

Flow Matching: 从数学原理到完整推导

逐步理解速度场、连续性方程、条件流匹配及机器人动作生成

2026 年 10 月

目录

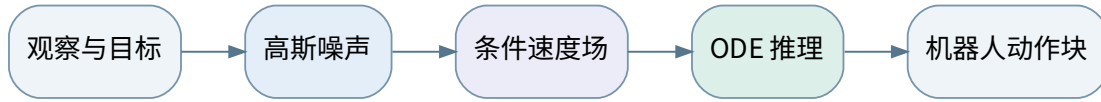
一、第一步：为什么需要 Flow Matching?	2
二、第二步：为什么引入 ODE?	3
三、第三步：为什么需要连续性方程?	4
四、第四步：Flow Matching 的理想目标是什么?	5
五、第五步：Conditional Flow Matching 的构造	6
六、第六步：为什么条件速度回归能够得到全局速度场?	7
七、第七步：为什么 MSE 能学到条件期望?	9
八、第八步：为什么还要引入 Gaussian Conditional Path?	10
九、第九步：Flow Matching 实际如何训练和生成?	12
十、第十步：为什么 Flow Matching 能生成多模态分布?	13
十一、第十一步： $\pi_{0.5}$ 如何使用 Flow Matching?	14
十二、第十二步：Flow Matching 如何用于强化学习微调?	15
十三、总结：Flow Matching 的完整数学逻辑	18

课程首页 · English · PDF

项目	说明
适用范围	CFM 理论; OpenPI pi0.py 代码引用固定如下; 示例不代表已完成 微调实验
文档核验日期	2026-10-18
许可	CC BY-NC-SA 4.0 · Xiong Zeng and contributors

一、第一步：为什么需要 Flow Matching?

本章知识地图



1.1 生成模型究竟要解决什么问题?

假设训练数据来自未知分布:

$$x \sim p_{\text{data}}(x).$$

例如：图像生成时， x 是一张图片；视频生成时， x 是一段视频；机器人控制时， x 可以是一个 Action Chunk。

目标是学习生成模型，使得：

$$x_{\text{generated}} \sim p_{\text{data}}(x).$$

真实数据分布通常十分复杂。例如，机器人抓取杯子可能有左侧抓取、右侧抓取两种策略，对应多模态分布：

$$p_{\text{data}}(x) = \frac{1}{2}p_{\text{left}}(x) + \frac{1}{2}p_{\text{right}}(x).$$

如果直接训练确定性网络 $f_{\theta}(o) \rightarrow x$ ，使用平方损失：

$$\mathcal{L}(\theta) = \mathbb{E} [\|f_{\theta}(o) - x\|^2],$$

其理论最优解是条件均值：

$$f_{\theta}^*(o) = \mathbb{E}[x \mid o].$$

这个均值可能既不是左侧抓取，也不是右侧抓取。

动机：我们希望学习整个数据分布，而不只是预测其均值。

1.2 从容易采样的高斯分布开始

从标准高斯分布采样非常简单：

$$x_0 \sim p_0(x) = \mathcal{N}(0, I).$$

因此可以换一种思路：不直接预测真实数据，而是学习一个变换

$$F_{\theta} : \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad x_1 = F_{\theta}(x_0),$$

使得

$$(F_\theta)_\# p_0 = p_{\text{data}}.$$

其中 $\#$ 表示 **Pushforward (推前分布)**。直观地讲，将许多高斯采样点通过 F_θ 后，输出点的总体分布应当与真实数据分布一致：

$$\underbrace{\mathcal{N}(0, I)}_{\text{简单高斯分布}} \xrightarrow{F_\theta} \underbrace{p_{\text{data}}}_{\text{复杂多模态分布}}.$$

关键问题变成：怎样学习这个分布变换？

二、第二步：为什么引入 ODE?

2.1 将复杂变换拆成连续的小变换

直接学习 $x_1 = F_\theta(x_0)$ 可能很困难。为此，引入连续的**虚拟生成时间**：

$$t \in [0, 1],$$

约定 $t = 0$ 时对应高斯分布， $t = 1$ 时对应真实数据分布：

$$t = 0 : x_0 \sim p_0, \quad t = 1 : x_1 \sim p_{\text{data}}.$$

在两个端点之间，样本沿连续轨迹移动：

$$x_0 \longrightarrow x_t \longrightarrow x_1.$$

定义时间相关的速度场：

$$v_\theta(x, t) : \mathbb{R}^d \times [0, 1] \rightarrow \mathbb{R}^d,$$

并建立常微分方程 (ODE)：

$$\frac{dx_t}{dt} = v_\theta(x_t, t). \quad (1)$$

初始条件为 $x_0 \sim \mathcal{N}(0, I)$ 。积分得到：

$$x_1 = x_0 + \int_0^1 v_\theta(x_t, t) dt.$$

动机：神经网络不再一次性预测最终样本，而是回答“给定当前位置 x_t 和时间 t ，样本下一步应该沿什么方向移动？”这将全局生成问题转化成局部速度预测问题。

这种模型属于 **Continuous Normalizing Flow (CNF, 连续归一化流)**。

2.2 从控制理论角度理解

从动力系统角度，Flow Matching 学习非自治系统：

$$\dot{x} = v_{\theta}(x, t).$$

但它的目标不是单个状态收敛到平衡点，而是让**整体概率分布**按照设计的路径演化：

$$p_0 \longrightarrow p_t \longrightarrow p_1.$$

普通控制常关心 $x(0) \rightarrow x_{\text{target}}$ ；Flow Matching 更关心：

$$p_0(x) \rightarrow p_{\text{target}}(x).$$

无需预先规定每个噪声样本必须对应哪个真实数据点；最终生成分布正确才是核心要求。

三、第三步：为什么需要连续性方程？

3.1 从粒子的运动出发

设大量粒子 $x_t^{(1)}, \dots, x_t^{(N)}$ 均满足：

$$\frac{dx_t^{(i)}}{dt} = v_t(x_t^{(i)}).$$

这些粒子共同构成概率密度 $p_t(x)$ 。粒子流入某一区域，该区域的密度增大；粒子流出该区域，密度减小。

因此，必须建立速度场与概率密度变化之间的关系。

3.2 连续性方程的推导

取任意光滑测试函数 $\phi: \mathbb{R}^d \rightarrow \mathbb{R}$ ，考虑：

$$\mathbb{E}[\phi(x_t)] = \int \phi(x) p_t(x) dx.$$

对时间求导（假设具备交换求导与积分的正则性）：

$$\frac{d}{dt} \mathbb{E}[\phi(x_t)] = \int \phi(x) \frac{\partial p_t(x)}{\partial t} dx.$$

另一方面，链式法则给出：

$$\frac{d}{dt} \phi(x_t) = \nabla \phi(x_t)^\top \frac{dx_t}{dt} = \nabla \phi(x_t)^\top v_t(x_t).$$

于是：

$$\frac{d}{dt} \mathbb{E}[\phi(x_t)] = \int \nabla \phi(x)^\top v_t(x) p_t(x) dx.$$

假设边界项为零，对右侧进行分部积分：

$$= - \int \phi(x) \nabla \cdot [p_t(x) v_t(x)] dx.$$

比较两种表达式：

$$\int \phi(x) \left[\frac{\partial p_t(x)}{\partial t} + \nabla \cdot (p_t(x) v_t(x)) \right] dx = 0.$$

由于对任意适当的 ϕ 都成立，得到：

$$\boxed{\frac{\partial p_t(x)}{\partial t} + \nabla \cdot [p_t(x) v_t(x)] = 0.} \quad (2)$$

这就是 **Continuity Equation (连续性方程)**。

3.3 连续性方程的物理含义

将散度项展开：

$$\frac{\partial p_t}{\partial t} + v_t^\top \nabla p_t + p_t \nabla \cdot v_t = 0.$$

其中 $v_t^\top \nabla p_t$ 表示密度的对流变化， $\nabla \cdot v_t$ 表示局部速度场的发散。

当 $\nabla \cdot v_t > 0$ 时，局部粒子趋向扩张，密度倾向降低；当 $\nabla \cdot v_t < 0$ 时，局部粒子趋向汇聚，密度倾向升高。

沿 ODE 轨迹还可得到：

$$\boxed{\frac{d}{dt} \log p_t(x_t) = -\nabla \cdot v_t(x_t).} \quad (3)$$

这是 CNF 能够计算样本对数密度的理论基础。

动机： ODE 描述单个粒子的运动，连续性方程描述由这些粒子组成的概率分布如何变化。两者建立了生成动力学与目标分布之间的联系。

四、第四步：Flow Matching 的理想目标是什么？

4.1 如果已知正确的速度场

假设我们事先选定了一条概率分布路径 $p_t(x)$ ，满足：

$$p_{t=0} = p_0, \quad p_{t=1} = p_{\text{data}},$$

并且存在速度场 $u_t(x)$ ，使得：

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t u_t) = 0.$$

那么可以直接回归这个速度场：

$$\boxed{\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{\substack{t \sim \mathcal{U}[0,1] \\ x \sim p_t}} [\|v_\theta(x, t) - u_t(x)\|^2].} \quad (4)$$

如果能够精确学到 $v_\theta = u_t$ ，由该速度场诱导的 ODE 就可实现选定的分布演化。

4.2 为什么理想目标不能直接训练?

真实情况下，我们通常只拥有：

$$x_1 \sim p_{\text{data}},$$

并没有观测到“从高斯逐渐演变为真实数据”的过程，因此既不知道中间的 $p_t(x)$ ，也不知道所需的 $u_t(x)$ 。

核心动机：能否人为设计一条容易采样、速度又可直接计算的路径，并用这条路径给神经网络提供训练信号？

这就引出了 Conditional Flow Matching (CFM)。

五、第五步：Conditional Flow Matching 的构造

5.1 构造两个端点

从高斯分布与真实数据分布中分别采样：

$$x_0 \sim p_0 = \mathcal{N}(0, I), \quad x_1 \sim p_{\text{data}}.$$

最简单的方式是令两个端点独立：

$$x_0 \perp x_1.$$

对于一对端点 (x_0, x_1) ，人为指定匀速直线运动：

$$\boxed{x_t = (1-t)x_0 + tx_1.} \quad (5)$$

检查边界条件： $t=0$ 时 $x_t = x_0$ ； $t=1$ 时 $x_t = x_1$ 。

例如：

$$x_0 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad x_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

对 (5) 求导：

$$\boxed{\frac{dx_t}{dt} = x_1 - x_0.} \quad (6)$$

在上述例子中，目标速度是 $[4, 2]^\top$ 。它不依赖 t ，因为我们选择了匀速直线插值。

为什么用直线？对固定端点而言它计算最简单，速度标签能够显式写出，且不需要训练额外模型或模拟中间轨迹。

5.2 直接得到可训练的损失函数

训练时随机采样：

$$x_0 \sim \mathcal{N}(0, I), \quad x_1 \sim p_{\text{data}}, \quad t \sim \mathcal{U}[0, 1].$$

再构造：

$$x_t = (1-t)x_0 + tx_1, \quad u_t = x_1 - x_0.$$

用神经网络预测速度：

$$v_{\theta}(x_t, t) \approx x_1 - x_0.$$

得到：

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t, x_0, x_1} \left[\|v_{\theta}((1-t)x_0 + tx_1, t) - (x_1 - x_0)\|^2 \right]. \quad (7)$$

这就是常用的**直线路径 Conditional Flow Matching**。

一轮训练只需采样一个数据点、一份噪声、一个随机时间，计算插值和目标速度，再做一次网络前向传播与均方误差（MSE）优化。

关键优点：训练时不需要求解 ODE，因此这种训练方式通常称为 *simulation-free training*。

但是还有一个深层疑问：网络输入只有 (x_t, t) ，并不知道真实端点 (x_0, x_1) 。为什么回归 $x_1 - x_0$ 就能得到正确全局速度场？

六、第六步：为什么条件速度回归能够得到全局速度场？

6.1 区分单条路径速度与边缘速度

给定端点 (x_0, x_1) ，单条路径的速度是：

$$u_t(x \mid x_0, x_1) = x_1 - x_0.$$

它依赖于端点选择，可称为**条件速度**。

但推理时并不知道最终数据端点，因此网络必须预测只依赖当前位置和时间的速度场：

$$v_t(x).$$

同一个 (x, t) 可能被许多不同的直线路径经过；它们的速度可能并不相同。网络应该预测什么？答案是：**这些条件速度的条件期望**。

6.2 严格推导边缘速度场

用大写字母表示随机变量。定义：

$$Z = (X_0, X_1), \quad X_t = (1-t)X_0 + tX_1,$$

以及：

$$U_t = X_1 - X_0.$$

由于 $dX_t/dt = U_t$ ，对于光滑测试函数 ϕ ：

$$\frac{d}{dt} \mathbb{E}[\phi(X_t)] = \mathbb{E}[\nabla \phi(X_t)^\top U_t].$$

应用条件期望的塔式性质：

$$= \mathbb{E}[\nabla \phi(X_t)^\top \mathbb{E}[U_t \mid X_t]].$$

定义：

$$\boxed{v_t^*(x) = \mathbb{E}[U_t \mid X_t = x].} \quad (8)$$

于是：

$$\frac{d}{dt} \mathbb{E}[\phi(X_t)] = \int \nabla \phi(x)^\top v_t^*(x) p_t(x) dx.$$

通过分部积分：

$$= - \int \phi(x) \nabla \cdot [p_t(x) v_t^*(x)] dx.$$

另一方面：

$$\frac{d}{dt} \mathbb{E}[\phi(X_t)] = \int \phi(x) \partial_t p_t(x) dx.$$

因此：

$$\boxed{\partial_t p_t(x) + \nabla \cdot [p_t(x) v_t^*(x)] = 0.} \quad (9)$$

也就是说，条件期望构成了一个能够实现目标边缘概率路径的速度场：

$$\boxed{v_t^*(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x].} \quad (10)$$

这是 CFM 的核心数学结论之一。

6.3 用概率密度重新表达

更一般地，令 z 为条件变量， $p_t(x \mid z)$ 为条件概率路径， $u_t(x \mid z)$ 为其条件速度。边缘分布为：

$$p_t(x) = \int p_t(x \mid z) q(z) dz.$$

可定义：

$$\boxed{v_t^*(x) = \frac{\int u_t(x \mid z) p_t(x \mid z) q(z) dz}{p_t(x)}.} \quad (11)$$

其本质仍是：

$$v_t^*(x) = \mathbb{E}[u_t(x \mid Z) \mid X_t = x].$$

对固定端点的确定性插值，条件路径退化为位于插值点的 Dirac 分布，使用上一节的随机变量推导更直接。

注意：连续性方程可能允许不止一个满足要求的速度场。Flow Matching 并不是寻找一个与路径选择无关、唯一存在的“真实速度场”，而是在指定概率路径后，学习能够实现这条路径的速度场。

七、第七步：为什么 MSE 能学到条件期望？

7.1 最小二乘回归的正交分解

固定时间 t ，定义：

$$m_t(X_t) = \mathbb{E}[U_t | X_t].$$

考虑训练损失：

$$\mathcal{L}(\theta) = \mathbb{E}[\|v_\theta(X_t, t) - U_t\|^2].$$

在括号内加上、减去 m_t ：

$$v_\theta - U_t = (v_\theta - m_t) + (m_t - U_t).$$

展开平方：

$$\begin{aligned} \|v_\theta - U_t\|^2 &= \|v_\theta - m_t\|^2 + \|m_t - U_t\|^2 \\ &\quad + 2(v_\theta - m_t)^\top (m_t - U_t). \end{aligned}$$

由于：

$$\mathbb{E}[m_t - U_t | X_t] = 0,$$

并且 $v_\theta(X_t, t) - m_t(X_t)$ 仅依赖于 X_t ，交叉项取期望为零：

$$\mathbb{E}[(v_\theta - m_t)^\top (m_t - U_t)] = 0.$$

于是：

$$\boxed{\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}[\|v_\theta(X_t, t) - m_t(X_t)\|^2] + \mathbb{E}[\|U_t - m_t(X_t)\|^2].} \quad (12)$$

第二项与参数 θ 无关。因此，在相同采样分布和平方误差定义下：

$$\boxed{\arg \min_{\theta} \mathcal{L}_{\text{CFM}}(\theta) = \arg \min_{\theta} \mathcal{L}_{\text{FM}}(\theta).} \quad (13)$$

在可交换求导与积分的条件下，进一步有：

$$\boxed{\nabla_{\theta} \mathcal{L}_{\text{CFM}} = \nabla_{\theta} \mathcal{L}_{\text{FM}}.} \quad (14)$$

7.2 为什么这是关键突破？

我们想学的是：

$$v_t^*(x) = \mathbb{E}[U_t | X_t = x],$$

但这个条件期望难以显式计算。

CFM 的做法是：不先计算条件期望，而是直接回归随机采样的 U_t 。MSE 的理论性质会使最优预测自动逼近条件期望。

$$\underbrace{\text{难计算的边缘速度场}}_{v_t^*(x)} \Leftarrow \underbrace{\text{可采样的条件速度}}_{X_1 - X_0}.$$

这是 Conditional Flow Matching 能被高效训练的根本原因。

7.3 为什么训练路径是直线，生成轨迹却不一定是直线？

训练时每对端点的路径确实是直线：

$$X_t = (1 - t)X_0 + tX_1.$$

但最终学习的是条件平均速度：

$$v_t^*(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x].$$

例如，两条路径在同一个位置相交，条件速度分别为：

$$u_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad u_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

如果两条路径在该位置的条件权重相同，则：

$$v_t^*(x) = \frac{1}{2}u_1 + \frac{1}{2}u_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

这个速度不等于任意一条单独路径的速度。因此求解 $\dot{x}_t = v_t^*(x_t)$ 后得到的生成轨迹可以是弯曲的。

重要结论：训练样本使用直线插值，不代表最终 ODE 生成轨迹必然是直线。

八、第八步：为什么还要引入 Gaussian Conditional Path?

前面的直线插值只是 Flow Matching 的一个特例。原始框架允许使用更一般的条件概率路径。

8.1 一般高斯条件路径

给定一个数据样本 x_1 ，定义：

$$p_t(x \mid x_1) = \mathcal{N}(x; \mu_t(x_1), \sigma_t^2 I). \quad (15)$$

其中 $\mu_t(x_1)$ 控制分布中心如何移动， σ_t 控制分布如何扩张或收缩。

该分布可以通过采样表示：

$$X_t = \mu_t(x_1) + \sigma_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \quad (16)$$

对时间求导：

$$\frac{dX_t}{dt} = \dot{\mu}_t(x_1) + \dot{\sigma}_t \epsilon.$$

根据 (16):

$$\epsilon = \frac{X_t - \mu_t(x_1)}{\sigma_t}.$$

代入可得条件速度场:

$$u_t(x \mid x_1) = \dot{\mu}_t(x_1) + \frac{\dot{\sigma}_t}{\sigma_t}[x - \mu_t(x_1)]. \quad (17)$$

推导要求相应时刻 $\sigma_t > 0$; 对于终点方差趋零的情形, 应按极限理解端点。

动机: 只要人为指定 μ_t 和 σ_t , 就能直接计算条件速度, 使用同样的 MSE 框架训练模型。直线插值不是唯一可选路径。

8.2 最简单的线性高斯路径

选择:

$$\mu_t(x_1) = tx_1, \quad \sigma_t = 1 - (1 - \sigma_{\min})t,$$

其中 $0 \leq \sigma_{\min} \ll 1$ 。于是:

$$X_t = tx_1 + [1 - (1 - \sigma_{\min})t]\epsilon. \quad (18)$$

当 $t = 0$ 时, $X_0 = \epsilon \sim \mathcal{N}(0, I)$ 。

当 $t = 1$ 时, $X_1 = x_1 + \sigma_{\min}\epsilon$, 即:

$$X_1 \mid x_1 \sim \mathcal{N}(x_1, \sigma_{\min}^2 I).$$

当 $\sigma_{\min} = 0$ 时, 终点恰好为真实样本; 若 $\sigma_{\min} > 0$, 终点边缘分布是数据分布与一个小高斯的卷积, 并非严格等于原始数据分布。

8.3 推导速度场

对 (18) 求导:

$$\frac{dX_t}{dt} = x_1 - (1 - \sigma_{\min})\epsilon.$$

因此目标速度为:

$$u_t = x_1 - (1 - \sigma_{\min})\epsilon. \quad (19)$$

令 $\sigma_{\min} = 0$, 得到:

$$X_t = (1 - t)\epsilon + tx_1,$$

以及:

$$u_t = x_1 - \epsilon. \quad (20)$$

这正是之前的直线插值 Flow Matching。

总结：常见的 $x_1 - \epsilon$ 速度回归目标可以看作一般 Gaussian Conditional Flow Matching 的一个特例。不同的概率路径会影响监督目标、学习难度与数值积分表现。

九、第九步：Flow Matching 实际如何训练和生成？

9.1 训练过程

给定真实数据样本：

$$x_1 \sim p_{\text{data}}.$$

依次执行以下操作：

1. 采样高斯噪声： $x_0 \sim \mathcal{N}(0, I)$ 。
2. 随机采样时间： $t \sim \mathcal{U}(0, 1)$ 。
3. 构造插值样本： $x_t = (1 - t)x_0 + tx_1$ 。
4. 构造目标速度： $u_t = x_1 - x_0$ 。
5. 网络预测速度： $\hat{u}_t = v_\theta(x_t, t)$ 。
6. 以 $\mathcal{L} = \|\hat{u}_t - u_t\|^2$ 优化参数。

整个训练过程不需要对生成 ODE 进行逐步积分。

下面是简化的 PyTorch 实现：

```
import torch

def flow_matching_loss(model, x1, cond=None):
    # x1 shape: [B, D] or [B, H, D]
    B = x1.shape[0]

    # 1. Gaussian noise
    x0 = torch.randn_like(x1)

    # 2. Random flow timestep
    t = torch.rand(B, device=x1.device)
    t_view = t.view(B, *[1] * (x1.ndim - 1))

    # 3. Linear interpolation
    xt = (1 - t_view) * x0 + t_view * x1

    # 4. Target velocity
    target_v = x1 - x0

    # 5. Predict velocity
    pred_v = model(xt, t, cond)

    # 6. Mean squared error
    loss = ((pred_v - target_v) ** 2).mean()
    return loss
```

9.2 推理过程

生成时不再需要真实样本 x_1 。首先采样：

$$x_0 \sim \mathcal{N}(0, I),$$

然后求解：

$$\frac{dx_t}{dt} = v_\theta(x_t, t).$$

最简单的积分方法是 Euler。将 $[0, 1]$ 均分为 K 步，令 $\Delta t = 1/K$ ，则：

$$x_{k+1} = x_k + \Delta t v_\theta(x_k, k\Delta t). \quad (21)$$

最终 $x_K \approx x_1$ 。PyTorch 伪代码：

```
@torch.no_grad()
def sample_flow(model, shape, cond=None, K=10, device="cuda"):
    x = torch.randn(shape, device=device)
    dt = 1.0 / K

    for k in range(K):
        t = torch.full((shape[0],), k * dt, device=device)
        v = model(x, t, cond)
        x = x + dt * v

    return x
```

除了 Euler，还可以使用 Heun、Midpoint、Runge-Kutta 和自适应 ODE 求解器。高阶求解器可能降低数值误差，但单个步长往往需要更多次网络评估，因此需要在精度和 NFE（函数评估次数）之间权衡。

十、第十步：为什么 Flow Matching 能生成多模态分布？

假设相同观察 o 对应左右两种抓取方式：

$$p(A | o) = \frac{1}{2}p_{\text{left}}(A | o) + \frac{1}{2}p_{\text{right}}(A | o).$$

普通 MSE 策略 $f_\theta(o)$ 只看到观察 o ，最优输出为条件均值 $\mathbb{E}[A | o]$ ，可能落在两种动作之间。

而 Flow Matching 的模型输入为：

$$v_\theta(x_t, t, o),$$

其中起始样本来自：

$$x_0 \sim \mathcal{N}(0, I).$$

不同初始噪声可以被传输到不同动作模态。

即使给定 x_0 和 o 后 ODE 的输出是确定的， X_0 自身仍是随机变量，因此最终：

$$X_1 = \Phi_1(X_0; o)$$

也仍是随机变量，其条件分布为：

$$p_\theta(A | o) = (\Phi_1(\cdot; o))_\# \mathcal{N}(0, I). \quad (22)$$

关键区别：Flow Matching 虽然也使用 MSE，但它回归的是传输速度场而非最终动作的单一均值，所以可以对多模态动作分布建模。

十一、第十一步： $\pi 0.5$ 如何使用 Flow Matching?

11.1 生成对象是 Action Chunk

本章只从生成模型数学角度研究动作块；示教数据集、Action Chunk 执行长度、时序对齐与重规划规范统一放在数据采集与模仿学习。

机器人动作模型希望学习：

$$p(A \mid o, \ell),$$

其中：

$$A = [a_0, a_1, \dots, a_{H-1}]$$

是未来一段动作序列； o 是视觉和机器人状态， ℓ 是语言或高层任务条件。

若单步动作维度为 d_a ，则：

$$A \in \mathbb{R}^{H \times d_a}.$$

例如 $H = 50$ 、 $d_a = 18$ ，则整个 Action Chunk 有 900 个连续变量。Flow Matching 把整个动作块作为高维变量进行生成，而不是将每一维动作逐个自回归采样。

11.2 条件 Flow Matching 如何生成动作块?

先沿用本讲义的时间约定： $t = 0$ 是噪声， $t = 1$ 是真实动作。令：

$$\epsilon \sim \mathcal{N}(0, I),$$

并构造：

$$A_t = (1 - t)\epsilon + tA. \quad (23)$$

目标速度：

$$u_t = A - \epsilon. \quad (24)$$

于是训练 Action Expert：

$$\mathcal{L}_{\text{FM}} = \mathbb{E} [\|v_\theta(A_t, t, o, \ell) - (A - \epsilon)\|^2]. \quad (25)$$

相比无条件 Flow Matching，只需将观察和语言条件输入速度场。理想条件速度场为：

$$v_t(x \mid o, \ell) = \mathbb{E}[U_t \mid X_t = x, o, \ell]. \quad (26)$$

这表示网络学习：在给定观察和任务要求时，如何将高斯噪声变成合理的动作块。

部署衔接：Flow Matching 的生成目标与机器人推理时的动作块切换是不同问题。异步执行、前缀 Inpainting、Training-time RTC 以及延迟评估集中见数据采集与模仿学习的 RTC 小节。

11.3 重要实现细节：OpenPI 使用相反的时间方向

为了复现 $\pi 0.5$ ，必须区分论文讲解中的时间约定和实际代码实现。

在 OpenPI 官方 `pi0.py` 的相关 Flow Matching 实现中，时间约定是：

$$t = 1 : \text{噪声}, \quad t = 0 : \text{真实动作}.$$

具体插值为：

$$A_t = t\epsilon + (1 - t)A. \quad (27)$$

因此：

$$\frac{dA_t}{dt} = \epsilon - A. \quad (28)$$

速度目标为：

$$\mathcal{L} = \|v_\theta(A_t, t, o) - (\epsilon - A)\|^2. \quad (29)$$

即预测的是 **noise - action**，而不是 **action - noise**。这两种定义都可以使用，但生成时间方向必须与速度目标保持一致。

生成从 $A_1 \sim \mathcal{N}(0, I)$ 开始，按 $t : 1 \rightarrow 0$ 积分。使用 K 步 Euler 时， $\Delta t = -1/K$ ：

$$A_{t+\Delta t} = A_t + \Delta t v_\theta(A_t, t, o),$$

即：

$$A_{t-1/K} = A_t - \frac{1}{K} v_\theta(A_t, t, o). \quad (30)$$

积分终点 A_0 就是生成的动作块。OpenPI 的相关默认配置常采用 10 个积分步，并可缓存视觉语言主干前缀的 KV，以降低每步重复计算的成本；具体参数应以使用的代码版本和配置为准。

Flow Matching Velocity 不等于机器人关节速度

必须区分两种时间：

- **生成时间 t** ：ODE 从噪声生成 Action Chunk 的虚拟时间。
- **物理时间**：机器人实际执行 $a_0, a_1, \dots, a_{H-1}$ 的时间。

如果 $A_t \in \mathbb{R}^{50 \times 18}$ ，则：

$$v_\theta(A_t, t, o) \in \mathbb{R}^{50 \times 18}.$$

该速度描述的是整个 noisy action chunk 沿生成时间的变化，并非机械臂的物理关节速度。

十二、第十二步：Flow Matching 如何用于强化学习微调？

这一节把前面的推导联系到 $\pi 0.5$ 、DSRL 和 **Residual Velocity Adaptation** 的研究问题。

12.1 预训练 Flow Matching 学到了什么?

模仿学习得到一个速度场:

$$v_{\text{pre}}(x, t, o),$$

它对应的生成分布大致与演示动作分布 $p_{\text{demo}}(A | o)$ 一致。

通过:

$$\frac{dA_t}{dt} = v_{\text{pre}}(A_t, t, o)$$

能够生成与训练演示数据相似的动作, 但这些动作未必在当前任务中具有最高回报。

例如, 相同物体有多种有效抓取方式, 但成功率、稳定性和完成时间可能不同。强化学习后训练的目标是让生成分布更偏向高回报动作。

12.2 为什么可以学习 Residual Velocity?

一种自然的设计是冻结预训练速度场, 只训练一个较小的残差:

$$v_{\text{new}}(x, t, o) = v_{\text{pre}}(x, t, o) + \Delta v_{\phi}(x, t, o). \quad (31)$$

生成 ODE 因此变成:

$$\frac{dA_t}{dt} = v_{\text{pre}}(A_t, t, o) + \Delta v_{\phi}(A_t, t, o). \quad (32)$$

动机类似控制理论中的 Residual Control: 保留预训练策略的基础能力, 同时通过增量修正适应新的奖励或任务目标。

但要注意, 这是一种后训练算法设计, 并不是原始 Flow Matching 理论必然要求的结构。

12.3 为什么不能把 Reward 直接当作 Velocity Label?

预训练中, 条件速度目标是:

$$u_t = A - \epsilon.$$

这是由明确的路径方程求导得到的监督信号。

强化学习得到的通常是:

$$R(A, o),$$

它是对最终动作或执行轨迹的回报, 不直接告诉我们每个中间生成状态上的残差速度:

$$\Delta v_{\phi}(A_t, t, o).$$

所以不能简单把 Advantage 解释成生成空间中的速度标签。需要建立 **Reward** $\rightarrow$ **目标动作分布** $\rightarrow$ **速度场更新** 的合理联系。

12.4 Advantage-Weighted Flow Matching 的一个自然设计

假设在线探索得到数据：

$$\mathcal{D} = \{(o_i, A_i, R_i)\}_{i=1}^N.$$

为高回报动作赋予更大的正权重：

$$w_i = \exp\left(\frac{\widehat{\text{Adv}}_i}{\beta}\right), \quad \beta > 0.$$

定义加权残差速度回归目标：

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathcal{D}, \epsilon, t} \left[w(A, o) \left\| v_{\text{pre}}(A_t, t, o) + \Delta v_{\phi}(A_t, t, o) - (A - \epsilon) \right\|^2 \right], \quad (33)$$

其中：

$$A_t = (1 - t)\epsilon + tA.$$

还可以加入残差正则化：

$$\lambda \mathbb{E} [\|\Delta v_{\phi}(A_t, t, o)\|^2]$$

来限制与预训练速度场的偏离。

为什么这个方案有数学动机？

回顾：MSE 回归的最优解是相应联合分布下条件速度的条件期望。

对固定观察 o ，定义重新加权后的动作分布：

$$p_w(A | o) = \frac{w(A, o)p_{\mathcal{D}}(A | o)}{Z(o)}, \quad (34)$$

其中：

$$Z(o) = \mathbb{E}_{A \sim p_{\mathcal{D}}(\cdot | o)} [w(A, o)].$$

在训练数据有足够覆盖、权重非负可积、模型容量和优化充分的前提下，加权 CFM 可以解释为对加权目标动作分布进行速度场学习。

因此研究逻辑可以写成：

$$\text{Advantage} \longrightarrow \text{加权动作分布} \longrightarrow \text{Flow Matching} \longrightarrow \text{新速度场}.$$

重要限制：加权 Flow Matching 不自动等价于在线 Policy Gradient，也不保证策略性能单调改善。实际表现仍受到数据覆盖、Advantage 估计、分布偏移、数值积分以及约束设计的影响。

十三、总结：Flow Matching 的完整数学逻辑

13.1 八个关键环节

1. 确定生成目标。从高斯分布生成真实数据：

$$p_0 = \mathcal{N}(0, I) \longrightarrow p_1 = p_{\text{data}}.$$

2. 用 ODE 定义连续变换。

$$\dot{x}_t = v_\theta(x_t, t).$$

其诱导的概率路径应满足：

$$\partial_t p_t + \nabla \cdot (p_t v_\theta) = 0.$$

3. 理想目标是回归全局速度场。

$$\mathcal{L}_{\text{FM}} = \mathbb{E}[\|v_\theta(x, t) - v_t^*(x)\|^2],$$

但 v_t^* 通常未知。

4. 构造可计算的条件路径。

$$x_t = (1 - t)x_0 + tx_1, \quad u_t = x_1 - x_0.$$

5. 证明边缘速度是条件期望。

$$v_t^*(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x].$$

6. 利用 MSE 的 L^2 投影性质。

$$\mathbb{E}\|v_\theta - U_t\|^2 = \mathbb{E}\|v_\theta - v_t^*\|^2 + C.$$

7. 得到简单的训练损失。

$$\min_{\theta} \mathbb{E}_{\substack{t \sim \mathcal{U}[0,1] \\ x_0 \sim \mathcal{N}(0, I) \\ x_1 \sim p_{\text{data}}}} \left[\|v_\theta((1 - t)x_0 + tx_1, t) - (x_1 - x_0)\|^2 \right].$$

8. 生成时通过 ODE 积分。

$$\begin{cases} x_0 \sim \mathcal{N}(0, I), \\ \dot{x}_t = v_\theta(x_t, t), \\ x_1 \sim p_\theta \approx p_{\text{data}}. \end{cases}$$

最后一行表示生成样本的总体分布近似真实数据分布，并不意味着每个生成样本必须等于训练集中某个已有样本。

13.2 三个最重要的理解

第一，Flow Matching 是学习概率分布的动力系统。

$$\dot{x}_t = v_t(x_t) \iff \partial_t p_t = -\nabla \cdot (p_t v_t).$$

它将生成模型与连续时间动力系统联系起来。

第二，CFM 的核心是条件期望与 L^2 投影。

$$v_t^*(x) = \mathbb{E}[U_t \mid X_t = x].$$

这解释了为什么简单的随机路径监督就足以构造复杂分布的速度场学习问题。

第三，训练时的直线插值不等于推理时的直线轨迹。神经网络学习的是全局边缘速度场，受条件路径与样本分布共同影响。

对于 Flow Matching Policy 的在线后训练，可以进一步研究：

$$\text{改变动作分布} \iff \text{改变概率传输的速度场}.$$

如何利用 Reward 或 Advantage 稳定改变速度场、同时保留预训练生成能力，是一个独立于标准 Flow Matching、值得系统研究的优化问题。

文档说明：本版仅保留 Flow Matching 基础推导、条件流匹配、实现流程、 $\pi 0.5$ 和强化学习残差速度微调内容；已删除指定的两部分，并将后续章节与公式编号连续调整。