Deep Learning Practical Work 3a
Flow Matching

Homework

Data, code, and PDF version of this file are available at
`https://rdfia.github.io`

Goals

The goal of this practical work is to explore the flow matching algorithm for image synthesis.

Section 1 – Theoretical foundation

Flow matching Lipman et al. (2022) is an algorithm for generation of data, which models the generation as an iterative process. The goal, as is the case for all random generation of data, is to transform a sample from a distribution μ_0, into a sample from another distribution μ_1, which we cannot sample directly (because it is too complicated). In the case of Flow Matching, this is done by training a neural network to predict the *velocity* v_t of a *flow* $\phi_t : \mathbb{R}^d \mapsto \mathbb{R}^d$, with $t \in (0, 1)$. Originally, flows were designed to represent fluid dynamics: it is a vector field which indicates, given an input particle's position at time 0, the position of that particle at time t if it followed the motion of the fluid. In terms of image generation, $X_0 \sim \mu_0$ is the input particle, and $X_1 = \phi_1(X_0)$. The goal of Flow Matching is to define the flow and its velocity such that $X_1 \sim \mu_1$.

Mathematically, the flow is defined with the following differential equation:

$$\begin{aligned} \frac{d\phi_t(x)}{dt} &= v_t(x) && \text{Velocity field} \\ \phi_0(x) &= x && \text{Starting point: identity at 0.} \end{aligned} \tag{1}$$

This indeed says that v_t is the velocity of ϕ_t: the flow is completely specified by v_t. Thus, the main question is how to define v_t such that $X_1 \sim \mu_1$.

The main idea of Flow Matching is to use construct the flow using *simple (straight) paths* between samples X_0 and X_1:

$$X_t := (1 - t)X_0 + tX_1. \tag{2}$$

Subsequently, v_t is defined as:

$$v_t(x) := \mathbb{E}_{X_0 \sim \mu_0, X_1 \sim \mu_1} \left[X_1 - X_0 | X_t = x \right]. \tag{3}$$

This is the average velocity vector of simple paths between randomly chosen samples X_0 and X_1, such that the path passes through $X_t = x$. Thus, the Flow Matching training algorithm consists in training a neural

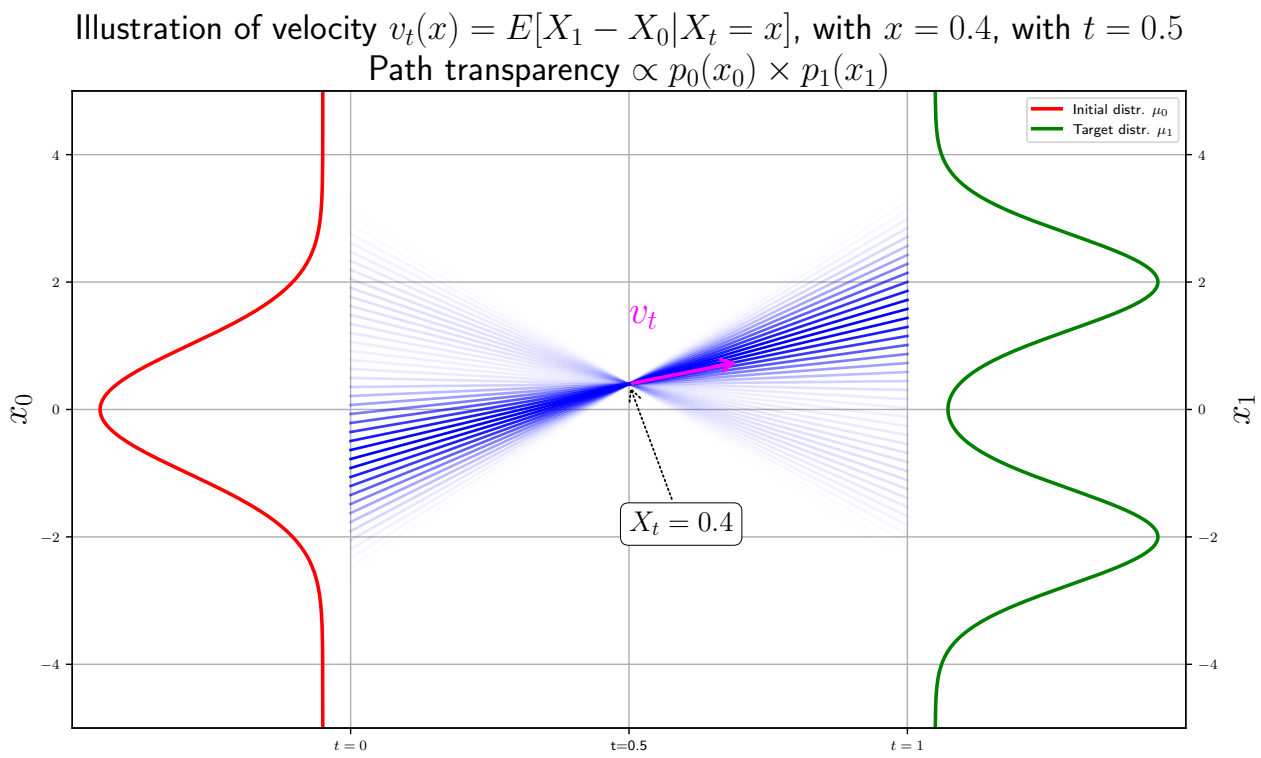

Figure 1: Illustration of velocity field v_t in Flow Matching.

network f_t to predict v_t by minimising the following loss:

$$\mathbb{E}_{X_0 \sim \mu_0, X_1 \sim \mu_1} \left[((X_1 - X_0) - f_t(X_t))^2 \right]. \tag{4}$$

This neural network is often a U-Net, and the time step t is given to the network by concatenating it as an extra dimension to the input.

More generally, we can define paths between X_0 and X_1 using:

$$X_t := \alpha_t X_0 + \beta_t X_1, \tag{5}$$

with $\alpha_0 = \beta_1 = 1$ and $\alpha_1 = \beta_0 = 0$ (border conditions). In this case, v_t is defined as:

$$v_t(x) := \mathbb{E}_{X_0 \sim \mu_0, X_1 \sim \mu_1} \left[\frac{dX_t}{dt} | X_t = x \right]. \tag{6}$$

The couple (α_t, β_t) is known as the *schedule* of the algorithm. This is closely linked to the variance schedule of Diffusion Models. The schedule $(\alpha_t = 1 - t, \beta_t = t)$ is known as the *linear schedule*, since it represents a linear interpolation between X_0 and X_1.

Finally, once we have trained f_t, we can generate samples X_1 by evaluating $\phi_1(X_0)$ using a numerical integration (Euler) scheme, using a series of timesteps t_i:

$$\phi_1(X_0) = \int_0^1 v_t(X_0) dt \approx X_0 + \sum_{i=1}^{N-1} (t_{i+1} - t_i) f_{t_i}(X_{t_i}) \tag{7}$$

Section 2 – Implementation part 1: 2D

In this part, we will implement the Flow Matching algorithm in the case where the data are in $d = 2$ dimensions. We will use the following MLP architecture:

- fully-connected, $d+1$ input neurons, n_{hidden} output neurons, followed by ReLU
- fully-connected, n_{hidden} input neurons, n_{hidden} output neurons, followed by ReLU
- fully-connected, n_{hidden} input neurons, n_{hidden} output neurons, followed by ReLU
- fully-connected, n_{hidden} input neurons, d output neurons, followed by ReLU

There are $d+1$ input neurons because the last dimension contains the time-step. We will first use the "moons" dataset to represent the target distribution μ_1. Fill in the code provided to carry out generation for this dataset.

Questions

1. A popular schedule of Diffusion Models/Flow Matching is the "cosine schedule". This is of the form $\alpha_t = \cos(at), \beta_t = \sin(bt)$. Determine the smallest a and b such that the border conditions are respected;
2. Calculate $\frac{dX_t}{dt}$ in the case of the cosine schedule;

Implement the cosine schedule in the code.

Schedules are often categorised as "variance preserving" or not. This originally comes from the Diffusion Model literature where noise is progressively added to the data, to create a Markov chain. In the case of Flow Matching, if the interpolation does not change the variance of the data, in the case where μ_0 and μ_1 have the same variance, then we can call the schedule variance preserving.

3. Consider the linear schedule between two Gaussian random variables: $X_t = (1-t)X_0 + tX_1$, with $X_0 \sim \mathcal{N}(0,1)$ and $X_1 \sim \mathcal{N}(1,1)$. By looking at the case $t = \frac{1}{2}$, show that the schedule is not variance preserving;
4. Show that the cosine schedule is variance preserving (using the same technique);
5. ★ Why is it an advantage to be variance preserving (think about the geometry of the generation process) ?

Now, modify the code so that the target distribution μ_1 is a Gaussian distribution of mean $(1,1)$ and identity covariance (variance of 1 in each dimension and 0 covariance). Carry out the generation with both linear and cosine schedules.

6. Do the experimental observations confirm your theoretical predictions concerning the variance preservation of the schedules ? To be sure about this, you can calculate the variance of the generated data at each time step.

Section 3 – Implementation part 2: images (mnist)

In this part, we will implement the Flow Matching algorithm in the case of images, from the mnist database. Note that in this case, we have a database of a finite size, instead of the Moons and Gaussian data which can be generated infinitely. Therefore we will iterate over a certain number of *epochs*, rather than iterations.

We will use the following U-Net architecture, with skip connections, for mnist input images of size $28 \times 28 \times 1$. The "DoubleConv" layer is already defined in the code, and consists of a convolution, a ReLU, a convolution and a ReLU:

- **Input:**
  - Input image $x_t \in \mathbb{R}^{B\times 1\times 28\times 28}$
  - Scalar time $t \in \mathbb{R}^B$ expanded to a time map and concatenated:

$$x_{\text{in}} = \text{concat}(x_t, t\text{-map}) \in \mathbb{R}^{B\times(2)\times 28\times 28}$$

- **Encoder:**
  - DoubleConv$(2 \to 32)$ (28×28)
  - MaxPool(2×2) (14×14)
  - DoubleConv$(32 \to 64)$ (14×14)
  - MaxPool(2×2) (7×7)
- **Middle layer ("Bottleneck"):**
  - DoubleConv$(64 \to 64)$ (7×7)
- **Decoder:**
  - TransposedConv$(64 \to 64)$, kernel 2×2 stride 2 (14×14)
  - Add skip connection from encoder feature map (e_2)
  - DoubleConv$(64 \to 32)$ (14×14)
  - TransposedConv$(32 \to 32)$, kernel 2×2, stride 2 (28×28)
  - Add skip connection from encoder feature map (e_1)
  - DoubleConv$(32 \to 32)$ (28×28)
- **Output:**
  - 1×1 convolution $(c_1 \to C_{\text{out}})$
  - Output tensor $\in \mathbb{R}^{B\times C_{\text{out}}\times 28\times 28}$

Implement the previous architecture, and the training code, to generate images from mnist. You may test the two different schedules to compare the results.

7. What is the major advantage of Diffusion Models/Flow Matching over Generative Adversarial Networks (GANs) ?
8. What is the computational disadvantage of Diffusion Models/Flow Matching in comparison to GANs ?
9. ★ GANs generate images in one "step", ie with one neural network, whereas Flow Matching does this in several steps. In light of this, if we allow each method to have a fixed number of neural network parameters, why do you think the Flow Matching algorithm produces better results ?

3.1 Image-to-image translation

In the Flow Matching, there is no particular reason why μ_0 should be a Gaussian distribution: as long as one can sample from it (or draw samples from a database), Flow Matching can be applied[1]. This leads to the idea of "Image-to-Image Translation" Isola et al. (2017). Image-to-image translation methods take an input image from one domain (for example a photograph) and converts it to another domain (such as a sketch).

Keeping the architecture, optimiser and hyper-parameters the same, modify your code to carry out image-to-image translation between the following datasets:

[1] Whether this is theoretically valid is another question

- Source dataset: "Fashion mnist";
- Target dataset: mnist;

The data loader for fashion mnist is given. You will have to modify the trajectory sampling function too, to sample from the source distribution. Note: you may have to wait about 100 epochs to get a reasonable result.

References

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017. URL `http://openaccess.thecvf.com/content_cvpr_2017/html/Isola_Image-To-Image_Translation_With_CVPR_2017_paper.html`.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow Matching for Generative Modeling. September 2022. URL `https://openreview.net/forum?id=PqvMRDCJT9t`.