Training Flow and Diffusion Models

Lecturer: Qiang Sun — Lecture 05

---

1 Training the Generative Model

In the last few classes, we constructed a generative model with a time-dependent vector field u_t^θ (parameterized by a neural network), and we derived a training target u_t^{target}. In this section we describe how to *train* u_t^θ so that $u_t^\theta \approx u_t^{\text{target}}$.

We proceed in three steps:

1. Restrict to ODEs and recover *flow matching*.
2. Extend to SDEs via *(conditional) score matching*.
3. Specialize to Gaussian probability paths and recover *denoising diffusion models*.

1.1 Flow Matching

Consider a flow model (ODE) of the form

$$X_0 \sim p_{\text{init}}, \qquad \mathrm{d}X_t = u_t^\theta(X_t)\,\mathrm{d}t. \qquad \blacktriangleright \text{ flow model} \tag{1.1}$$

As we learned, we want the neural network u_t^θ to equal the marginal vector field u_t^{target}. In other words, we would like to find parameters θ so that $u_t^\theta \approx u_t^{\text{target}}$.

In the following, we denote by $\text{Unif} = \text{Unif}[0,1]$ the uniform distribution on the interval [0, 1], and by $\mathbb{E}$ the expected value of a random variable. An intuitive way of obtaining $u_t^\theta \approx u_t^{\text{target}}$ is to use a mean-squared error, aka to use the *flow matching loss* defined as

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},\, x\sim p_t}\Big[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2\Big] \tag{1.2}$$

$$= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)}\Big[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2\Big], \tag{1.3}$$

where in the second line we used that sampling $x \sim p_t$ is equivalent to first sampling $z \sim p_{\text{data}}$ and then sampling $x \sim p_t(\cdot \mid z)$.

Intuitively, this loss says: First, draw a random time $t \in [0,1]$. Second, draw a random point z from our data set, sample from $p_t(\cdot \mid z)$ (e.g., by adding some noise), and compute $u_\theta^t(x)$. Finally, compute the mean-squared error between the output of our neural network and the marginal vector field $u_t^{\text{target}}(x)$.

Intractability of the marginal target. Unfortunately, we are not done here. While we do know the formula for u_t^{target} by the marginalization trick:

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x \mid z)\frac{p_t(x \mid z)\,p_{\text{data}}(z)}{p_t(x)}\,dz, \tag{1.4}$$

we cannot compute it efficiently because the above integral is intractable. Instead, we will exploit the fact that the *conditional* velocity field $u_t^{\text{target}}(x \mid z)$ is tractable. To do so, let us define the *conditional flow matching loss*

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)}\left[\left\|u_\theta^t(x) - u_t^{\text{target}}(x \mid z)\right\|^2\right]. \tag{1.5}$$

Note the difference to eq. (1.2): we use the conditional vector field $u_t^{\text{target}}(x \mid z)$ instead of the marginal vector field $u_t^{\text{target}}(x)$. As we have an analytical formula for $u_t^{\text{target}}(x \mid z)$, we can minimize the above loss easily. But wait, what sense does it make to regress against the conditional vector field if it is the marginal vector field we care about? As it turns out, by explicitly regressing against the tractable, conditional vector field, we are implicitly regressing against the intractable, marginal vector field. The next result makes this intuition precise.

Theorem 1.1 (Marginal vs. conditional flow matching)
The marginal flow matching loss equals the conditional flow matching loss up to a constant:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathcal{L}_{\text{CFM}}(\theta) + C,$$

where C is independent of θ. In particular,

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

Hence, minimizing $\mathcal{L}_{\text{CFM}}$ (e.g. with SGD) is equivalent to minimizing $\mathcal{L}_{\text{FM}}$.

Proof of Theorem 1.1. The proof works by expanding the mean-squared error into three components and removing constants:

$$\begin{aligned}
\mathcal{L}_{\text{FM}}(\theta) &\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},\, x\sim p_t}\left[\left\|u_t^\theta(x) - u_t^{\text{target}}(x)\right\|^2\right] \\
&\overset{(ii)}{=} \mathbb{E}_{t\sim\text{Unif},\, x\sim p_t}\left[\left\|u_t^\theta(x)\right\|^2 - 2u_t^\theta(x)^\top u_t^{\text{target}}(x) + \left\|u_t^{\text{target}}(x)\right\|^2\right] \\
&\overset{(iii)}{=} \mathbb{E}_{t\sim\text{Unif},\, x\sim p_t}\left[\left\|u_t^\theta(x)\right\|^2\right] - 2\mathbb{E}_{t\sim\text{Unif},\, x\sim p_t}\left[u_t^\theta(x)^\top u_t^{\text{target}}(x)\right] + \underbrace{\mathbb{E}_{t\sim\text{Unif}[0,1],\, x\sim p_t}\left[\left\|u_t^{\text{target}}(x)\right\|^2\right]}_{=:C_1} \\
&\overset{(iv)}{=} \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)}\left[\left\|u_t^\theta(x)\right\|^2\right] - 2\mathbb{E}_{t\sim\text{Unif},\, x\sim p_t}\left[u_t^\theta(x)^\top u_t^{\text{target}}(x)\right] + C_1.
\end{aligned}$$

Here (i) holds by definition, in (ii) we used the formula $\|a-b\|^2 = \|a\|^2 - 2a^\top b + \|b\|^2$, in (iii) we defined the constant C_1, and in (iv) we used the sampling procedure of p_t. Let us reexpress the second summand:

$$\begin{aligned}
\mathbb{E}_{t\sim\text{Unif},\,x\sim p_t}\left[u_t^\theta(x)^\top u_t^{\text{target}}(x)\right] &\overset{(i)}{=} \int_0^1 \int p_t(x)\, u_t^\theta(x)^\top u_t^{\text{target}}(x)\, dx\, dt \\
&\overset{(ii)}{=} \int_0^1 \int p_t(x)\, u_t^\theta(x)^\top \left[\int u_t^{\text{target}}(x \mid z) \frac{p_t(x \mid z)\, p_{\text{data}}(z)}{p_t(x)}\, dz\right] dx\, dt \\
&\overset{(iii)}{=} \int_0^1 \iiint u_t^\theta(x)^\top u_t^{\text{target}}(x \mid z)\, p_t(x \mid z)\, p_{\text{data}}(z)\, dz\, dx\, dt \\
&\overset{(iv)}{=} \mathbb{E}_{t\sim\text{Unif},\,z\sim p_{\text{data}},\,x\sim p_t(\cdot|z)}\left[u_t^\theta(x)^\top u_t^{\text{target}}(x \mid z)\right].
\end{aligned}$$

Here in (i) we expressed the expected value as an integral, in (ii) we used eq. (43), in (iii) we used the fact that integrals are linear, and in (iv) we expressed the integral as an expected value. Note that this was really the crucial step of the proof: the beginning of the equality used the marginal vector field $u_t^{\text{target}}(x)$, while the end uses the conditional vector field $u_t^{\text{target}}(x \mid z)$. Plugging this into the equation for $\mathcal{L}_{\text{FM}}$ yields:

$$\begin{aligned}
\mathcal{L}_{\text{FM}}(\theta) &\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},\,z\sim p_{\text{data}},\,x\sim p_t(\cdot|z)}\left[\left\|u_t^\theta(x)\right\|^2\right] - 2\mathbb{E}_{t\sim\text{Unif},\,z\sim p_{\text{data}},\,x\sim p_t(\cdot|z)}\left[u_t^\theta(x)^\top u_t^{\text{target}}(x \mid z)\right] + C_1 \\
&\overset{(ii)}{=} \mathbb{E}_{t\sim\text{Unif},\,z\sim p_{\text{data}},\,x\sim p_t(\cdot|z)}\left[\left\|u_t^\theta(x)\right\|^2 - 2u_t^\theta(x)^\top u_t^{\text{target}}(x \mid z) + \left\|u_t^{\text{target}}(x \mid z)\right\|^2\right] \\
&\quad \underbrace{- \mathbb{E}_{t\sim\text{Unif},\,z\sim p_{\text{data}},\,x\sim p_t(\cdot|z)}\left[\left\|u_t^{\text{target}}(x \mid z)\right\|^2\right]}_{=:C_2} + C_1 \\
&\overset{(iii)}{=} \mathbb{E}_{t\sim\text{Unif},\,z\sim p_{\text{data}},\,x\sim p_t(\cdot|z)}\left[\left\|u_t^\theta(x) - u_t^{\text{target}}(x \mid z)\right\|^2\right] + C_2 + C_1 \\
&\overset{(iv)}{=} \mathcal{L}_{\text{CFM}}(\theta) + \underbrace{C_2 + C_1}_{=:C}.
\end{aligned}$$

Here in (i) we plugged in the derived equation, in (ii) we added and subtracted the same value, in (iii) we again used the formula $\|a-b\|^2 = \|a\|^2 - 2a^\top b + \|b\|^2$, and in (iv) we defined a constant in θ. This finishes the proof. □

After training u_t^θ, we may sample by simulating the learned ODE

$$\mathrm{d}X_t = u_t^\theta(X_t)\, \mathrm{d}t, \quad X_0 \sim p_{\text{init}} \tag{1.6}$$

using Euler method, to obtain samples $X_1 \sim p_{\text{data}}$. This whole pipeline is called flow matching in the literature (Lipman et al., 2022; Liu et al., 2022; Lipman et al., 2024; Albergo et al., 2025). The training procedure is visualized in fig. 1. Let us now instantiate the conditional flow matching loss for the choice of Gaussian probability paths.

Example 1.1 (Flow matching for Gaussian conditional probability paths)

Let us return to the example of Gaussian probability paths $p_t(\cdot \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$, where we may sample from the conditional path via

$$\varepsilon \sim \mathcal{N}(0, I_d) \quad \Longrightarrow \quad x_t = \alpha_t z + \beta_t \varepsilon \sim \mathcal{N}(\alpha_t z, \beta_t^2 I_d) = p_t(\cdot \mid z). \tag{1.7}$$

Algorithm 1: Flow Matching Training Procedure (here for Gaussian CondOT path $p_t(x \mid z) = \mathcal{N}(tz, (1-t)^2)$)

```
Require: A dataset of samples $z \sim p_{\text{data}}$, neural network $u_t^\theta$
for each mini-batch of data do
    Sample a data example $z$ from the dataset
    Sample a random time $t \sim \text{Unif}_{[0,1]}$
    Sample noise $\varepsilon \sim \mathcal{N}(0, I_d)$
    Set $x = tz + (1-t)\varepsilon$                      // General: $x \sim p_t(\cdot \mid z)$

    Compute loss
        $\mathcal{L}(\theta) = \left\|u_t^\theta(x) - (z - \varepsilon)\right\|^2$        // General: $\|u_t^\theta(x) - u_t^{\text{target}}(x \mid z)\|^2$
    Update the model parameters $\theta$ via gradient descent on $\mathcal{L}(\theta)$
```

As we derived previously, the conditional vector field $u_t^{\text{target}}(x \mid z)$ is given by

$$u_t^{\text{target}}(x \mid z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x, \tag{1.8}$$

where $\dot{\alpha}_t = \partial_t \alpha_t$ and $\dot{\beta}_t = \partial_t \beta_t$ are the respective time derivatives. Plugging in this formula, the conditional flow matching loss reads

$$\begin{aligned}\mathcal{L}_{\text{CFM}}(\theta) &= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim\mathcal{N}(\alpha_t z, \beta_t^2 I_d)}\left[\left\|u_t^\theta(x) - \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z - \frac{\dot{\beta}_t}{\beta_t} x\right\|^2\right] \\ &\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \varepsilon\sim\mathcal{N}(0, I_d)}\left[\left\|u_t^\theta(\alpha_t z + \beta_t \varepsilon) - \left(\dot{\alpha}_t z + \dot{\beta}_t \varepsilon\right)\right\|^2\right].\end{aligned}$$

Here in (i) we plugged in eq. (1.7) and replaced x by $\alpha_t z + \beta_t \varepsilon$. Note the simplicity of $\mathcal{L}_{\text{CFM}}$: we sample a data point z, sample some noise ε, and then compute a mean squared error. Let us make this even more concrete for the special case $\alpha_t = t$ and $\beta_t = 1-t$. The corresponding probability path $p_t(\cdot \mid z) = \mathcal{N}(tz, (1-t)^2)$ is sometimes referred to as the (Gaussian) *CondOT probability path*. Then we have $\dot{\alpha}_t = 1$ and $\dot{\beta}_t = -1$, so that

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \varepsilon\sim\mathcal{N}(0, I_d)}\left[\left\|u_t^\theta\big(tz + (1-t)\varepsilon\big) - (z - \varepsilon)\right\|^2\right]. \tag{1.9}$$

Many famous state-of-the-art models have been trained using this simple yet effective procedure, e.g. *Stable Diffusion 3*, Meta's *Movie Gen Video*, and probably many more proprietary models. In Fig. 9, we visualize it in a simple example and in Algorithm 5 we summarize the training procedure.

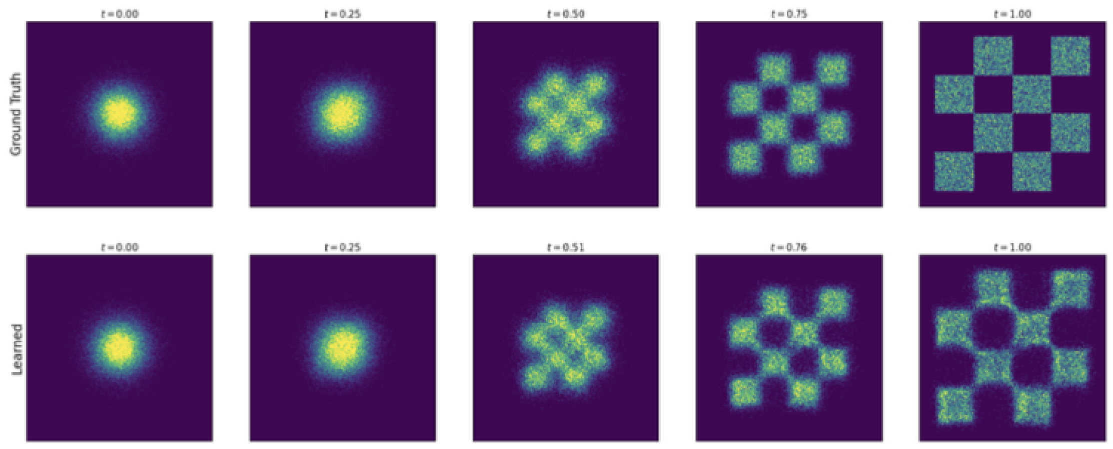

Figure 1: Illustration of Theorem 1.1 with a Gaussian CondOT probability path: simulating an ODE from a trained flow matching model. The data distribution is the chess board pattern (top right). Top row: Histogram from ground truth marginal probability path pt(x). Bottom row: Histogram of samples from flow matching model. As one can see, the top row and bottom row match after training (up to training error). The model was trained using an algorithm called class-free guidanced flow model, which we will see in the next class.

1.2 Score Matching

Let us extend the algorithm we just found from ODEs to SDEs. Remember we can extend the target ODE to an SDE with the same marginal distribution given by

$$dX_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2}\nabla \log p_t(X_t)\right] dt + \sigma_t\, dW_t, \qquad X_0 \sim p_{\text{init}}, \tag{1.10}$$

which implies

$$X_t \sim p_t \qquad (0 \leq t \leq 1). \tag{1.11}$$

Here u_t^{target} is the marginal vector field and $\nabla \log p_t(x)$ is the *marginal score function*, represented via the formula

$$\nabla \log p_t(x) = \int \nabla \log p_t(x \mid z) \frac{p_t(x \mid z)\, p_{\text{data}}(z)}{p_t(x)}\, dz. \tag{1.12}$$

To approximate the marginal score $\nabla \log p_t$, we can use a neural network that we call the *score network* $s_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$. In the same way as before, we can design a *score matching loss* and a *conditional score matching loss*:

$$\mathcal{L}_{\text{SM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)}\left[\left\|s_t^\theta(x) - \nabla \log p_t(x)\right\|^2\right], \qquad \blacktriangleright \text{ score matching loss,}$$

$$\mathcal{L}_{\text{CSM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)}\left[\left\|s_t^\theta(x) - \nabla \log p_t(x \mid z)\right\|^2\right], \qquad \blacktriangleright \text{ conditional score matching loss.}$$

Here again the difference is using the marginal score $\nabla \log p_t(x)$ versus using the conditional score $\nabla \log p_t(x \mid z)$. As before, we ideally would want to minimize the score matching loss, but cannot do so because we do not know $\nabla \log p_t(x)$. However, similarly as before, the conditional score matching loss is a tractable alternative.

Theorem 1.2 (Marginal vs. conditional score matching)
The score matching loss equals the conditional score matching loss up to a constant:

$$\mathcal{L}_{\mathrm{SM}}(\theta) = \mathcal{L}_{\mathrm{CSM}}(\theta) + C, \tag{1.13}$$

where C is independent of the parameters θ. Therefore, their gradients coincide:

$$\nabla_\theta \mathcal{L}_{\mathrm{SM}}(\theta) = \nabla_\theta \mathcal{L}_{\mathrm{CSM}}(\theta). \tag{1.14}$$

In particular, for the minimizer $\theta^\star$, it holds that $s_t^{\theta^\star} = \nabla \log p_t$.

Proof of Theorem 1.2. Note that the formula for $\nabla \log p_t$ looks the same as the formula for u_t^{target}. Therefore, the proof is identical to the proof of Theorem 1.1, replacing u_t^{target} with $\nabla \log p_t$. □

After training, for any diffusion coefficient $\sigma_t \geq 0$, we can sample by simulating

$$X_0 \sim p_{\mathrm{init}}, \qquad \mathrm{d}X_t = \left(u_t^\theta(X_t) + \frac{\sigma_t^2}{2} s_t^\theta(X_t)\right)\mathrm{d}t + \sigma_t \,\mathrm{d}W_t. \tag{1.15}$$

In theory, every σ_t should give samples $X_1 \sim p_{\mathrm{data}}$ at perfect training. In practice, we encounter two types of errors: (1) numerical errors from simulating the SDE imperfectly and (2) training errors (that is, the model u_t^θ is not exactly equal to u_t^{target}). Therefore, there is an optimal unknown noise level σ_t; this can be determined empirically by testing different values in practice. At first sight, it might seem to be a disadvantage that we have to learn both s_t^θ and u_t^θ if we want to use diffusion models as opposed to flow models. However, note that we can often directly learn s_t^θ and u_t^θ in a single network with two outputs, so that the additional computational effort is usually minimal. Further, as we will see now for the special case of the Gaussian probability path, s_t^θ and u_t^θ may be converted into one another so that we do not have to train them separately.

Remark 1.1 (Denoising diffusion models)
If you are familiar with diffusion models, you have probably encountered the term *denoising diffusion model.* This model has become so popular that most people nowadays drop the word "denoising" and simply use the term "diffusion model" to describe it. In the language of this document, these are simply diffusion models with Gaussian probability paths $p_t(\cdot \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$. However, it is important to note that this might not be immediately obvious if you read some of the first diffusion model papers: they use a different time convention (time inverted), so you need to apply an appropriate time re-scaling, and they construct their probability path via so-called *forward processes.*

Example 1.2 (Denoising diffusion models: Score matching for Gaussian probability paths)

First, let us instantiate the denoising score matching loss for the case $p_t(\cdot \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$. As we derived in eq. (28), the conditional score $\nabla \log p_t(x \mid z)$ has the formula

$$\nabla \log p_t(x \mid z) = -\frac{x - \alpha_t z}{\beta_t^2}. \tag{1.16}$$

Plugging in this formula, the conditional score matching loss becomes

$$\begin{aligned}
\mathcal{L}_{\text{CSM}}(\theta) &= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)}\left[\left\| s_t^\theta(x) + \frac{x - \alpha_t z}{\beta_t^2}\right\|^2\right] \\
&\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \varepsilon\sim\mathcal{N}(0,I_d)}\left[\left\| s_t^\theta(\alpha_t z + \beta_t \varepsilon) + \frac{\varepsilon}{\beta_t}\right\|^2\right] \\
&= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \varepsilon\sim\mathcal{N}(0,I_d)}\left[\frac{1}{\beta_t^2}\left\| \beta_t s_t^\theta(\alpha_t z + \beta_t \varepsilon) + \varepsilon\right\|^2\right].
\end{aligned}$$

Here in (i) we plugged in eq. (1.7) and replaced x by $\alpha_t z + \beta_t \varepsilon$. Note that the network s_t^θ essentially learns to predict the noise that was used to corrupt a data sample z. Therefore, the above training loss is also called *denoising score matching*, and it was one of the first procedures used to learn diffusion models.

It was soon realized that the above loss is numerically unstable for $\beta_t \approx 0$ close to zero (i.e., denoising score matching only works if a sufficient amount of noise is added). In some of the first works on denoising diffusion models (see *Denoising Diffusion Probabilistic Models* by Ho et al. (2020)), it was therefore proposed to drop the constant $1/\beta_t^2$ in the loss and reparameterize s_t^θ into a *noise predictor* network $\varepsilon_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ via

$$-\beta_t s_t^\theta(x) = \varepsilon_t^\theta(x).$$

This yields the DDPM training objective

$$\mathcal{L}_{\text{DDPM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \varepsilon\sim\mathcal{N}(0,I_d)}\left[\left\| \varepsilon_t^\theta(\alpha_t z + \beta_t \varepsilon) - \varepsilon\right\|^2\right]. \tag{1.17}$$

Beyond its simplicity, there is another useful property of the Gaussian probability path: By learning s_t^θ or ε_t^θ, we also learn u_t^θ automatically and the other way around.

Proposition 1.3 (Conversion formula for Gaussian probability paths)

For the Gaussian probability path $p_t(\cdot \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$, it holds that the conditional (resp. marginal) vector field can be converted into the conditional (resp. marginal) score via

$$u_t^{\text{target}}(x \mid z) = \left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t\right) \nabla \log p_t(x \mid z) + \frac{\dot{\alpha}_t}{\alpha_t} x, \tag{1.18}$$

$$u_t^{\text{target}}(x) = \left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t\right) \nabla \log p_t(x) + \frac{\dot{\alpha}_t}{\alpha_t} x. \tag{1.19}$$

Algorithm 2: Score Matching Training Procedure for Gaussian probability path

```
Require: A dataset of samples $z \sim p_{\text{data}}$, score network $s_t^\theta$ or noise predictor $\varepsilon_t^\theta$
for each mini-batch of data do
    Sample a data example $z$ from the dataset
    Sample a random time $t \sim \text{Unif}_{[0,1]}$
    Sample noise $\varepsilon \sim \mathcal{N}(0, I_d)$
    Set $x_t = \alpha_t z + \beta_t \varepsilon$                    // General case:  $x_t \sim p_t(\cdot \mid z)$

    Compute loss
        $\mathcal{L}(\theta) = \left\| s_t^\theta(x_t) + \frac{\varepsilon}{\beta_t} \right\|^2$        // General case:  $\|s_t^\theta(x_t) - \nabla \log p_t(x_t \mid z)\|^2$

        Alternatively: $\mathcal{L}(\theta) = \left\| \varepsilon_t^\theta(x_t) - \varepsilon \right\|^2$
    Update the model parameters $\theta$ via gradient descent on $\mathcal{L}(\theta)$
```

Here the formula for the above marginal vector field u_t^{target} is called the *probability flow ODE* in the literature (more precisely, the corresponding ODE).

Proof of Proposition 1.3. For the conditional vector field and conditional score, we can derive

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

Here in (i) we just did some algebra. By taking integrals, the same identity holds for the marginal flow vector field and the marginal score function:

$$\begin{aligned} u_t^{\text{target}}(x) &= \int u_t^{\text{target}}(x \mid z) \frac{p_t(x \mid z)\, p_{\text{data}}(z)}{p_t(x)}\, dz \\ &= \int \left[\left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t \right) \nabla \log p_t(x \mid z) + \frac{\dot{\alpha}_t}{\alpha_t} x \right] \frac{p_t(x \mid z)\, p_{\text{data}}(z)}{p_t(x)}\, dz \\ &\overset{(i)}{=} \left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t \right) \nabla \log p_t(x) + \frac{\dot{\alpha}_t}{\alpha_t} x, \end{aligned}$$

where in (i) we used the formula for $\nabla \log p_t(x)$. □

We can use the conversion formula to parameterize the score network s_t^θ and the vector field network u_t^θ into one another via

$$u_t^\theta = \left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t \right) s_t^\theta(x) + \frac{\dot{\alpha}_t}{\alpha_t} x. \tag{54}$$

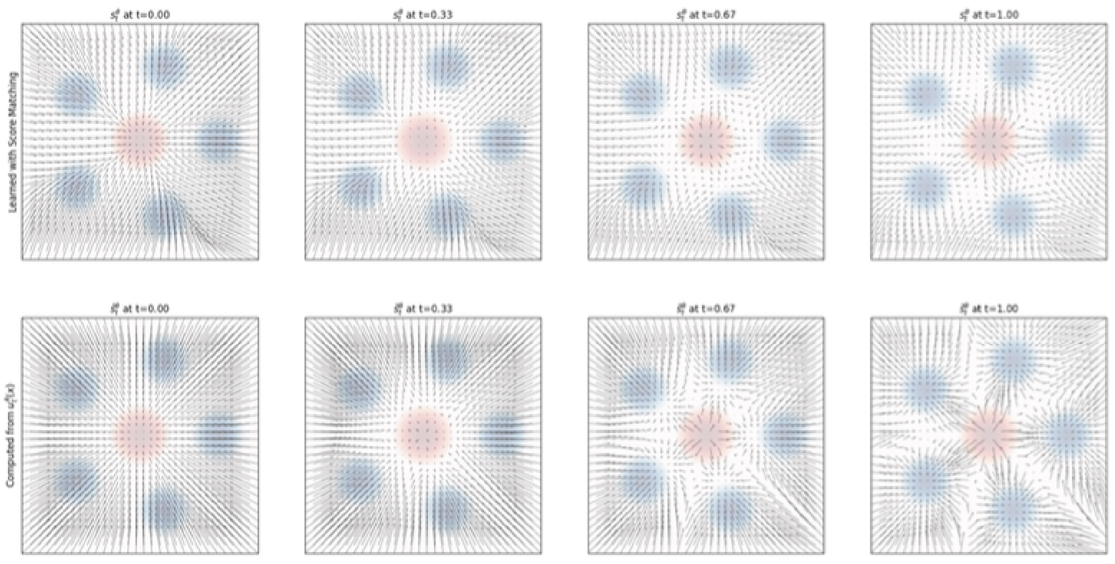

Figure 2: A comparison of the score, as obtained in two different ways. Top: A visualization of the score field $s_t^\theta(x)$ learned independently with score matching (see Algorithm 2). Bottom: A visualization of the score field $\widetilde{s}_t(x)$ parameterized using $u_t^\theta(x)$ as in eq. (54).

Similarly, so long as $\beta_t^2\dot{\alpha}_t - \alpha_t\dot{\beta}_t\beta_t \neq 0$ (which is always true for $t \in [0, 1)$), it follows that

$$s_t^\theta(x) = \frac{\alpha_t u_t^\theta(x) - \dot{\alpha}_t x}{\beta_t^2\dot{\alpha}_t - \alpha_t\dot{\beta}_t\beta_t}. \tag{1.20}$$

Using this parameterization, it can be shown that the denoising score matching and the conditional flow matching losses are the same up to a constant. We conclude that **for Gaussian probability paths there is no need to separately train both the marginal score and the marginal vector field**, as knowledge of one is sufficient to compute the other. In particular, we can choose whether we want to use flow matching or score matching to train the model. In Fig. 2, we compare visually the score as approximated using score matching and the parameterized score using eq. (54).

If we have trained a score network s_t^θ, we know by eq. (1.15) that we can use an arbitrary $\sigma_t \geq 0$ to sample from the SDE

$$X_0 \sim p_{\text{init}}, \qquad dX_t = \left[\left(\frac{\beta_t^2\dot{\alpha}_t}{\alpha_t} - \beta_t\dot{\beta}_t + \frac{\sigma_t^2}{2}\right) s_t^\theta(x) + \frac{\dot{\alpha}_t}{\alpha_t}\, x\right] dt + \sigma_t\, dW_t \tag{1.21}$$

to obtain samples $X_1 \sim p_{\text{data}}$ (up to training and simulation error). This corresponds to stochastic sampling from a denoising diffusion model.

1.3 A Guide to the Diffusion Model Literature

There are several equivalent viewpoints in the literature, and it is useful to know the translations.

Discrete time vs. continuous time. Early diffusion models were introduced in discrete time as Markov chains; later work showed these correspond to continuous-time SDEs. The continuous-time view avoids committing to a discretization during training and leads to cleaner identities (equalities rather than bounds).

"Forward process" vs. probability paths. Classic diffusion papers describe a noising SDE (a "forward process") that maps $z \sim p_{\text{data}}$ toward a Gaussian as time increases. In practice, training only requires the closed-form conditional distribution $x_t \mid x_0 = z$, which for commonly used forward drifts leads exactly to Gaussian probability paths.

Time reversal vs. Fokker–Planck / continuity. Training targets can be derived either via time reversal arguments or directly from the Fokker–Planck / continuity equations. For generative modeling we typically care about the endpoint X_1, so many sampling procedures use alternative dynamics (e.g. probability flow ODE) that are not literal time reversals but share the same marginals.

Summary 1.1 (Training the generative model)
Flow matching consists of training a neural network u_t^θ via minimizing the *conditional flow matching loss*

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{z\sim p_{\text{data}},\, t\sim\text{Unif},\, x\sim p_t(\cdot|z)}\left[\left\|u_t^\theta(x) - u_t^{\text{target}}(x \mid z)\right\|^2\right], \tag{1.22}$$

where $u_t^{\text{target}}(x \mid z)$ is the conditional vector field (see Algorithm 5). After training, one generates samples by simulating the corresponding ODE (see Algorithm 1). To extend this to a diffusion model, we can use a score network s_t^θ and train it via *conditional score matching*

$$\mathcal{L}_{\text{CSM}}(\theta) = \mathbb{E}_{z\sim p_{\text{data}},\, t\sim\text{Unif},\, x\sim p_t(\cdot|z)}\left[\left\|s_t^\theta(x) - \nabla \log p_t(x \mid z)\right\|^2\right], \tag{1.23}$$

also known as the denoising score matching loss. For every diffusion coefficient $\sigma_t \geq 0$, simulating the SDE (e.g., via Algorithm 2)

$$X_0 \sim p_{\text{init}}, \qquad dX_t = \left[u_t^\theta(X_t) + \frac{\sigma_t^2}{2}\, s_t^\theta(X_t)\right] dt + \sigma_t\, dW_t \tag{1.24}$$

will result in generating approximate samples from p_{data}. One can empirically find the optimal choice of $\sigma_t \geq 0$.

Gaussian probability paths. For the special case of a Gaussian probability path $p_t(\cdot \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$, the conditional score matching loss is also called *denoising score matching.* This loss and the conditional flow matching loss are then given by

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

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

In this case, there is no need to train s_t^θ and u_t^θ separately, as we can convert them post training via the formula

$$u_t^\theta(x) = \left(\frac{\beta_t^2\dot{\alpha}_t}{\alpha_t} - \beta_t\dot{\beta}_t\right) s_t^\theta(x) + \frac{\dot{\alpha}_t}{\alpha_t}\, x. \tag{1.25}$$

Also here, after training we can simulate the SDE in eq. (62) via Algorithm 2 to obtain samples X_1.

Denoising diffusion models. Denoising diffusion models are diffusion models with Gaussian probability paths. For this reason, it is sufficient for them to learn either u_t^θ or s_t^θ, as these quantities can be converted into one another.

References

Albergo, M., Boffi, N. M., and Vanden-Eijnden, E. (2025). Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26(209):1–80.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. (2022). Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*.

Lipman, Y., Havasi, M., Holderrieth, P., Shaul, N., Le, M., Karrer, B., Chen, R. T., Lopez-Paz, D., Ben-Hamu, H., and Gat, I. (2024). Flow matching guide and code. *arXiv preprint arXiv:2412.06264*.

Liu, X., Gong, C., and Liu, Q. (2022). Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*.