IV. Continuous Normalizing Flows (CNF)

- For NF are based on the change of variables.

\[ \begin{align*}
X_1 &= \phi(X_0), \\
X_1 &= \phi_0 \rightarrow \phi_1 \rightarrow \cdots \rightarrow \phi_N \rightarrow X_N \\
\text{\(P_N(X_i) = P_{i-1}(X_{i-1}) \left| \det \frac{\partial \phi_i}{\partial X_{i-1}} \right|^{-1} \Rightarrow P_N(X_N) = P_0(X_0) \prod_{i=1}^{N} \left| \det \frac{\partial \phi_i}{\partial X_{i-1}} \right|^{-1} \right.} \\
\end{align*} \]

\( p_N \) density is the "pushforward" of \( p_o \) under \( f = f_N \circ \cdots \circ f_1 \) "finite" flow

Drawbacks:
1) \( f \) has to be invertible bijection. this may be a limitation
2) full Jacobian needs to be computed \( \Theta(d^3) \): slow
3) can be solved by restricting Jacobian matrix \( \Theta(d) \)
   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"

\[ \phi = \phi_N \circ \cdots \circ \phi_1 \rightarrow \phi = \phi_t, t \in [0, 1] "\text{continuous time}\" \text{ replaces the flow index} \]

the point of CNF is to learn how to continuously evolve a simple distribution gets into a complex one \( p(t) \)
• $\phi_t$ is a "trajectory" or "path" in time described by some dynamics, we can write down an ODE

\[
\begin{cases}
\frac{d\phi_t(x)}{dt} = v_t(\phi_t(x)) \\
\phi_0(x) = x
\end{cases}
\]

such that $\phi_t = \text{id}$

the solution to this ODE is straightforward:

\[x_t = x_0 + \int_0^t v_t(x_t) dt\]

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

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

\[
\begin{cases}
p_0 = q \\
p_1 = p
\end{cases}
\]

$\Rightarrow$ Probabilities need to be conserved over time under the vector field $v_t$.

\[\begin{aligned}
\frac{\partial p_t(x)}{\partial t} + \nabla_x \cdot \dot{p}_t(x) &= 0 \\
\text{continuity equation} &\quad \text{or "Lovelace" equation}
\end{aligned}\]

\[\dot{p}_t(x) = p_t(x) v_t(x)\]
\[
\frac{\partial P_t(x)}{\partial t} = -\sum_{\lambda=1}^d \frac{\partial}{\partial x_{\lambda}} \left( v_t^\lambda(x) P_t(x) \right) \quad \text{PDE give the evolution of the probability at a fixed point in space} \]

We are interested in the change in density along trajectories \( X_t \): the total derivative is:

\[
dP_t(x_t) = \frac{\partial P_t(x_t)}{\partial x_t} \, dx_t + \frac{\partial P_t(x_t)}{\partial t} \, dt
\]

We can use the ODE for \( X_t \) and the continuity equation to get:

\[
\frac{d P_t(x_t)}{dt} = \frac{\partial P_t(x_t)}{\partial x_t} \, v_t(x_t) - \nabla_x \left[ v_t(x_t) \, P_t(x_t) \right]
\]

\[
= \nabla_x \left[ v_t(x_t) \, P_t(x_t) \right] - (\nabla_x \cdot v_t(x_t)) \, P_t(x_t) - \nabla_x \left[ v_t(x_t) \, P_t(x_t) \right]
\]

\[
= -P_t(x_t) \left( \nabla_x \cdot v_t(x_t) \right)
\]

\[
\Rightarrow \quad \frac{d}{dt} \log P_t(x_t) = -\nabla_x \cdot v_t(x_t)
\]

\[
\frac{d}{dt} \log P_t(x_t) = -\text{Tr} \left[ \frac{\partial v_t}{\partial x} \right] \quad \text{Trace of the Jacobian matrix} !
\]

Integrating yields the "instantaneous change of variables" formula:

\[
\log P_t(x_t) = \log P_0(x_0) - \int_0^1 dt \, \text{Tr} \left[ \frac{\partial v_t}{\partial x} \right]
\]
Notice that \[
\begin{align*}
\begin{array}{ccc}
\text{CNF} & \rightarrow & \text{CNF} \\
\text{Det} \left( \frac{\partial \mathbf{v}}{\partial \mathbf{x}} \right) & \rightarrow & \text{Tr} \left( \frac{\partial \mathbf{v}}{\partial \mathbf{x}} \right)
\end{array}
\end{align*}
\]

* Another way to understand why \( \text{Det} \rightarrow \text{Trace} \) in the infinitesimal limit is that near the identity matrix the determinant behaves like the trace:

\[
\det \left( 1 + \varepsilon \mathbf{A} \right) = 1 + \varepsilon \text{Tr}(\mathbf{A}) + \mathcal{O}(\varepsilon^2) \quad \text{for } \varepsilon \to 0.
\]

* **CNF as Neural ODE (NODE) models**

The CNF aims to model the continuous-time dynamical system that evolves a latent space distribution \( p(\mathbf{z}) = \mathcal{N}(\mathbf{z} | 0, \mathbf{I}) \)

into the data with a "Neural ODE":

\[
\begin{align*}
\frac{d}{dt} \phi_t(\mathbf{x}) &= \mathbf{u}_t^\mathbf{\theta}(\phi_t(\mathbf{x})) \\
\text{LATENT SPACE} & \quad \mathbf{z} = \mathbf{x}_0 \\
\text{DATA} & \quad \mathbf{x} = \mathbf{x}_1 \\
\end{align*}
\]

where \( \mathbf{u}_t^\mathbf{\theta} \) is parametrized by a Neural Network.

* Solving the neural ODE yields the flow \( \phi_t^\mathbf{\theta} \) called a time-one map.

This map transforms back and forth the data \( \mathbf{x} \) into its latent rep. \( \mathbf{z} \):

\[
\Rightarrow \begin{cases}
\mathbf{x} = \phi_1^\mathbf{\theta}(\mathbf{z}) &= \mathbf{z} + \int_0^1 dt \ \mathbf{u}_t^\mathbf{\theta}(\mathbf{x}_t) \\
\mathbf{z} = (\phi_1^\mathbf{\theta})^{-1} &\mathbf{x} = \mathbf{x} - \int_0^1 dt \ \mathbf{u}_t^\mathbf{\theta}(\mathbf{x}_t) \quad \ldots \quad \text{inverse}
\end{cases}
\]
Remark 1: Unlike autoregressive NF, computing the inverse map for a CNF has the same complexity as forward direction.

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

\[ X_t - X_{t-1} = \mathcal{E} V_t^\theta (X_{t-1}), \quad \mathcal{E} \in \mathbb{R} \]

This has the same form as a Residual Neural Network (ResNet) with a block \( h_\theta = \mathcal{E} V_t^\theta \).

\[ \text{(ResNet)} \]

Training CNF's:

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

\[ \mathcal{L}(x, \theta) = \log p(x) = \log \lambda (x, \theta) - \int_0^1 \text{Tr} \left[ \frac{\partial \mathcal{E} V_t^\theta}{\partial X_t} \right] \]

\[ \Rightarrow \text{CNF allow for free-form Jacobian matrices} \]
No restriction on architecture like for MAF or coupling flows.

→ more expressive

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

$$\mathcal{L}(x; \theta) = \text{ODEsolve}(x, \theta) \quad \nabla_{\theta} \mathcal{L}(x, \theta) = \text{compute gradient}$$

- Formulate gradient computation as a separate ODE in $\frac{dx}{dt} = a_{\theta}$ known as adjoint ODE it has solution:

$$\nabla_{\theta} x = -\int_{t}^{0} dt \ a_{\theta}^T \cdot \frac{\partial x}{\partial \theta}$$

solving this adjoint ODE backwards in time $t_{f} \rightarrow t_{0}$

- Fast trace computation:

  one can further improve the complexity of $\text{Tr}[\mathcal{J}]$
  by using Hutchinson's trace estimator:

  suppose $\varepsilon$ is a noise vector such that $\{ \begin{array}{c} E(\varepsilon) = 0 \\
  \text{Cov}(\varepsilon) = I_{d \times d} \end{array} \}$

  $$\text{Tr}[\mathcal{J}] = \text{Tr}[\mathcal{A} E(E^{T}E)] = \mathbb{E} \text{Tr}[E^{T}A \varepsilon \varepsilon^{T}] = \mathbb{E}_{\varepsilon \sim \mathcal{N}(0, I)} \left[ \varepsilon^{T}A \varepsilon \right]$$

  $$\text{Tr}[\mathcal{A}] \approx \frac{1}{M} \sum_{i=1}^{M} E_{i}^{T} A \varepsilon_{i} \quad \text{Monte Carlo}$$
this is a stochastic estimator that and scales better!

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

\[ \text{caveat: } \] the jacobian matrix of the CNF flows are not fully free-form 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 \( \phi_t(x) \) time-one map

\[ \rightarrow \] way out Augmented NODE (ANODE)

\[ \text{embed the data in higher dim space...} \]

\[ \text{this lifts the topological obstruction!} \]

- **issues with CNFs**

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

\[ \rightarrow \] based on the Euler, Runge-Kutta, etc methods

\[ \rightarrow \] expensive, needs many time-steps in order to give accurate results

\[ \rightarrow \] numerically unstable

\[ \rightarrow \] 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!
Flow-matching is a simple training objective for CNF's that allows for scalable training — scales better than MLE objective and more stable...

\[
\begin{align*}
\frac{d\Phi_t(x)}{dt} &= U_t(\Phi_t(x)), \quad \Phi_0 = \text{id} \quad (\Phi_0(x) = x) \\
\frac{\partial P_t(x)}{\partial t} &= -\nabla_x (U_t(x) P_t(x))
\end{align*}
\]

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.

\[
L_{FM} = \mathbb{E}_{t \sim U[0,1], x \sim P_t(x)} \| U_t^\theta(x) - U_t(x) \|^2
\]

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

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

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

[Neural Network]
Problem: How do we model the prob. path $p_t(x)$? what to take for $u_t$?

We only know that:

\[
\begin{align*}
    p_0(x) &= P_{\text{base}} = N(0,1) \\
    p_1(x) &= p_{\text{data}}
\end{align*}
\]

Solution: Model simpler conditional/joint quantities that when marginalized give you back the quantities you are interested in.

This leads to "integral representations!"

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

\[
\begin{align*}
    p_0(x|y) &= P_{\text{base}}(x) = N(0,1) \quad \text{(independent of $y$)} \\
    p_1(x|y) &= \delta(x-y)
\end{align*}
\]

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:

\[
p_t(x) = \int dx_1 p_t(x|x_1) p_{\text{data}}(x_1)
\]

gives the correct boundary conditions above.
\[ t=0 : \quad p_0(x) = \int dx_1 \ p_{\text{base}}(x) \ p_{\text{data}}(x_1) = p_{\text{base}}(x) \]
\[ t=1 : \quad p_1(x) = \int dx_1 \ \delta(x-x_1) \ p_{\text{data}}(x_1) = p_{\text{data}}(x) \]

- **Gaussian conditional probability paths**

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

\[ p_t(x|y) = N(x|\delta_t y, \delta_t^2 I) \]

- The most natural choice is to take a Gaussian interpolation, since the Dirac delta is recovered as the narrow width of a Gaussian.

\[ \delta_t, \delta_t \text{ are functions that satisfy } \left\{ \begin{array}{l} (\delta_0, \sigma_0) = (0, 1) \\ (\delta_1, \sigma_1) = (1, 0) \end{array} \right. \]
Remark: Diffusion models, explained later in the course, also lead to Gaussian conditional prob. paths with particular choice of the mean $y_t$ and covariance $\sigma_t$...

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

$$x_t = y_t x_t + \sigma_t \epsilon, \quad \epsilon \sim N(0, I) \quad \text{(reparam trick)}$$

- the conditional vector field can also be computed:

$$u_t(x|x_t) = \dot{y}_t x_t + \sigma_t \epsilon$$

- one can show that the vector field $u_t$ that generates $p_t(x|x)$ can be represented by:
\[ u_t(x) = \int dx_i u_t(x(x_i)) \frac{P_t(x(x_i)) P_t(x)}{p_t(x)} \]

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

\[ \Rightarrow \text{Unfortunately, } u_t(x|x_i) \text{ can't be integrated. We still don't know the denominator } p_t(x) \ldots \text{ back to square 1?} \]

• Conditional Flow-matching to the rescue:

Instead of \( L_{FM} \), consider regressing the conditional vector field

\[ L_{CFM}(\theta) = \mathbb{E}_{x \sim p_t(x|x_i)} \left\| u_{t}(x) - u_{t}(x|x_i) \right\| ^2 \]

**Theorem:** Minimizing the objective \( L_{CFM}(\theta) \) is equivalent to minimizing \( L_{FM}(\theta) \).
\[ \nabla_\theta \mathcal{L}_{FM}(\theta) = \nabla_\theta \mathcal{L}_{CFM}(\theta) + \text{const.} \]

- This is another common trick in ML: if a loss is intractable, cook up a simpler loss that has the same minimum!

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

- the simplest model: conditional optimal transport

\[
\begin{align*}
\text{take:} & \quad \left\{ \begin{array}{l}
x_t = t \\
\sigma_t = (1-t)
\end{array} \right. \\
\Rightarrow & \quad \begin{cases}
P_t(x | x_i) = \mathcal{N}(x / t x_i, (1-t)^2) \\
\mathcal{U}_t(x | x_i) = x_i - \tilde{\varepsilon} \quad \text{a straight line for conditional vector field,} \\
\quad \text{with constant speed!}
\end{cases}
\end{align*}
\]

\[ \mathcal{L}_{CFM}(\theta) = \mathbb{E}_{\tilde{\varepsilon} \sim \mathcal{N}(0,\sigma)} \left\| \mathcal{U}^\theta_t(x) - \tilde{\varepsilon} - x_i \right\|^2 \]
Remarks: 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