 + \int_0^1 dt\, v_t(x_t)$$

$\Rightarrow$ Probability densities are now time-dependent $p_t(x)$.

we refer to $p_t(x)$ as the "probability path" interpolating

$$\begin{cases} p_0 = q \\ p_1 = p \end{cases}$$



- Probabilities need to be conserved over time under the vector field v_t

$$\Rightarrow \boxed{\frac{\partial p_t(x)}{\partial t} + \underbrace{\nabla_x \cdot}_{\text{divergence}}\, j(x) = 0}$$

continuity equation or "Liouville" equation.

probability flux $j(x) \equiv p_t(x)\, v_t(x)$

$$\Rightarrow \quad \frac{\partial p_t(x)}{\partial t} = -\sum_{i=1}^{d} \frac{\partial}{\partial x_i}\left(v_t^i(x)\, p_t(x)\right)$$

← PDE give the evolution of the probability at a fixed point in space ⚠

- we are interested in the change in density along trajectories x_t: the total derivative is:

$$d p_t(x_t) = \frac{\partial p_t(x_t)}{\partial x_t}\, dx_t + \frac{\partial p_t(x_t)}{\partial t}\, dt$$

we can use the ODE for x_t and the continuity equation to get:

$$\begin{aligned}
\frac{d p_t(x_t)}{dt} &= \frac{\partial p_t(x_t)}{\partial x_t}\, v_t(x_t) - \nabla_x \cdot \left[v_t(x_t)\, p_t(x_t)\right] \\
&= \cancel{\nabla_x \cdot \left[v_t(x_t)\, p_t(x_t)\right]} - \left(\nabla_x \cdot v_t(x_t)\right) p_t(x_t) - \cancel{\nabla_x \cdot \left[v_t(x_t)\, p_t(x_t)\right]} \\
&= -p_t(x_t)\left(\nabla_x \cdot v_t(x_t)\right)
\end{aligned}$$

$$\Rightarrow \frac{d}{dt} \log p_t(x_t) = -\nabla_x \cdot v_t(x_t)$$

$$\boxed{\frac{d}{dt} \log p_t(x_t) = -\operatorname{Tr}\left[\frac{\partial v_t}{\partial x}\right]}$$

← Trace of the Jacobian matrix!

integrating yields the "instantaneous change-of-variables" formula:

$$\boxed{\log p_t(x_t) = \log p_0(x_0) - \int_0^1 dt\, \operatorname{Tr}\left[\frac{\partial v_t}{\partial x}\right]}$$

Notice that $\begin{cases} \boxed{\text{NF}} \longrightarrow \boxed{\text{CNF}} \\ \text{Det}\left(\dfrac{\partial v}{\partial x}\right) \longrightarrow \text{Tr}\left(\dfrac{\partial v}{\partial x}\right) \end{cases}$

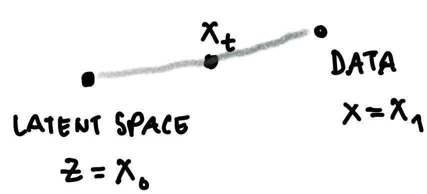
- Another way to understand why Det → Trace in the infinitesimal limit is that near the identity matrix the determinant behaves like the trace:

$$\det(\mathbb{1} + \varepsilon A) \simeq \mathbb{1} + \varepsilon \, \text{Tr}(A) + \mathcal{O}(\varepsilon^2) \quad \text{for} \quad \varepsilon \to 0 .$$

- CNF as Neural ODE (NODE) models

the CNF aims to model the continuous-time dynamical system that evolves a latent space distribution $q(z) = N(z|0,\mathbb{1})$ into the data with a "Neural ODE":

$$\boxed{\frac{d}{dt}\phi_t(x) = v_t^\theta(\phi_t(x))}$$



$$\begin{cases} p_0 = p_{base} = N(0,\mathbb{1}) \\ p_1 = p_{data} \end{cases}$$

where vector field v_t^θ is parametrized by a Neural Network.

- Solving the neural ODE yields the flow ϕ_1^θ called a time-one map.

this map transforms back and forth the data x into its latent rep. z:

$$\Rightarrow \begin{cases} x = \phi_1^\theta(z) = z + \int_0^1 dt \; v_t^\theta(x_t) \\ z = (\phi_1^\theta)^{-1}(x) = x - \int_0^1 dt \; v_t^\theta(x_t) \; \text{........ inverse} \end{cases}$$

Remark 1:

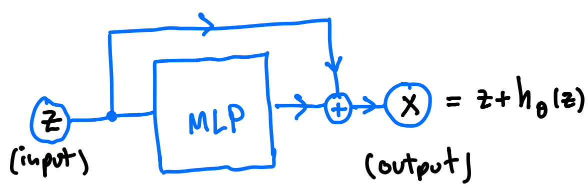
Unlike autoregressive NF, computing the inverse map for a CNF has same complexity as forward direction!

Remark 2: Neural ODE can be thought as an infinitely deep neural network. this can be seen by solving the ODE via Euler method:

$$x_t - x_{t-1} = \varepsilon\, v_t^\theta(x_n) \quad , \varepsilon \ll 1$$

this has the same form as a Residual Neural Network (ResNet) with a block $h_\theta = \varepsilon v_t^\theta$.

[R. Chen et al. NeurIPS 2018]



(ResNet)

- Training CNF's

As with NF, we can train the CNF using Maximum likelihood Estimation

$$\boxed{\mathcal{L}(x,\theta) = \log p_\theta(x) = \log p_{base}(z) - \int_0^1 dt\, \mathrm{Tr}\left[\frac{\partial v_t^\theta}{\partial x_t}\right]}$$

Remark 3: computing the likelihood does not involve computing $\mathrm{Det}(J) \sim \mathcal{O}(d^3)$ but instead $\mathrm{Tr}[J]$ which has a better complexity of $\mathcal{O}(d)$!

$\Rightarrow$ CNF allow for free-form Jacobian matrices

No restriction on architecture like for MAF or Coupling flows.

$\longrightarrow$ More expressive

- In practice we need to optimize $\mathcal{L}(\theta) \Rightarrow$ Backpropagate through a numeric ODE solution implemented via some algorithm:

$$\mathcal{L}(x,\theta) = \text{ODEsolve}(x,\theta) \qquad \nabla_\theta \mathcal{L}(x,\theta) = ?$$

compute gradient

$\Rightarrow$ Formulate gradient computation as a separate ODE in $\frac{\partial \mathcal{L}}{\partial x_t} \equiv a_t$ known as adjoint ODE it has solution:

$$\boxed{\nabla_\theta \mathcal{L} = -\int_1^0 dt \; a_t^T \cdot \frac{\partial v_t^\theta}{\partial \theta}}$$

solving this adjoint ODE backwards in time $t_1 \to t_0$

- Fast trace computation:

One can further improve the complexity of $Tr[J]$ by using Hutchinson's trace estimator:

suppose ε is a noise vector such that $\begin{cases} \mathbb{E}(\varepsilon) = 0 \\ Cov(\varepsilon) = \mathbb{1}_{d \times d} \end{cases}$ e.g. $\varepsilon \sim N(0, \mathbb{1})$

$$Tr[A] = Tr[A \, \mathbb{E}(\varepsilon^T \varepsilon)] = \mathbb{E} \, Tr[\underbrace{\varepsilon^T A \varepsilon}_{\#}] = \mathbb{E}_{\varepsilon \sim P(\varepsilon)} [\varepsilon^T A \varepsilon]$$

$$\boxed{Tr[A] \approx \frac{1}{M} \sum_{i=1}^{M} \varepsilon_i^T A \varepsilon_i}$$

Monte Carlo

this is a stochastic estimator that and scales better!

exercise: show that $\mathbb{E}\left(\frac{1}{M}\sum_{i=1}^{M}\varepsilon_i^T A \varepsilon_i\right) = \mathrm{Tr}(A)$ i.e. estimation is unbiased

<u>caveat</u>: the jacobian matrix of the CNF flows are <u>not</u> fully <u>free-form</u>

in fact they happen to be positive definite matrices

i.e. all eigenvalues are positive. E.g. one can't write down

a function $f(x) = -x$ as a $\phi_1(x)$ time-one map

⇝ way out Augmented NODE (ANODE)

embed the data in higher dim space...

this lifts the topological obstruction!

- <u>issues with CNFs</u>

Training and Sampling the CNF model requires solving the neural ODE, in the forward or backward direction, using Numerical ODE solvers

⇝ based on the Euler, Runge-Kutta, etc methods

→ expensive, needs many time-steps in order to give accurate results

→ numerically unstable

→ In practice CNFs don't scale well to large datasets

MORAL OF STORY: WHATEVER IS GAINED IN EXPRESSIBILITY OVER finite NF is lost IN PRACTICE!

V. Flow-matching [Lipman et al. ICLM 2023] NEW

- Flow-matching is simple training objective for CNF's that allows for scalable training ⇝ scales beter than MLE objective and more stable...

$$\begin{cases} \dfrac{d\phi_t(x)}{dt} = u_t(\phi_t(x)) \quad , \phi_0 = id \qquad (\phi_0(x) = x) \\ \\ \dfrac{\partial p_t(x)}{\partial t} = -\nabla_x (u_t(x)\, p_t(x)) \end{cases}$$

- we say that u_t generates the prob. paths $p_t(x)$ if the above eq. are satisfied.

the idea is to directly regress the velocity field u_t with an MSE loss

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

flow-matching! objective

Neural Network

$t \sim U[0,1]$ is the uniform distribution.

Huge benefit: No need to solve ODE during training u_t^θ which usually requires going sequentially through time-steps (e.g. Euler method)

Here time can be sampled non-sequentially (Parallelized)...

• problem: How do we model the prob. path $p_t(x)$? what to take for u_t?

we only know that $\begin{cases} p_0(x) = p_{base} = N(0,\mathbb{1}) \\ p_1(x) = p_{data} \end{cases}$ I

• solution? Model simpler conditional/joint quantities that when marginalized give you back the quantities you are interested in.

⇝ leads to "integral representations"!

Define conditional probability path $p_t(x|y)$ conditioned on some random variable y such that:

$$\begin{cases} p_0(x|y) = p_{base}(x) = N(0,\mathbb{1}) & \text{(independent of } y) \\ p_1(x|y) = \delta(x-y) \end{cases}$$

the conditional prob. path $p_t(x|y)$ interpolates between the std Gaussian at $t=0$ and the delta function centered around "y" at $t=1$.

• Marginalizing over the data distribution:

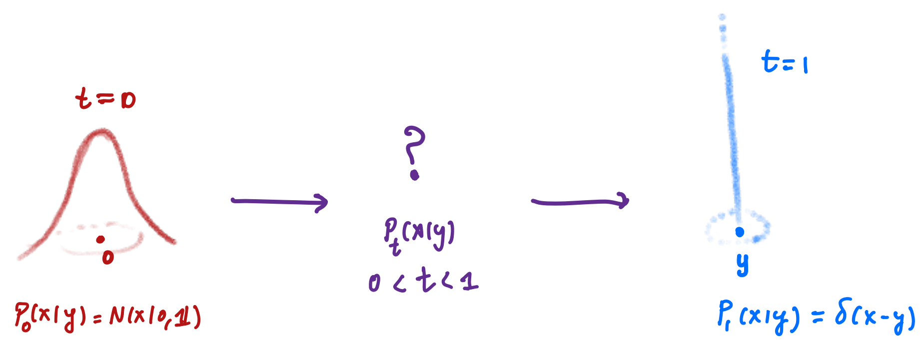
$$p_t(x) = \int dx_1\, p_t(x|x_1)\, p_{data}(x_1)$$

gives the correct boundary conditions I above:

$$
\begin{cases}
\bullet\ t=0: & p_0(x) = \int dx_1\, p_{base}(x)\, p_{data}(x_1) \\
 & \quad = p_{base}(x) \cdot \underbrace{\int dx_1\, p_{data}(x_1)}_{1} = p_{base}(x) \\
\bullet\ t=1: & p_1(x) = \int dx_1\, \delta(x-x_1)\, p_{data}(x_1) \\
 & \quad = p_{data}(x)
\end{cases}
$$

- **Gaussian conditional probability paths:**

- Notice that the conditional paths $p_t(x|y)$ are easier to model



- the most natural choice is to take a Gaussian interpolation, since the dirac delta is recoverd as the narrow width of a Gaussian.

$$\boxed{p_t(x|y) = \mathcal{N}(x \,|\, \gamma_t\, y\,,\ \sigma_t^2 \mathbb{1})}$$

Gaussian conditional probabilty path!

γ_t, σ_t are functions that satisfy $\begin{cases} (\gamma_0, \sigma_0) = (0,1) \\ (\gamma_1, \sigma_1) = (1,0) \end{cases}$

Remark: Diffusion models, explained later in the course, also lead to Gaussian conditional prob. paths with particular choices of the mean γ_t and covariance σ_t ...

- Sampling $x_t \sim P_t(x|x_1)$ yields a cond. trajectory of the form

$$\boxed{x_t = \gamma_t x_1 + \sigma_t \varepsilon}, \quad \varepsilon \sim \mathcal{N}(0, \mathbb{1}) \quad \text{(Reparam trick)}$$
$$= P_{base}$$

- the conditional vector field can also be computed:

$$\boxed{u_t(x|x_1) = \dot{\gamma}_t x_1 + \dot{\sigma}_t \varepsilon} \leftarrow \text{Recall that } \dot{x}_t = u_t(x|x_1)$$

- one can show that the vector field v_t that generates $P_t(x|$ can be represented by:

$$\boxed{u_t(x) = \int dx_1\, u_t(x|x_1)\, \frac{p_t(x|x_1)\, p_1(x_1)}{p_t(x)}}$$

i.e. aggregation of conditional vector fields $u_t(x|y)$ that generate $p_t(x|y)$ via the continuity equation.

→ Unfortunately, $u_t(x)$ can't be integrated. We still don't know the denominator $p_t(x)$ .... back to square 1?..

- Conditional Flow-matching to the rescue:

Instead of $\mathcal{L}_{FM}$, consider regressing the conditional vector field

$$\boxed{\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{\substack{t\sim U[0,1] \\ x\sim p_t(x|x_1)}} \left\| u_t^\theta(x) - u_t(x|x_1) \right\|^2}$$

THEOREM: minimizing the objective $\mathcal{L}_{CFM}^\theta$ is equivalent to minimizing $\mathcal{L}_{FM}^\theta$!

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

- this is another comon trick in ML: if a Loss is intractable cook up a simpler loss that has the same minima!

- Notice that if we assume Gaussian conditional probability paths, where we specify γ_t and σ_t, then $\mathcal{L}_{CFM}(\theta)$ is fully computable!

- <u>the simplest model</u>: conditional optimal transport

take: $\begin{cases} \gamma_t = t \\ \sigma_t = (1-t) \end{cases}$

$$\Rightarrow \begin{cases} p_t(x|x_1) = \mathcal{N}(x \,|\, t x_1, (1-t)^2) \\ u_t(x|x_1) = x_1 - \varepsilon \end{cases}$$

straight line for conditional vector field with constant speed!

$$\boxed{\mathcal{L}_{CFM}(\theta) = \mathop{\mathbb{E}}_{\substack{t \sim U[0,1] \\ \varepsilon \sim N(0,\mathbb{1}) \\ x_1 \sim p_{data}}} \left\| u_t^\theta(x) - \varepsilon - x_1 \right\|^2}$$

Remark:
- Using the CFM objective for training is fast!
- But once we learn the conditional vector field in order to sample from the CNF we still need to solve the NODE... ← slow...

NEXT TOPICS?

- Going beyond Gaussian base distribution:
- Optimal Transport Flow-matching
- Diffusion models