Training Flow and Diffusion Models

Reminder - Marginal vector field:

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x|z) \frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z$$

Conditional vector field

← High computational cost!!!

Marginalization trick: The marginal vector field defines an ODE that "follows" the marginal probability path:

$$X_0 \sim p_{\text{init}}, \quad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t^{\text{target}}(X_t)\mathrm{d}t \quad \Rightarrow X_t \sim p_t$$

This means that the final point fulfils: $X_1 \sim p_{\text{data}}$

3.1 Flow Matching

u_t^θ

Goal: $u_t^\theta \simeq u_t^{\text{target}}$ (marginal VF) ⇒ Learn marginal VF.

Flow matching loss: $\mathcal{L}_{\text{fm}}(\theta) = \mathbb{E}\left[\| u_t^\theta(x) - u_t^{\text{target}}(x) \|_2^2\right]$

$t \sim \text{Unif}$: t is uniform in $[0, 1]$.

$z \sim p_{\text{data}}$: draw samples from data

$x \sim p_t(\cdot|z)$: draw samples from $p(\cdot|z)$ for fixed z.

We want: minimizer of $\mathcal{L}_{\text{fm}}$.

But this loss is not tractable! ⇒ Because $u_t^{\text{target}}(x)$ is an integral. Hard to compute.

Conditional Flow matching loss:

$$\mathcal{L}_{\text{cfm}}(\theta) = \mathbb{E}\left[\| u_t^\theta(x) - u_t^{\text{target}}(x|z) \|_2^2\right]$$

$t \sim \text{Unif}$: t is uniform in $[0, 1]$.

① Tractable.

② But minimize CFM loss

will help minimize FM loss? $z \sim p_{data}$: draw samples from data

$x \sim p_t(\cdot|z)$: draw samples from $p(\cdot|z)$ for fixed z.

Main Thm: Minimize $L_{cfm}(\theta)$ $\Leftrightarrow$ Minimize $L_{fm}(\theta)$.

Conditional Prob. Path, Vector Field, and Score

	Notation	Key property	Gaussian example
Conditional Probability Path	$p_t(\cdot\|z)$	Interpolates p_{init} and a data point z	$\mathcal{N}(\alpha_t z, \beta_t^2 I_d)$
Conditional Vector Field	$u_t^{\text{target}}(x\|z)$	ODE follows conditional path	$\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$
Conditional Score Function	$\nabla \log p_t(x\|z)$	Gradient of log-likelihood	$-\frac{x - \alpha_t z}{\beta_t^2}$

Can be derived directly if we set α_t, β_t in advance.

Thm 4.1 $L_{fm}(\theta) = L_{cfm}(\theta) + C$, where $C > 0$ is independent of θ.

① for minimizer θ^* of L_{cfm}: $u_t^{\theta^*} = u_t^{\text{target}}$

② $\nabla_\theta L_{cfm}(\theta) = \nabla_\theta L_{fm}(\theta)$

$\Rightarrow$ SGD process will be exactly the same.

Algorithm 3 Flow Matching Training Procedure (General)

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do** SGD.
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$. Another idea: not uniform time sampling.
4: Sample $x \sim p_t(\cdot|z)$
5: Compute loss

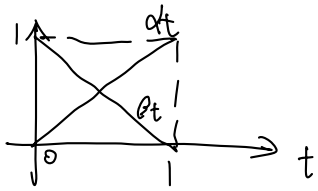
$$\mathcal{L}(\theta) = \|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2$$

6: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
7: **end for**

nn. Architecture: Takes input (x, t) $\rightarrow$ output $u_t^\theta(x)$. (para by θ)

$\mathcal{L}_{cfm}$ for Gaussian cond path

$$p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$$



$$u_t^{target}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$$

$$\varepsilon \sim \mathcal{N}(0, I_d) \Rightarrow \alpha_t z + \beta_t \varepsilon \overset{\text{dist}}{=} x \sim p_t(\cdot|z).$$

Then $$\mathcal{L}_{cmf}(\theta) = \mathbb{E}_{\substack{t\sim \text{unif} \\ z\sim p_{data} \\ x\sim \mathcal{N}(\alpha_t z, \beta_t^2 I_d)}} \left[\left\| u_t^\theta(x) - \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z - \frac{\dot{\beta}_t}{\beta_t} x \right\|_2^2 \right]$$

Replace $x = \alpha_t z + \beta_t \varepsilon$.

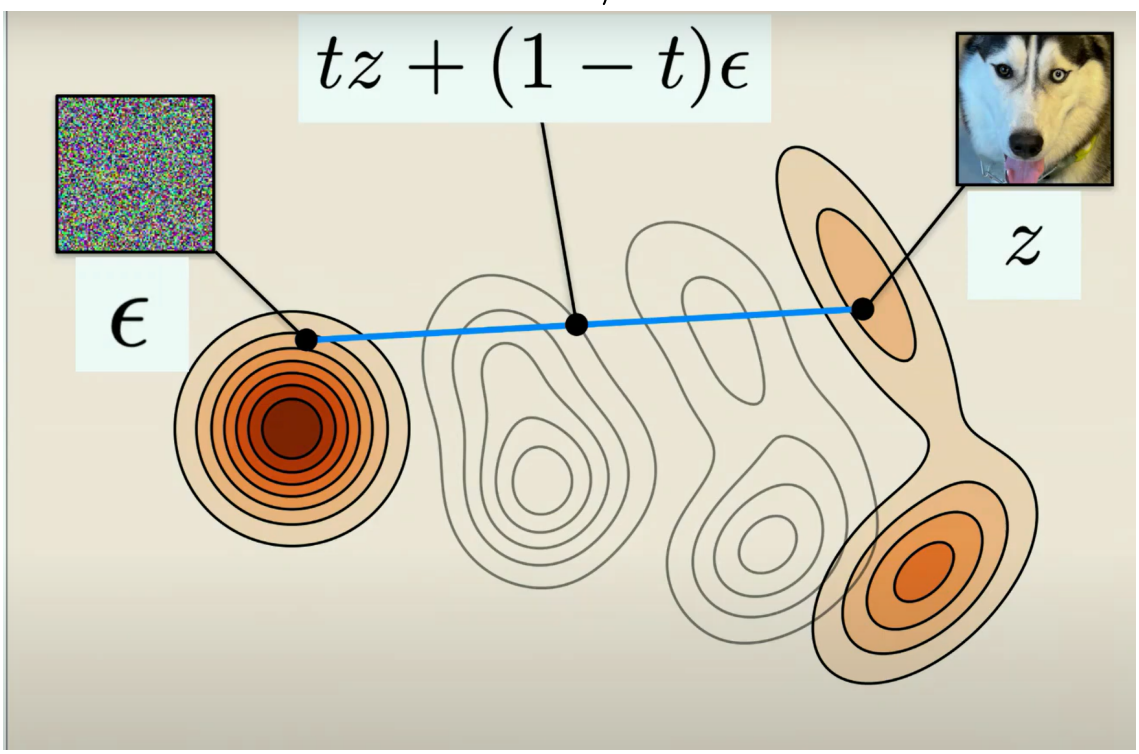
$$= \mathbb{E}_{\substack{t\sim \text{unif} \\ z\sim \text{data} \\ \varepsilon\sim \mathcal{N}(0, I_d)}} \left[\left\| u_t^\theta(\alpha_t z + \beta_t \varepsilon) - (\dot{\alpha}_t z + \dot{\beta}_t \varepsilon) \right\|_2^2 \right]$$

A easy choice for α_t, β_t: $\begin{cases} \alpha_t = t & \dot{\alpha}_t = 1 \\ \beta_t = 1-t. & \dot{\beta}_t = -1 \end{cases}$

$$= \mathbb{E}_{\substack{t\sim \text{unif} \\ z\sim \text{data} \\ \varepsilon\sim \mathcal{N}(0, I_d)}} \left[\left\| u_t^\theta(t z + (1-t)\varepsilon) - (z - \varepsilon) \right\|_2^2 \right]$$

Cond OT path.
(Optimal Transport Path)

A straight line between noise & data



Algorithm 4 Flow Matching Training for CondOT path

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$ ✓. Easy to sp.
5: Set $x = tz + (1-t)\epsilon$
6: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - (z - \epsilon)\|^2$$

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

Stable diffusion: still use this training algo.

---

proof of Thm 4.1.

$$\mathcal{L}_{fm}(\theta) = \mathbb{E}_{t,z,x}\left[\|u_t^\theta(x) - u_t^{target}(x)\|_2^2\right] = \mathbb{E}\left[\|u_t^\theta(x)\|_2^2 + \|u_t^{target}(x)\|_2^2 - 2u_t^\theta(x)^\top u_t^{target}(x)\right]$$

$$\mathcal{L}_{cfm}(\theta) = \mathbb{E}\left[\|u_t^\theta(x)\|_2^2 + \|u_t^{target}(x|z)\|_2^2 - 2u_t^\theta(x)^\top u_t^{target}(x|z)\right].$$

Denote $C = \mathbb{E}\left[\|u_t^{target}(x)\|_2^2 - \|u_t^{target}(x|z)\|_2^2\right]$. ... A constant.

$$\mathcal{L}_{fm}(\theta) - \mathcal{L}_{cfm}(\theta) = 2\mathbb{E}\left[u_t^\theta(x)^\top u_t^{target}(x|z) - u_t^\theta(x)^\top u_t^{target}(x)\right] + C$$

$$u_t^{target}(x) = \int u_t^{target}(x|z)\,\frac{p_t(x|z)\,p_{data}(z)}{p_t(x)}\,dz \qquad t, z, x.$$

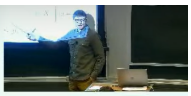
$$\mathbb{E}\left[u_t^\theta(x)^\top u_t^{target}(x)\right] = \mathbb{E}\left[u_t^\theta(x)^\top \int_z u_t^{target}(x|z)\,\frac{p_t(x|z)\,p_{data}(z)}{p_t(x)}\,dz\right]$$

$$= \int_0^1 \int_x \int_z p_t(x)\; u_t^\theta(x)^\top u_t^{target}(x|z)\,\frac{p_t(x|z)\,p_{data}(z)}{p_t(x)}\,dz\,dx\,dt$$

$$= \mathbb{E}\left[u_t^\theta(x)^\top u_t^{target}(x|z)\right].$$

4.2 Score Matching.

Reminder - Marginal score function:



$$\nabla \log p_t(x) = \int \nabla \log p_t(x|z) \frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z$$

Conditional score function

SDE extension trick: The marginal score function allows to extend the ODE to a SDE with arbitrary diffusion coefficient:

score function: ODE ⇒ SDE.

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2} \nabla \log p_t(X_t) \right] \mathrm{d}t + \sigma_t \mathrm{d}W_t$$

$$\Rightarrow X_t \sim p_t$$

Already solved by FM

Train these 2.

推导:

$$\nabla \log p_t(x) = \frac{\nabla_x p_t(x)}{p_t(x)} = \frac{\nabla_x \int p_t(x|z) p_{\text{data}}(z) \, dz}{p_t(x)}$$

$$= \int \nabla_x \log p_t(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} dz$$

Also hard to directly compute $\nabla \log p_t(x)$.

Score Network: s_t^θ (θ parameters) Goal: $s_t^\theta \simeq \nabla \log p_t$.

Score Matching loss:
$$\mathcal{L}_{\text{sm}}(\theta) = \mathbb{E}_{t,z,x}\left[\| s_t^\theta(x) - \nabla \log p_t(x) \|_2^2 \right]$$

Denoising Matching loss:
$$\mathcal{L}_{\text{dsm}}(\theta) = \mathbb{E}_{t,z,x}\left[\| s_t^\theta(x) - \nabla \log p_t(x|z) \|_2^2 \right]$$

Similar to FM.

SM loss: ✗ tractable

DSM loss: ✓ tractable.

Thm 4.2. $\mathcal{L}_{\text{sm}}(\theta) = \mathcal{L}_{\text{dsm}}(\theta) + C$. for C const of θ.

① for θ^* of $\mathcal{L}_{\text{dsm}}$. $s_t^{\theta^*} = \nabla \log p_t$.

(2) $\nabla_\theta L_{sm}(\theta) = \nabla_\theta L_{dsm}(\theta)$.

The pf is exactly the same as PM part.

Algorithm 6 Score Matching Training Procedure (General)

Require: A dataset of samples $z \sim p_{\text{data}}$, score network s_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample $x \sim p_t(\cdot|z)$
5: Compute loss

$$\mathcal{L}(\theta) = \|s_t^\theta(x) - \nabla \log p_t(x|z)\|^2$$

6: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
7: **end for**

Eg. for Gaussian prob path,

$$\nabla \log p_t(x|z) = -\frac{x - \alpha_t z}{\beta_t^2}$$

$$L_{dsm}(\theta) = \mathbb{E}_{\substack{t\sim \text{Unif} \\ z\sim \text{data} \\ x\sim \mathcal{N}(\alpha_t z, \beta_t^2 I_d)}} \left[\left\| s_t^\theta(x) + \frac{x - \alpha_t z}{\beta_t^2} \right\|_2^2 \right]$$

$x = \alpha_t z + \beta_t \varepsilon$.
$\varepsilon \sim \mathcal{N}(0, I_d)$

$$= \mathbb{E}_{\substack{t\sim \text{Unif} \\ z\sim \text{data} \\ \varepsilon\sim \mathcal{N}(0, I_d)}} \left[\left\| s_t^\theta(\alpha_t z + \beta_t \varepsilon) + \frac{\varepsilon}{\beta_t} \right\|_2^2 \right]$$

Algorithm 5 Score Matching Training Procedure for Gaussian probability

Require: A dataset of samples $z \sim p_{\text{data}}$, score network s_t^θ or noise predictor ϵ_t^θ
Require: Schedulers α_t, β_t with $\alpha_0 = \beta_1 = 0, \alpha_1 = \beta_0 = 1$

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x_t = \alpha_t z + \beta_t \epsilon$
6: Compute loss

$$\mathcal{L}(\theta) = \|s_t^\theta(x_t) + \frac{\epsilon}{\beta_t}\|^2$$

Numerically unstable for low beta

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

For Gaussian prob path with $\beta_t = 1 - t$,

$\beta_t \to 0$ as $t \to 1$. $\frac{\epsilon}{\beta_t} \to \infty$?

To deal with this problem, DDPM was proposed. ① Drop $\frac{1}{\beta_t}$ ② $\xi_t(x) = -\beta_t s_t^\theta(x)$.

Let $\xi_t^\theta(x) = -\beta_t s_t^\theta(x)$, a noisy predictor. Σ_t^θ is another nn.

$$\Sigma_t^\theta : \underset{x}{\mathbb{R}^d} \times \underset{t}{[0,1]} \mapsto \mathbb{R}^d$$

$$\mathcal{L}_{DDPM}(\theta) = \mathbb{E}_{\substack{t \sim \text{Unif} \\ z \sim p_{data} \\ \varepsilon \sim \mathcal{N}(0, I_d)}} \left[\| \xi_t^\theta(\alpha_t z + \beta_t \varepsilon) - \varepsilon \|_2^2 \right].$$

Algorithm 4 Score Matching Training Procedure for Gaussian probability path

Require: A dataset of samples $z \sim p_{\text{data}}$, score network s_t^θ or noise predictor ϵ_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x_t = \alpha_t z + \beta_t \epsilon$ (General case: $x_t \sim p_t(\cdot|z)$)
6: Compute loss

$$\mathcal{L}(\theta) = \|s_t^\theta(x_t) + \frac{\epsilon}{\beta_t}\|^2 \qquad \text{(General case: } = \|s_t^\theta(x_t) - \nabla \log p_t(x_t|z)\|^2\text{)}$$

Alternatively: $\mathcal{L}(\theta) = \|\epsilon_t^\theta(x_t) - \epsilon\|^2$ when $t \to 1$.

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

Stochastic sampling with diffusion models

$$X_0 \sim p_{\text{init}}, \quad dX_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2} \nabla \log p_t(X_t) \right] dt + \sigma_t dW_t$$

Plugin neural networks

$$\Rightarrow X_t \sim p_t$$

$$X_0 \sim p_{\text{init}}, \quad dX_t = \left[u_t^\theta(X_t) + \frac{\sigma_t^2}{2} s_t^\theta(X_t) \right] dt + \sigma_t dW_t$$

After training $\Rightarrow X_t \sim p_t$

newal networks into the SD that we had

Denoising Diffusion Models (DDMs)

Denoising diffusion models

= Diffusion models with a Gaussian probability path

(our standard example) $\mathcal{N}(\alpha_t z, \beta_t^2 I_d)$

Terminology (by many people)

Denoising diffusion model = diffusion model

i.e. many people drop the word "denoising" and just say "diffusion model".

Special property for DDM:

Obs: $\begin{cases} u_t^{target}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right)z + \frac{\dot{\beta}_t}{\beta_t}x. \\ \nabla \log p_t(x|z) = -\frac{x-\alpha_t z}{\beta_t^2} = -\frac{\alpha_t}{\beta_t^2}z - \frac{1}{\beta_t^2}x. \end{cases}$

> Both comb of z & x.
> ⇒ Convert them to each other.

Easy to show:

$$u_t^{target}(x|z) = \left(\beta_t^2\frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t\beta_t\right)\nabla \log p_t(x|z) + \frac{\dot{\alpha}_t}{\alpha_t}x.$$

$$u_t^{target}(x) = \left(\beta_t^2\frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t\beta_t\right)\nabla \log p_t(x) + \frac{\dot{\alpha}_t}{\alpha_t}x.$$

Conversion formula for DDMs

One can also convert the score network into vector field network post training:

$$u_t^\theta = \left(\beta_t^2\frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t\beta_t\right)s_t^\theta(x) + \frac{\dot{\alpha}_t}{\alpha_t}x$$

In denoising diffusion models, we don't have to train the vector field and score network separately

→ they can be converted into one another post-training

The first generation of diffusion models only did score matching!

We only need to train one of u_t^θ & $s_t^\theta(x)$.

Next steps: nn. Architectures.

conditioned on a prompt ?

Brief literature overview (see lecture notes for details)

- There is a whole zoo of different flow and diffusion model (Don't be confused!)
- First diffusion models: **Discrete time** (no ODE/SDEs)
- First SDE-based model: Forward-backward process using **time-reversal** of SDEs → construction of a probability path
- **Inverted time convention**: Data distribution at t=0

Here: Flow matching and stochastic interpolants:

- Arguably most simple flow and diffusion algorithms
- Allows you to restrict yourself to flows
- Allows you go from arbitrary p_{init} to arbitrary p_{data}

Note: Our method allows to convert arbitrary distributions into arbitrary distributions!

We not always start from Gaussian. The distribution can be arbitrary.

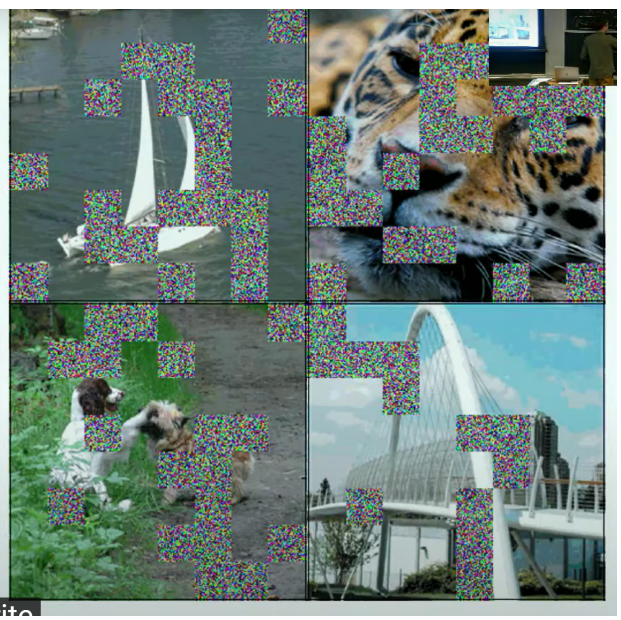
Bridging arbitrary distributions - Example

Videos without audio → videos with audio

Low resolution images → high resolution images

Unperturbed cells → perturbed cells

etc.



noise that's
for what what we write

These are not pure Gaussian noise at start.