Flow Matching for Generative Modeling

Pramook Khungurn

July 22, 2024

This note was written as I read the "Flow Matching for Generative Modeling" paper by Lipman et al. [5].

1 Background

- A data item is denoted by $x = (x^1, x^2, \dots, x^d) \in \mathbb{R}^d$.
- A **probability density path** is a function $p : [0,1] \times \mathbb{R}^d \to \mathbb{R}^+ \cup \{0\}$ such that each $p(t, \cdot)$ is a probabilty density function on $\mathbb{R}^d$. In other words, it holds that
$$\int p(t,x)\,\mathrm{d}x = 1$$
for all $t \in [0,1]$.
- For a time dependent function $f : [0,1] \times \mathbb{R}^d \to R$ for some range set R, we may write $f(t,x)$ as $f_t(x)$ to emphasize time dependence. Moreover, we can refer to $f_t : \mathbb{R}^d \to R$ as a function in its own right.
  - With this, we may say that p_t is a probability distribution on $\mathbb{R}^d$.
- A **time-dependent vector field** is a function $v : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$.
- Given a time dependent vector field v, its **flow** is another vector field $\phi : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ defined by the ordinary differential equation
$$\frac{\mathrm{d}}{\mathrm{d}t}\phi_t(x) = v_t(\phi_t(x))$$
and the initial condition $\phi_0(x) = x$.
  - In other words, $\phi_t(x)$ is the position at time t of the particle that starts at x at time 0 and follows the trajectory defined by taking v as the time-dependent vector field.
- Chen et al. proposed the **neural ordinary differential equation**model [1]. The idea is to model the vector field v with a neural network $v_t(x;\theta)$. We then train it so that ϕ_1 has the property that we want.
  - If you want a refresher on neural ODE, then read my previous note on it [2].
- A neural ODE can be used to transform a probability distribution to another. Say, we start with a probability distribution p_0 on $\mathbb{R}^d$. Then, we do the following.
  - Sample $x \sim p_0$.
  - Compute $x' = \phi_t(x)$ by integrating the neural ODE from 0 up to t.

Let us denote the probability density of x' by p_t. It follows thet

$$p_t(x') = p_0(\phi_t^{-1}(x')) \det \left[\frac{\partial \phi_t^{-1}}{\partial x}(x') \right]. \tag{1}$$

This is the standard formula for tranformation of probability distribution. You can find this in section 3.1 on my notes on the subject [4].

- The formula in Equation (1) is not that great because there is an issue with variable capture. The x in ∂x is not a variable but a shorthard the positiional argument of a function. I previously have introduced a system to deal with this kind of problem [3]. So, let's write the equation using that notation.

  First, we note that $\phi_t(x) = q(t, x)$ is a function that maps a $(d+1)$-dimensional space to a d-dimensional space. So, we can treat it in the same way as a function of signature $\mathbb{R}^{d+1} \to \mathbb{R}^d$. In other words, we can say that ϕ takes $d + 1$ inputs. We can then divide the $d + 1$ inputs into two blocks.

  - The first block is the first argument alone. Using Python slice notation, it is "$1:2$.". Using my "chapter" notation, it can be abbreviated as §1.
  - The second block is the rest of the arguments. Using Python slice notation, it is "$2:d+2$." Using my "chapter" notation, it can be abbreviate as §2.

  Hence, using my notation for partial derivatives, we can rewrite the equation as:

  $$p_t(x') = p_0(\phi_t^{-1}(x')) \det \nabla_{\S 2} \phi_t^{-1}(x')$$

  or, to be even briefer

  $$p_t(x') = p_0(\phi_t^{-1}(x')) \, |\nabla_{\S 2} \phi_t^{-1}(x')|$$

- Let $f : \mathbb{R}^d \to \mathbb{R}$ and let $v : \mathbb{R}^d \to \mathbb{R}^d$. A **push-foward** (or a change of variable) of f according to v is a function of $g : \mathbb{R}^d \to \mathbb{R}$ defined by

  $$g(y) = f(v^{-1}(y)) \, |\nabla v^{-1}(y)|.$$

  Here, ∇ denotes the derivative operator, which gets you the Jacobian matrix. We denote the push-forward of f according to v as $[v]_* f$.

- In the context of the discussion so far, we have that $p_t = [\phi_t]_* p_0$.
- When we use a neural ODE to transform a probability distribution from one to another (i.e., transforming p_0 from p_1), we call the resulting model a **continuous normalizing flow** model.

2 Flow Matching

2.1 Flow Matching Objective

- We want to use the above framework to transform a simple noise distribution $p_0 = p_{\text{noise}}$ to a data distribution $p_1 = p_{\text{data}}$.
  - p_{noise} is typically a Gaussian distribution $p_0 = \mathcal{N}(0, I)$.
  - As in most ML settings, we do not have access to the density function p_{data}, but we only have samples from the distribution.
- Suppose we know a probability path p_t and a time-dependent vector field u_t that has the following property:

- p_0 is the desired noise distribution, and p_1 is the desired data distribution.
- u_t is the vector field such that $p_t = [u_t]_* p_0$.

Suppose again that we want to model u_t with a neural network $v_t(x;\theta)$. Then, we may do it my minimizing the **flow matching objective**:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = E_{t\ \mathcal{U}([0,1]), x \sim p_t} \big[\| u_t(x) - v_t(x;\theta) \|^2 \big].$$

- The flow maching objective is usable if we know p_t and u_t before hand. However, in our settings, we do not know anything about u_t, and we only know $p_0 = p_{\mathrm{noise}}$ and $p_1 = p_{\mathrm{data}}$ but nothing in between.

2.2 Rewriting condtional paths and vector fields

- We still do not know what p_t exactly is, but let us engage in wishful thinking and try to dictate its form.
- Let $x_1 \sim p_{\mathrm{data}}$ be a data item. We look at the conditional probability density $p_t(x|x_1)$. Let us require that
  1. $p_0(x|x_1) = p_{\mathrm{noise}}(x)$, and
  2. $p_1(x|x_1) = \mathcal{N}(x; x_1, \sigma^2 I)$ where σ is a small number.
- Now, we have that

$$p_t(x) = \int p_t(x|x_1) p_{\mathrm{data}}(x_1)\,\mathrm{d}x_1.$$

Moreover, if we choose σ to be small enough, we would have that

$$p_1(x) \approx p_{\mathrm{data}}(x).$$

-

References

[1] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations, 2019.

[2] Khungurn, P. Neural ordinary differential equations. `https://pkhungurn.github.io/notes/notes/ml/neural-ode/neural-ode.pdf`. Accessed: 2024-07-19.

[3] Khungurn, P. Notation for multivariable derivatives. `https://pkhungurn.github.io/notes/notes/math/multivar-deriv-notations/multivar-deriv-notations.pdf`. Accessed: 2024-07-22.

[4] Khungurn, P. Probability under transformation. `https://pkhungurn.github.io/notes/notes/gfx/pdf-transform/pdf-transform.pdf`. Accessed: 2024-07-22.

[5] Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., and Le, M. Flow matching for generative modeling, 2023.