Learning Dynamics from Samples

Anne Weiß

Born 29th September 2000 in Darmstadt, Germany

6th September 2024

Master's Thesis Mathematics

Advisor: Prof. Dr. Jochen Garcke

Second Advisor: Prof. Dr. Barbara Verfürth

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Acknowledgements

I would like to express my sincere gratitude to those who made this thesis possible through their guidance and support.

I am especially grateful to my supervisor, Prof. Dr. Garcke, for the opportunity to write a thesis that bridges Machine Learning and Mathematics. His invaluable feedback, guidance, and experience have been essential in the development of this work.

I would also like to extend my thanks to the Deutschlandstipendium and my donor, Jörg Bewersdorff, for their financial support during my studies.

Finally, I am deeply thankful to my partner, Ben, for his emotional support and thorough proofreading.

Contents

Notation and Abbreviations

Notation

$\mathbb{R}$	Set of real numbers
x_t	Position of the particle at time t
X_t	Random position of the stochastic Process $\boldsymbol{X} = (X_t)_t$ at time t
u_t	vector field or drift term
$g(t)$	diffusion term in a SDE
μ	Initial distribution
ν	Target distribution
$\mathbb{E}_{x\sim q}[f(x)]$	Expected value with respect to the probability density function q. Suppose $X \sim q$ is a random variable distributed according to q, then $\mathbb{E}_{x\sim q}[f(x)] = \mathbb{E}[f(X)]$
$\mathcal{N}(\mu, \sigma^2)$	Univariate normal distribution with mean μ and variance σ^2
$\mathcal{N}(x \mid \mu, \sigma^2)$	Probability density function of the univariate normal distribution
$\mathcal{U}([0,1])$	Uniform distribution over the interval $[0,1]$
$\phi\#f(x)$	Pushforward of f defined by $\phi\#f(x) = f(\phi^{-1}(x)) \det\left[\frac{\partial \phi^{-1}}{\partial x}(x)\right]$
$\phi\#\pi(A)$	Pushforward of the probability measure π by $\phi : \Omega \to \mathcal{X}$, defined by $\phi\#\pi(A) = \pi(\phi^{-1}(A))$ for all measurable sets $A \subseteq \mathcal{X}$
Δf	Laplacian of the function f defined by $\Delta f = \nabla \cdot (\nabla f)$
$O(f)$	Big O notation
$W_p(\mu, \nu)$	p-Wasserstein distance between μ and ν as defined in (2.2.2)
$(W_t)_t$	Brownian motion
$\delta_x(A)$	Dirac measure, which is 1 if $x \in A$ and 0 otherwise for all measurable sets A
I_n or I	$n \times n$ identity matrix. If the dimension is clear we sometimes write I instead of I_n
$\mathcal{M}(\mathcal{X})$	The set of all Radon measures on the space $(\mathcal{X}, \mathcal{F})$, where $\mathcal{F}$ is a σ-algebra on $\mathcal{X}$
$\mathcal{M}_+(\mathcal{X})$	The set of positive Radon measures on the space $\mathcal{X}$
$\mathcal{P}(\mathcal{X})$	The set of probability measures on the space $\mathcal{X}$
$\mathcal{P}_2(\mathcal{X})$	The set of probability measures on the space $\mathcal{X}$ which have finite second moment

Abbreviations

scRNA-seq	single-cell RNA sequencing
CNF	Continuous Normalizing Flow
OT	Optimal Transport
UOT	Unbalanced Optimal Transport
ODE	Ordinary Differential Equation
SDE	Stochastic Differential Equation
CFM	Conditional Flow Matching
I-CFM	Independent Conditional Flow Matching as introduced in Section 3.2.1
OT-CFM	Optimal Transport Conditional Flow Matching as introduced in Section 3.2.2
SB-CFM	Schrödinger Bridge Conditional Flow Matching as introduced in Section 3.2.3
UOT-CFM	Unbalanced OT Conditional Flow Matching as introduced in Section 3.3
$[SF]^2M$	Simulation free Score and Flow Matching as introduced in Section 5.3.2
AM	Action Matching as introduced in Chapter 4
EAM	Entropic Action Matching as introduced in Section 5.2
I-AM/I-EAM	Variant of (entropic) Action Matching in which we interpolate between data points in a very simple way the method is introduced in Section 6.1

Chapter 1

Introduction

Understanding the complex dynamics that control the evolution of particles or individuals is fundamental in various disciplines, from natural science to Machine Learning. Learning underlying dynamics from static temporal snapshots of data is a difficult task. An example of temporal snapshots of a dynamic is given in Figure 1.1. This data structure arises naturally in areas as diverse as single-cell biology, quantum mechanics and generative modeling.

Single cell biology allows us to understand the behavior of individual cells. One powerful technique here is single-cell RNA sequencing (scRNA-seq) which helps us to analyze gene expressions at the single-cell level. It can identify which genes are expressed in each cell and quantify their expression levels. We are especially interested in understanding the cell dynamics which helps us to understand processes such as cell differentiation, development and disease progression. Unfortunately the cells are destroyed when measuring them. Thus, we cannot follow an individual cell over time. Instead, we only have static snapshots of cells and their gene expressions at different time points. At each time point we can assume that the cells are independently sampled. The data is thus cross-sectional, i.e. we have several independent samples at each point in time.

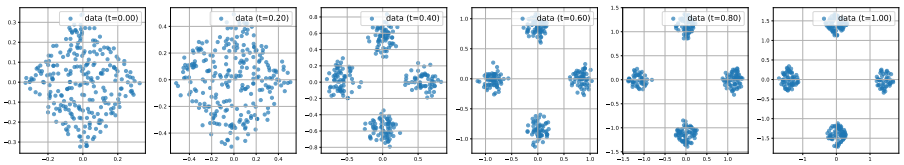

Figure 1.1: Temporal snapshots of an evolution of diamonds. We start with one diamond which then splits into four diamonds over time. We want to learn the underlying dynamics by only having access to the temporal snapshots.

In quantum mechanics measuring the wave function collapses the function. Thus, we only have temporal snapshots of data to describe the quantum state of a system.

Generative modeling aims at learning the patterns and distribution of the data in order to generate new data samples. A lot of research has been done over the last years. Many modern methods rely on interpolating between a simple prior distribution, often a Gaussian, and the data distribution. By this, we can sample from the simple prior distribution and then push the sample through some process to obtain a sample from the data distribution. Two common approaches here are *Normalizing Flows* and *Diffusion Models*. Normalizing flows transform the prior through a series of invertible mappings to obtain the data distribution [RM15]. *Continuous Normalizing Flows* (CNFs) use a continuous transformation which is described by a Neural Ordinary Differential Equation [Che+18]. As this approach simulates the ODE during training it is complex to scale it to large datasets. Diffusion models on the other hand start with the data points and gradually corrupt them with noise until they arrive at a simple prior distribution. They then need to learn to reverse the diffusion process. For this they use a simple regression training objective and do not need to simulate the diffusion process in training. This made Diffusion Models very popular. Examples for such models are *Score Matching with SDEs* [Son+21] or *Denoising Diffusion Probabilistic Models* [HJA20]. The methods discussed in this thesis are capable of learning probability paths that interpolate between empirical distributions that change over time. This allows us to use these methods to learn an interpolation path between a prior distribution and the data distribution, enabling us to generate samples from the data distribution.

In this work we mainly want to investigate the methods *Action Matching* [Nek+23b] and *Flow Matching* [Lip+22], [Ton+24]. Both try to train CNFs in a simulation-free way. This means we do not want to simulate the process during training by for example solving an ODE. We also look at extensions of these methods, for example for stochastic dynamics. Figure 1.2 shows how the methods we introduce in this thesis are interconnected.

To make our objective mathematically more rigorous we suppose that the dynamics, we want to learn, follows an Ordinary Differential Equation (ODE) or a Stochastic Differential Equation (SDE). The particles evolve in the time interval $[t_0, t_n]$. In the deterministic case let $(x_t)_{t\in[t_0,t_n]}$ be the process of particles, where x_t is the location of the particle at time t. We suppose that the particles evolve according to the ODE:

$$\mathrm{d}x_t = u_t(x_t)\,\mathrm{d}t,$$

where $x_{t_0} \sim \mu$ and $x_{t_n} \sim \nu$. We further assume that we have access to samples $x^j_{t_0}$ from μ for $j = 1, \dots n_{t_0}$ and $x^k_{t_n}$ from ν for $k = 1, \dots, n_{t_n}$. If $n > 1$ we further have access to samples $x^j_{t_i}$ for $j = 1, \dots, n_{t_i}$ and $i = 1, \dots n-1$. In the stochastic case the particles are described by the stochastic process $\boldsymbol{X} = (X_t)_{t\in[t_0,t_n]}$, where the (random) location of a particle at time t is described by the random variable X_t. The stochastic process evolves according to the SDE

$$\mathrm{d}X_t = u_t(X_t)\,\mathrm{d}t + g(t)\,\mathrm{d}W_t, \tag{1.0.1}$$

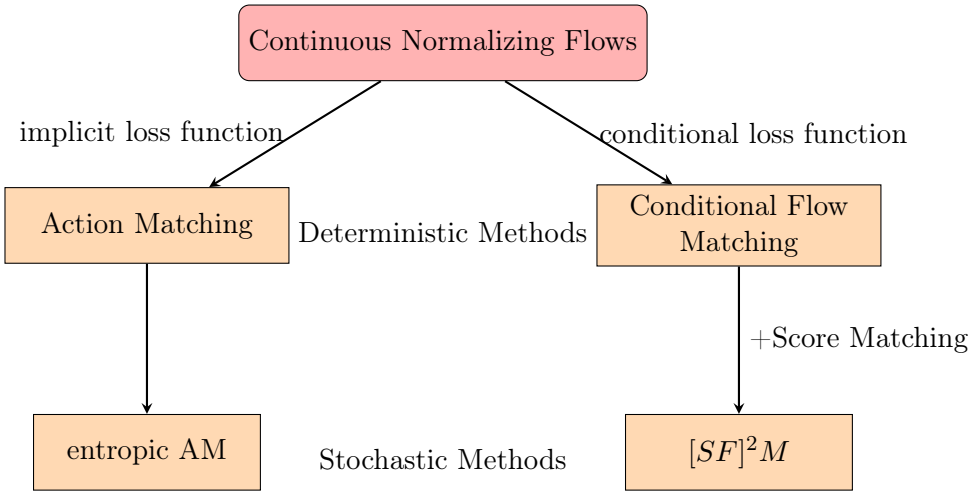

Figure 1.2: Connections between the methods we investigate in this Thesis.

where $g(t)$ is known. Assuming $g(t)$ is known imposes a limitation on the model, but given the cross-sectional nature of the data, this assumption is necessary. Despite this constraint, incorporating stochastic dynamics offers certain advantages. For instance, SDEs are known to explore the state space more thoroughly in generative modeling. Additionally, many processes are naturally modeled as SDEs, such as trajectory inference in single-cell analysis, which we discuss later.

Why Do Standard Time Series Methods Fail?

On the first glimpse the data has a time series structure. Time series have been studied for hundreds of years and many effective methods have been developed for forecasting. Standard methods include ARIMA models or time series regression [Hyn18]. More advanced models include Recurrent Networks, LSTMs [SH+97] or even Transformer [Wen+23]. All of those methods, however, require us to have knowledge of trajectories of the data. We, on the other hand, want to deal with static snapshots of data. One possibility would be to view the empirical distributions $\hat{p}_{t_i}$ as data points in Wasserstein space. This, however is an infinite dimensional space and poses a very complex problem. Thus, we will not use time series methods in the following.

Objective of This Work

In this work we investigate the methods of Action Matching and Flow Matching. While they solve the same problem, the two methods use different training objectives to tackle the tasks.

We want to investigate the assumptions that they have on the data and experimentially test how these methods perform in different data regimes, for example in high dimensional settings. We further want to look at stochastic extensions and their predictive performances compared to their deterministic counterparts. To gain a deeper understanding we developed several artificial datasets. In the end we will test the methods on a real single-cell dataset. We will further propose some adaptions to make the methods more robust in different settings.

Related Work

Both methods Flow Matching and Action Matching were inspired by CNFs [Che+18]. We introduce CNFs in Section 2.1. They have applications in generative modeling [Gra+19] and single-cell biology [Ton+20]. A big computational bottleneck of CNFs is, however, that to propagate data through the model, we need to solve an ODE numerically. Because of that CNFs scale poorly to large datasets. Many attempts were made to speed them up and make them more robust [Qua+20], [Yan+20], yet they still remain computationally expensive. Another approach is to regularize CNFs with the help of Optimal Transport. This yields straighter paths and thus the trajectories are easier to numerically integrate [Onk+21].

Dynamical Optimal Transport provides us with a way to transport one measure to another by geodesics. This gives us a very good idea how we might want to transport one measure to the next. Schrödinger bridges [Lé13] introduce an entropic regularized version of this problem. We discuss both concepts in Section 2.2.

We discuss related methods to infer trajectories in single-cell biology in Chapter 7.

Structure of the Thesis

In Chapter 2 we introduce Continuous Normalizing Flows and Optimal Transport to build a mathematical foundation. Here, we will learn about mathematical ways to transport one measure to another. In Chapter 3 and Chapter 4 we explain the two main methods of this thesis: Flow Matching and Action Matching. Afterwards in Chapter 5 we explore Stochastic extensions of the methods. Finally in Chapter 6 we test our models on different artificial data settings while in Chapter 7 we apply them to real single-cell data.

Contributions

- We introduce UOT-CFM which is an adaptation of Optimal Transport Conditional Flow Matching (OT-CFM). This adaption is more robust to outliers and cluster structures.
- We introduce independent Action Matching (I-AM) and independent entropic Action Matching (I-EAM). Those are variations of AM/EAM which interpolate the data artificially between two discrete time points for which samples are available.
- We use the Hutchinson trace estimator to make entropic Action Matching more robust and scalable.
- By thorough testing on different artificial datasets we develop guidelines when it is best to use each method.

Chapter 2

Preliminaries

2.1 Continuous Normalizing Flows

CNFs are a type of flow-based generative model. Originally, they were proposed for image generation. They transform a known distribution μ (usually a simple one like the standard Gaussian distribution) to another distribution ν (usually a more complex distribution) by applying a transformation to it. This transformation needs to be learned. CNFs assume that this transformation is continuous, more precisely that it can be defined by an ODE. That means we can describe the change in the input variable x_t by a smooth, time-dependent vector field $u : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ such that

$$\mathrm{d}x_t = u_t(x_t)\,\mathrm{d}t. \tag{2.1.1}$$

This transformation is invertible. In order to learn the transformation we parametrize u by a neural network $u_\theta(t, x)$ with learnable parameter θ. In contrast to *Generative Adversarial Networks* or *Variational Autoencoders* CNFs allow us to compute the exact likelihood of samples. The likelihood is calculated by the Instantaneous Change of Variables formula:

Theorem 2.1.1 (Theorem 1 in [Che+18])**.** *Let x_t be a finite continuous random variable with probability density function $p_t(x_t)$. Let $\frac{\mathrm{d}x}{\mathrm{d}t} = u(t, x_t)$ describe the continuous-in-time transformation of x_t. We assume that u is uniformly Lipschitz continuous in x and continuous in t, then the change in log probability also follows a differential equation,*

$$\frac{\partial \log p_t(x_t)}{\partial t} = -tr\left(\frac{du}{dx_t}\right).$$

CNFs are trained by Maximum Likelihood Estimation, i.e. we solve the ODE to transform the simple distribution to the more complex one and then calculate the log likelihood. More details can be found in [Che+18]. Solving an ODE in every training step is computational expensive

and scales poorly to high dimensions. In Chapter 3 and Chapter 4 we introduce simulation free objectives to this problem.

Let $\phi(t, x)$ be the solution of the ODE (2.1.1) with initial condition $\phi(0, x) = x$. The map ϕ is also called flow. In the following we use $\phi_t(x)$ and $\phi(t, x)$ interchangeably. The flow induces a marginal probability path via a pushforward $p_t := [\phi_t]\#p_0$, which is the density at time t when we transported p_0 along u. We define the pushforward by the Change of Variables formula:

$$[\phi_t]\#p_0(x) = p_0(\phi_t^{-1}(x)) \det \left[\frac{\partial \phi_t^{-1}}{\partial x}(x) \right]. \tag{2.1.2}$$

We can view the density as a function $p : [0, 1] \times \mathbb{R}^d \to \mathbb{R}$. This function is characterized by the *continuity equation*:

$$\frac{\partial p}{\partial t} = -\nabla \cdot (p_t u_t) \tag{2.1.3}$$

with initial condition p_0. Here $\nabla\cdot$ denotes the divergence operator, i.e. for $v : \Omega \to \mathbb{R}^d$ the divergence is given by

$$\nabla \cdot v(x) = \frac{\partial v_1}{\partial x_1}(x) + \cdots + \frac{\partial v_d}{\partial x_d}(x).$$

The continuity equation describes the conservation of mass principle in physics. If the continuity equation is fulfilled, we say that u generates p. Further we understand (2.1.3) in the distributional sense, i.e. $\forall \varphi \in C_c^\infty((0, T) \times \mathbb{R}^d)$ it holds

$$\int_0^T \int_{\mathbb{R}^d} (\partial_t \varphi(t, x) + \langle u_t(x), \nabla_x \varphi(t, x) \rangle) \, \mathrm{d}p_t(x) \, \mathrm{d}t = 0. \tag{2.1.4}$$

The case of Gaussian flows. Let $p_t(x) = \mathcal{N}(x \mid \mu_t, \sigma_t^2)$. While there can be more than one flow that produces p_t, a simple one is given by

$$\phi_t(x_0) = \mu_t + \sigma_t \left(\frac{x_0 - \mu_0}{\sigma_0} \right). \tag{2.1.5}$$

This is the unique integration map to the vector field

$$u_t(x) = \frac{\sigma_t'}{\sigma_t}(x - \mu_t) + \mu_t'. \tag{2.1.6}$$

Here σ_t' and μ_t' are the time derivatives of σ_t and μ_t. This vector field generates the Gaussian path with initial condition $\mathcal{N}(\mu_0, \sigma_0^2)$. For a proof of this result see [Lip+22].

2.1.1 Sampling from CNFs

In order to sample from the CNF we need to solve the ODE (2.1.1). Various numerical techniques exist for solving ODEs, and one of the simplest is the Euler method, which we briefly introduce

here. In this method, we initialize y_0 according to μ, and for each step $i = 0, \dots, n-1$ with a step width $h = \frac{1}{n}$, we iteratively compute y_{i+1} as follows:

$$y_{i+1} = y_i + h u_{ih}(y_i). \tag{2.1.7}$$

Denoting $y_1^{x_0}$ as the true solution at time $t = 1$ when starting at x_0, and $\hat{y}_1$ as the solution estimated by the Euler scheme with step width h, we find:

$$|y_1^{x_0} - \hat{y}_1| \in O(h).$$

In cases where paths are perfectly straight, the Euler scheme recovers the exact solution. Later, we explore methods designed to learn a vector field that generates very straight paths, making the Euler scheme particularly efficient for such approaches.

2.2 Optimal Transport

Optimal Transport has emerged in the 1980s and 1990s by the works of Brenier [Bre91] and others. Since then several books have been written about this topic, most famously by Villani [Vil03], [Vil09]. It is a powerful tool for quantifying the dissimiliarity between probability measures. The central problem it adresses is finding the most efficient way to transport mass from one distribution to another, taking into account the underlying geometry of the space. In recent years Optimal Transport has become increasingly important in Data Science and Machine Learning. It is an important tool to compare measures and datasets, which is fundamental in tasks ranging from generative modeling to domain adaption. Domain adaption aims at applying a Machine Learning algorithm which is trained on a source domain to a different but related domain. However, a big challenge with Optimal Transport is that it can be computationally complex, especially when dealing with large amounts of data. Additionally, it suffers from a curse of dimensionality. There is a lot of ongoing research to find faster ways to calculate Optimal Transport distances, like the Wasserstein distance. A coherent review about this topic is [PC+19]. Further, many dynamics in natural science are guided by the principle of least energy. We want to move particles by the shortest possible path between two points. While this is simple for a single particle which we can just move in a straight line, the task becomes more complex when we want to move many particles from one cloud of positions to the next in the optimal way. To go even further, we might have an abstract distribution of particles at some point in time and want to move this to another distribution of the particles. Thus learning a dynamic which moves the particles optimally is very useful in our problem setting.

We start by motivating how we can get from samples to measures. Following this we want to introduce the definition of Optimal Transport and give an overview of computational challenges of Optimal Transport. Afterwards we introduce the entropic-regularized Optimal Transport and discuss its computational advantages but also challenges. Finally we will define Unbalanced Optimal Transport which relaxes the mass conservation constraint on Optimal Transport.

2.2.1 From Samples to Measures

In the previous section about Continuous Normalizing Flows we discussed how we can learn an ODE to transport one continuous measure μ to a second one ν. While this is useful in theory, in practice we will never observe a whole distribution but only samples thereof. Thus we do not have access of μ and ν. Suppose we observe $x_0^0, \dots, x_0^n \sim \mu$. We then can define an empirical measure

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_0^i}.$$

Here $\delta_x(A)$ is 1 if $x \in A$ and 0 if $x \notin A$ for all measurable sets A. For simplification let us assume that μ is a measure over $\mathbb{R}$ and let $\hat{X}^{(n)} \sim \hat{\mu}_n$ and $X \sim \mu$. We denote by $\hat{F}_n(t) = \mathbb{P}(\hat{X}^{(n)} \leq t)$ and $F(t) = \mathbb{P}(X \leq t)$ the empirical and the true distribution functions of μ. By the Glivenko-Cantelli theorem we have as $n \to \infty$ that $\hat{F}_n(t) \to F(t)$ almost surely for every value of t. This allows us to approximate the true underlying distribution with samples. For multivariate measures the convergence of the empirical process is more complicated and will be skipped here. More details can be found in [SW09].

2.2.2 Optimal Transport Cost and Wasserstein Distance

Optimal Transport aims at measuring the distance between two measures taking into account the underlying geometry of the space. This is done by measuring the cost of transporting one measure to the other. For this part we follow [Vil09] and [PC+19]. We will not review the dual formulation of Optimal Transport as we will not need it.

We denote the set of all Radon measures on the space $(\mathcal{X}, \mathcal{F})$, where $\mathcal{F}$ is a σ-algebra on $\mathcal{X}$, by $\mathcal{M}(\mathcal{X})$. A Radon measure is a Borel measure that is finite on all compact sets, outer regular on all Borel sets and inner regular on open sets. With $\mathcal{M}_+(\mathcal{X})$ we denote the positive Radon measures, while $\mathcal{P}(\mathcal{X})$ denote the Radon probability measures.

Let $\mu \in \mathcal{P}(\mathcal{X})$, $\nu \in \mathcal{P}(\mathcal{Y})$ be probability measures defined over the spaces $\mathcal{X}$ and $\mathcal{Y}$. To define Optimal Transport we will need to consider a ground cost $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, which is the cost of transporting one unit of mass from $x \in \mathcal{X}$ to $y \in \mathcal{Y}$. We then look for the best coupling of μ and ν with respect to this cost to transport one measure to the other. Formally, a coupling is defined in the following way:

Definition 2.2.1. Let $\mu \in \mathcal{P}(\mathcal{X})$, $\nu \in \mathcal{P}(\mathcal{Y})$ be two probability measures. A coupling is a probability measure $\pi \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$ that has μ and ν as its marginals. That means it is an element of the following set

$$\Pi(\mu, \nu) = \{\pi \in \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \text{ s.t. } P_{\mathcal{X}} \# \pi = \mu \text{ and } P_{\mathcal{Y}} \# \pi = \nu\}$$

where $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ are the projections from the space $X \times \mathcal{Y}$ on $\mathcal{X}$ and $\mathcal{Y}$ respectively and $\#$ denotes the pushforward operation for measures.

The set $\Pi(\mu, \nu)$ always contains the independent coupling $\pi = \mu \otimes \nu$, such that for $A \subset \mathcal{X}$ and $B \subset \mathcal{Y}$ we have $\pi(A \times B) = \mu(A)\nu(B)$.

The **Optimal Transport Cost** between μ and ν is given by

$$C(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int c(x, y) \, \mathrm{d}\pi(x, y). \tag{2.2.1}$$

However, this is not necessarily a metric. If we instead choose the function $c(x, y)$ to be a metric then $C(\mu, \nu)$ also becomes a metric on the space of probability measures.

Definition 2.2.2 (Definition 6.1 in [Vil09])**.** Let $(\mathcal{X}, d)$ be a Polish metric space and let $p \in [1, \infty)$. For any two probability measures μ, ν on $\mathcal{X}$, the Wasserstein distance of order p between μ and ν is defined by the formula

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}} d(x, y)^p \, \mathrm{d}\pi(x, y) \right)^{1/p}. \tag{2.2.2}$$

Remark 2.2.3. *W_p is a metric on $\mathcal{P}(\mathcal{X})$.*

Remark 2.2.4. *This definition can be generalized to finite measures μ and ν having the same mass.*

Under weak conditions on the spaces $\mathcal{X}$ and $\mathcal{Y}$ and the cost function c one can show that a minimizing coupling exists (see Theorem 4.1. in [Vil09]).

Discrete Formulation. In the case, where $\mu = \sum_{i=1}^{n} w_i^{\mu} \delta_{x_i}$ and $\nu = \sum_{j=1}^{m} w_j^{\nu} \delta y_j$ are discrete measures with $\sum_{i=1}^{n} w_i^{\mu} = 1$ and $\sum_{j=1}^{m} w_j^{\nu} = 1$, the cost function can be thought as a matrix $\boldsymbol{C} \in \mathbb{R}_+^{n \times m}$ with $\boldsymbol{C}_{ij} = c(x_i, y_j)$. Then the Optimal Transport cost is defined by

$$C_{\boldsymbol{C}}(\mu, \nu) := \min_{\pi \in \Pi(\mu,\nu)} \langle \boldsymbol{C}, \pi \rangle = \min_{\pi \in \Pi(\mu,\nu)} \sum_{i,j} \boldsymbol{C}_{i,j} \pi_{i,j}.$$

We interpret π as a stochastic matrix $\pi \in \mathbb{R}_+^{n \times m}$, such that $\pi_{i,j} = \pi(x_i, y_j)$. A stochastic matrix is a matrix with non-negative entries and whose rows sum to one.

Computational Challenges of Optimal Transport. In Data Science and Machine Learning one often does not have access to the true measures μ and ν but only to n samples from μ and ν. Let the empirical distributions based on those samples be denoted as $\hat{\mu}$ and $\hat{\nu}$. Unfortunately, estimating the p-Wasserstein distance with the empirical distributions scales poorly in high dimensions. The number of samples needed for a certain accuracy decreases exponentially in the

dimension of the space d, which means in the high-dimensional setting the empircial distribution becomes less and less representative of the real distribution. In mathematical terms we have

$$\mathbb{E}\left[|W_p(\mu,\nu) - W_p(\hat{\mu},\hat{\nu})|\right] = O\left(n^{-1/d}\right).$$

This result was shown by [Dud68] and [FG15]. Luckily more recent work [WB19] showed that we can find better convergence rates, namely that the convergence rate depends on an intrinsic dimension of the measures rather than the dimension of the space. If the data lies in a low dimensional manifold and we have enough samples, then we can reasonably approximate the Wasserstein distance with empirical distributions.

Besides the curse of dimensionality for approximating Optimal Transport, calculating a discrete Optimal Transport solution is very time intensive. It is solved using a linear problem. The cost of calculating the Wasserstein distance comes with a complexity of $O(n^3 \log(n))$ [PC+19]. This is very slow for many applications in Machine Learning.

Finally to calculate the discrete Optimal Transport plan we are required to store the cost matrix $\boldsymbol{C}$ and the Optimal Transport plan. Thus the memory complexity is of order $O(n^2)$.

Dynamical Optimal Transport. We can also formulate a dynamic or time-dependent version of Optimal Transport. This leads to a continuous displacement of measures. It describes the Optimal Transport of μ to ν as a curve in measure space which minimizes a total length. We will restrict ourselves here to the case of quadratic cost in Euclidean space which was introduced by McCann [BB00]. The more general case of dynamic Optimal Transport is covered in [Vil09].

In the case of quadratic cost the dynamic form of the Optimal Transport problem is given as an optimization problem:

$$W_2(\mu,\nu)^2 = \inf_{p,u}(t_1 - t_0) \int_{t_0}^{t_1} \int_{\mathbb{R}^d} \|u_t(x)\|^2 \,\mathrm{d}p_t(x)\,\mathrm{d}t, \tag{2.2.3}$$

such that $p : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}$ is a curve in the space of measures, $u : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}$ is a time-dependent vector field, u and p fullfill the continuity equation, i.e.

$$\frac{\partial p_t}{\partial t} = -\nabla \cdot (p_t u_t)$$

and $p_{t_0} = \mu$, $p_{t_1} = \nu$. Solving the discrete dynamic Optimal Transport problem is very hard as one needs to minimize a nonsmooth optimization problem under affine constraints.

2.2.3 Entropy-Regularization of Optimal Transport.

Because of the computational challenges of Optimal Transport, an entropic regularization was introduced, which is easier to solve and scales better to high dimensions. This is done by adding an entropic regularization penalty to the original cost, i.e.

$$C_c^\varepsilon(\mu,\nu) := \min_{\pi\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\,\mathrm{d}\pi(x,y) + \varepsilon \mathrm{KL}(\pi \mid \mu\otimes\nu), \tag{2.2.4}$$

where $\varepsilon \geq 0$ is the regularization coefficient and KL is the Kullback-Leibler divergence given by

$$\mathrm{KL}(\pi \mid \xi) := \int_{\mathcal{X}\times\mathcal{Y}} \log\left(\frac{\mathrm{d}\pi}{\mathrm{d}\xi}(x,y)\right)\mathrm{d}\pi(x,y) - \int_{\mathcal{X}\times\mathcal{Y}} \mathrm{d}\pi(x,y) + \int_{\mathcal{X}\times\mathcal{Y}} \mathrm{d}\xi. \tag{2.2.5}$$

The term $\frac{\mathrm{d}\pi}{\mathrm{d}\xi}$ denotes the Radon-Nikodym derivative. If π does not have a density $\frac{\mathrm{d}\pi}{\mathrm{d}\xi}$ with respect to ξ then $\mathrm{KL}(\pi \mid \xi) = \infty$.

The penalty term is 0 if π is the independent coupling. Thus we regularize by moving the optimal coupling a bit into the direction of the independent coupling.

Discrete version. The discrete version of the entropy regularized Optimal Transport is given by

$$C_{\boldsymbol{C}}^\varepsilon = \min_{\pi\in\Pi(\mu,\nu)} \langle \boldsymbol{C}, \pi\rangle + \varepsilon \mathrm{KL}(\pi \mid \mu\otimes\nu), \tag{2.2.6}$$

with

$$\mathrm{KL}(\pi \mid \xi) = \sum_{i,j} \left(\pi_{i,j}\log\left(\frac{\pi_{i,j}}{\xi_{i,j}}\right) - \pi_{i,j} + \xi_{i,j}\right).$$

The objective (2.2.6) is an $\varepsilon-$strongly convex function. Thus the entropic Optimal Transport problem has a unique optimal solution. We also have that the discrete entropic Optimal Transport problem converges to the unregularized Optimal Transport problem, as $\varepsilon \to 0$. More precisely we have $C_{\boldsymbol{C}}^\varepsilon(\mu,\nu) \to C_{\boldsymbol{C}}(\mu,\nu)$. In the continuous case one can also show Γ-convergence of the cost functionals. However, the statement is more restrictive and complicated [Cla+21].

Calculating entropic Optimal Transport. Calculating the discrete regularized Optimal Transport problem has computational advantages to the unregularized formulation. It can be solved by the Sinkhorn scheme which utilizes iterative matrix-vector products. Further, the resulting approximate distance is smooth and thus can be differentiated using automatic differentiation. The Sinkhorn algorithm utilizes the following fact:

Proposition 2.2.5 (Proposition 4.3. in [PC+19]). *Let $\boldsymbol{C}$ be the cost matrix and $\boldsymbol{K}_{i,j} := e^{-\frac{\boldsymbol{C}_{i,j}}{\varepsilon}}$. Then the solution of (2.2.6) is unique and has the solution*

$$\pi_{i,j} = \boldsymbol{u}_i \boldsymbol{K}_{i,j} \boldsymbol{v}_j$$

for two (unknown) scaling variables $\boldsymbol{u} \in \mathbb{R}_+^n$ and $\boldsymbol{v} \in \mathbb{R}_+^m$.

Together with the marginal constraints on the unique solution π one gets a matrix scaling problem which can be solved with the Sinkhorn algorithm, which is an iterative algorithm [Cut13]. Entropic Optimal Transport has a computational cost of $O(n^2)$ for fixed $\varepsilon > 0$ [ANWR17].

Dyanmic version of entropic Optimal Transport: The Schrödinger bridge problem. In this subsection we follow [Str23]. Similiar to unregularized Optimal Transport we also can formulate a dynamic version of regularized Optimal Transport. The Schrödinger Bridge problem searches for the most likely measure $\boldsymbol{D} \in \mathcal{P}(C([0,1];\mathcal{X}))$, such that its time marginals D_0 and D_1 agree with μ and ν, and with the condition that $\boldsymbol{D}$ should be as close as possible to the law of a Brownian motion. In general one can also consider other reference processes than Brownian motion.

Before we can define the Schrödinger bridge problem we need to formalize the notion of time marginals D_t for a measure $\boldsymbol{D} \in \mathcal{P}(C([0,1];\mathcal{X}))$. First let us define the time projections $X_t : C([0,1];\mathcal{X}) \to \mathcal{X}$ by $X_t(\omega) := \omega_t$ for $\omega = (\omega_t)_{t\in[0,1]} \in C([0,1];\mathcal{X})$. We then provide $\mathcal{X}$ with the σ-algebra $\mathcal{F} = \sigma(X_t; t \in [0,1])$. Finally, we can define the time marginals $D_t \in \mathcal{P}(\mathcal{X})$ by

$$D_t(A) := (X_t)\#\boldsymbol{D} = \boldsymbol{D}(X_t^{-1}(A)) \quad \text{for all } A \in \mathcal{F}.$$

The Schrödinger bridge problem is given by

$$\min_{\boldsymbol{D}: D_0=\mu, D_1=\nu} \mathrm{KL}(\boldsymbol{D} \mid \boldsymbol{R}), \tag{2.2.7}$$

where we minimize over probability measures on the Wiener space $\boldsymbol{D} \in \mathcal{P}(C([0,1];\mathcal{X}))$ and $\boldsymbol{R} \in \mathcal{P}(C([0,1];\mathcal{X}))$ is the reference measure.

We restrict ourselves to the case, where $\boldsymbol{R} = (R_t)_{t\in[0,1]}$ is the law of a Brownian motion with diffusion σ. Choosing Brownian motion as a reference process is a natural choice in many applications. It is a mathematical model that describes the random movement of particles in a fluid. Nowadays it is used in a diverse range of applications such as describing the movement of assets in finance.

Let $\boldsymbol{D}^*$ be the measure which minimizes the Schrödinger bridge problem. Now let us suppose that $\mathcal{X}$ is an Euclidean space. Then the Schrödinger bridge problem can be restored from the entropy-regularized Optimal Transport problem with $\varepsilon = 2\sigma^2$ and quadratic cost $c(x,y) = |x-y|^2$: We

denote by $\pi^*_{2\sigma^2} \in \mathcal{P}(\mathcal{X} \times \mathcal{X})$ the minimizing coupling of the static entropy-regularized Optimal Transport problem and let $(x, y) \sim \pi^*_{2\sigma^2}$. Next we draw a Brownian bridge W^{xy} with diffusion σ joining x to y, i.e. σW^{xy}. The Brownian bridge with diffusion one can be defined by $W^{xy}_t = W_t - tW_1$. Then the law of σW^{xy} is $\boldsymbol{D}^*$.

That means we can write the solution $\boldsymbol{D}^*$ as mixture of Brownian bridges weighted by the entropic Optimal Transport plan. This was shown by [Föl88]. More precisely we can write

$$\boldsymbol{D}^* = \int \boldsymbol{W}^{xy}_\sigma \, \mathrm{d}\pi^*_{2\sigma^2}(x, y), \tag{2.2.8}$$

where $\boldsymbol{W}^{xy}_\sigma$ is the law of σW^{xy}. For a detailed analysis of the Schrödinger bridge problem see [Lé13].

2.2.4 Unbalanced Optimal Transport

In this section we follow [Fat+21b] and [PC+19]. Optimal Transport has a mass constraint and thus can only compare measures with the same mass. This has the drawback that all mass of μ has to be transported to mass of ν. In the Machine Learning setting this causes problems if we have outliers or missing values in the data. Unbalanced Optimal Transport, on the other hand softens this constraint and allows for mass creation and destruction. It is thus more robust to local variations in mass such as outliers. One example where this might be important, is single-cell biology. Here the data is dependent on the experimental setup, often contains a lot of noise and missing values and is subject to mass variations due to birth and death of cells.

Let μ and ν denote the measures we want to transport between. Balanced Optimal Transport restricts the transport measure π to be a coupling between μ and ν. We soften this restriction by only penalizing marginal deviation from μ and ν using Csiszàr divergence D_φ. The Csiszàr divergence is a functional which measures how close two measures are by investigating the pointwise ratio between the two measures. Before we can introduce the Csiszàr divergence we need to define entropy functions:

Definition 2.2.6 (Definition 8.1 in [PC+19])**.** A function $\varphi : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$ is an entropy function if it is lower semicontinuous, convex, $\mathrm{dom}(\varphi) \subset [0, \infty)$ and $\mathrm{dom}(\varphi) \cap (0, \infty) \neq \emptyset$. The speed of growth of φ at ∞ is described by

$$\varphi'_\infty = \lim_{x \to \infty} \varphi(x)/x \in \mathbb{R} \cup \{\infty\}.$$

Definition 2.2.7 (Definition 8.2 in [PC+19], Csizàr divergence)**.** Let φ be an entorpy fuction. For $\mu, \nu \in \mathcal{M}(\mathcal{X})$, let $\frac{\mathrm{d}\mu}{\mathrm{d}\nu}\nu + \mu^\perp$ be the Lebesgue decomposition of μ with respect to ν. The Lebesgue decomposition is the unique decomposition $\mu = \mu^s + \mu^\perp$ such that μ^s is absolutely continuous with respect to ν and $\mu^\perp$ and ν are singular. The divergence D_φ is defined by

$$D_\varphi(\mu \mid \nu) := \int_{\mathcal{X}} \varphi\left(\frac{\mathrm{d}\mu}{\mathrm{d}\nu}\right) \mathrm{d}\nu + \varphi'_\infty \mu^\perp(\mathcal{X}) \tag{2.2.9}$$

if μ and ν are nonnegative and ∞ otherwise.

Kullback-Leibler and Total Variation divergence are special cases of Csizàr divergences.

With this notion we can define the unbalanced Optimal Transport problem:

$$\mathrm{OT}^{\tau}_{\varphi,c}(\mu,\nu) = \min_{\pi\in\mathcal{M}_+(\mathcal{X}^2)} \int_{\mathcal{X}\times\mathcal{X}} c(x,y)\,\mathrm{d}\pi(x,y) + \tau\left(D_\varphi(\pi_0 \mid \mu) + D_\varphi(\pi_1 \mid \nu)\right), \tag{2.2.10}$$

where π_1 and π_2 are the marginals of π and τ is the marginal penalization. The marginals do not have to be equal to μ and ν anymore. We can see that the unbalanced Optimal Transport problem does not require to match the marginals exactly. Instead, τ determines how strongly mass variations are penalized.

An algorithm to solve this problem was proposed in [Cha+21]. The authors determined the computational complexity only computationally, which averaged to $O(n^{3.27})$. However, they only tested it on 2 datasets in 10 dimensions. In our experiments the algorithm performed significantly slower than balanced Optimal Transport even if we used a small sample size.

As with the balanced Optimal Transport problem we can define an entropic regularization of the unbalanced Optimal Transport problem. The objective is then defined as

$$\mathrm{OT}^{\tau,\varepsilon}_{\varphi,c}(\mu,\nu) = \min_{\pi\in\mathcal{M}_+(\mathcal{X}^2)} \int_{\mathcal{X}\times\mathcal{X}} c(x,y)\,\mathrm{d}\pi(x,y) + \varepsilon\mathrm{KL}(\pi \mid \mu\otimes\nu) \tag{2.2.11}$$

$$+ \tau\left(D_\varphi(\pi_0 \mid \mu) + D_\varphi(\pi_1 \mid \nu)\right). \tag{2.2.12}$$

This can be computed via a generalized Sinkhorn algorithm [Chi+18], [Séj+19]. For $D_\varphi = \mathrm{KL}$ the scheme converges with a rate of $O(n^2/\varepsilon)$.

The role of τ. When using Unbalanced Optimal Transport the parameter τ plays a crucial role. If we want to use unbalanced Optimal Transport in Machine Learning tasks we need to fine-tune and adapt τ to the dataset and the cost function c. In general it can be interpreted as the radius of transportation, meaning it determines how far we transport mass or whether or not we let mass to be constructed and destructed. Too large values of τ enforce mass conservation, while too small values can lead to very small marginal masses of the optimal map. Thus, we do not transport a lot of mass, which is also undesirable. In practice we need to tune the hyperparameter τ.

Chapter 3

Flow Matching

We introduced Continuous Normalizing Flows in Section 2.1. Flow Matching is a simulation-free approach for training CNFs, which means that we do not need to simulate an ODE during training. As an alternative, we want to learn the vector field $u_t(x)$ from (2.1.1) by regression. However, we do not have access to the true $u_t(x)$. Instead we will only learn a conditional version of u_t, which we will specify soon. Flow Matching allows us to train CNFs on a much faster scale. It was introduced in 2023 by [Lip+22] for transporting Gaussian noise to a data distribution. This approach was later generalized by [Ton+24] to handle transport between any two arbitrary distributions. Around the same time, other works proposed similar objectives for simulation-free training of CNFs [LGL23], [ABVE23]. In the next chapter, we will discuss Action Matching [Nek+23b], which also provides simulation-free training for CNFs. However, it derives an implicit objective when u_t is assumed to be a gradient field. To learn u_t directly, Action Matching requires additional assumptions.

We follow [Lip+22] and [Ton+24], while focusing directly on arbitrary measures. Let us consider a pair of probability distributions over $\mathbb{R}^d$ with densities μ and ν. Our goal is to find a function which transports μ to ν. More precisely, if X is a random variable which is distributed with density μ then we want to learn a function $f : \mathbb{R}^d \to \mathbb{R}^d$ such that $f(X)$ is distributed according to the density ν.

3.1 Conditional Flow Matching: General Approach

We use the same framework as in Section 2.1. That means we suppose the dynamics that generates the data is deterministic and follows (2.1.1), i.e.

$$\mathrm{d}x_t = u_t(x_t)\,\mathrm{d}t, \tag{3.1.1}$$

with $x_0 \sim \mu$ and $x_1 \sim \nu$. Let $p = (p_t)_{t\in[0,1]}$ denote the probability path, which describes the density of $(x_t)_{t\in[0,1]}$ and which is generated by u, i.e. p and u satisfy the continuity equation (2.1.3). Our goal is to learn u. Afterwards we are able to transport samples from μ to samples from ν by solving an ODE. However, we do not want to solve the ODE during training. A very simple way to learn u would be to use regression. Using this idea we define the Flow Matching objective as

$$\mathcal{L}_{FM}(\theta) := \mathbb{E}_{t\sim U([0,1]),x\sim p_t}\|v_\theta(t,x) - u_t(x)\|^2, \tag{3.1.2}$$

where v_θ is learned via a neural network and θ denotes the learnable parameter. $U([0,1])$ denotes the uniform distribution over the interval $[0,1]$. However, this loss is intractable as we do not have access to the time marginals p_t and u_t. There are many choices for p which transports μ to ν and we normally do not have access to a closed form representation of u. To simplify the problem we condition u and p on samples and solve a conditional objective. Specifically, we take a sample $x_0 \sim \mu$ and a sample $x_1 \sim \nu$ and then interpolate between those two data points. By appropriately mixing the conditional marginals u_t and p_t, we can recover the unconditional marginals p_t and u_t. Furthermore, we demonstrate that matching the conditional u_t is equivalent to the Flow Matching objective (3.1.2).

Our aim is to generate samples from the target distribution ν given samples from the source distribution ν. Since there are several valid choices for p and u given the information, it is impossible to match the 'true' functions. Nevertheless, our choices are guided by physical principles such as the principle of least action, and we will present various options. The specific choice will depend on the application context.

Using Conditional Probability Paths

We assume that we can write write p as a mixture of simpler probability paths. Let $x_0 \sim \mu$ and $x_1 \sim \nu$. Then we define $(p_t(x \mid x_0, x_1))_{t\in[0,1]}$ as a conditional probability path which is concentrated around x_0 at time 0 and around x_1 at time 1. To make this more rigorous we assume $p_0(x \mid x_0, x_1) = \mathcal{N}(x \mid x_0, \sigma^2 I)$ and $p_1(x \mid x_0, x_1) = \mathcal{N}(x \mid x_1, \sigma^2 I)$ for some small $\sigma > 0$. We then recover the unconditional $p = (p_t)_{t\in[0,1]}$ by mixing the conditional probability paths with the density $q(x_0, x_1)$, which we choose later. We get

$$p_t(x) = \int p_t(x \mid x_0, x_1) q(x_0, x_1)\, \mathrm{d}(x_0, x_1). \tag{3.1.3}$$

At the endpoints $t = 0$ and $t = 1$ the conditional path is concentrated around x_0 or x_1. So p_0 and p_1 closely approximate the marginals of q:

$$\begin{aligned} p_0(x) &= \int p_0(x \mid x_0, x_1) q(x_0, x_1)\, \mathrm{d}(x_0, x_1) \approx \int q(x, x_1)\, \mathrm{d}x_1 \\ p_1(x) &= \int p_1(x \mid x_0, x_1) q(x_0, x_1)\, \mathrm{d}(x_0, x_1) \approx \int q(x_0, x)\, \mathrm{d}x_0. \end{aligned}$$

Since we want p to interpolate between μ and ν, we need $p_0(x) \approx \mu$ and $p_1(x) \approx \nu$. Therefore, it is natural to choose $q(x_0, x_1)$ as a coupling of μ and ν. Additionally, we choose $(p_t(x \mid x_0, x_1))_t$ such that we have access to a closed form conditional vector field $(u_t(x \mid x_0, x_1))_{t\in[0,1]}$ which generates $(p_t(x \mid x_0, x_1))_{t\in[0,1]}$ according to the continuity equation (2.1.3). By proberly mixing the conditional vector fields, we can reconstruct an unconditional vector field u that generates p:

Theorem 3.1.1 (Theorem 3.1 in [Ton+24]). *The vector field*

$$u_t(x) = \int \frac{u_t(x \mid x_0, x_1)p_t(x \mid x_0, x_1)}{p_t(x)} q(x_0, x_1)\,\mathrm{d}(x_0, x_1) \tag{3.1.4}$$

generates the probability path $(p_t)_{t\in[0,1]}$ *defined by* (3.1.3) *from the initial condition* $p_0(x)$.

Even though, we have a method to calculate u, it might not be practical to use u directly. This is particularly true in higher dimensions, where the integrals involved in defining the probability path (3.1.3) and the vector field (3.1.4) may still be intractable. To address this, we focus on working directly with the conditional objectives and regress the conditional vector fields We define the conditional learning objective *Conditional Flow Matching* (CFM) as follows:

$$\mathcal{L}_{CFM}(\theta) := \mathbb{E}_{t\sim\mathcal{U}([0,1]),z\sim q,x\sim p_t(\cdot|z)}\|v_\theta(t, x) - u_t(x \mid z)\|^2, \tag{3.1.5}$$

where $z = (x_0, x_1)$ and the subscript of the expected value denotes that we take the expectation over t sampled uniformly from $[0, 1]$, z sampled from q and finally x sampled from $p_t(\cdot \mid z)$. Here $v_\theta : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ is parameterized by a neural network with weights θ. For the CFM objective it is only necessary that we can efficiently sample from the marginals $p_t(x \mid x_0, x_1)$ and compute $u_t(x \mid x_0, x_1)$. The algorithmic description can be found in Algorithm 1. The CFM objective is equivalent to the unconditional Flow Matching objective (3.1.2) as the following theorem shows:

Theorem 3.1.2 (Theorem 3.2 in [Ton+24]). *Let* $q, p_t(\cdot \mid z) \in L^2(\mathbb{R}^d)$ *and* u_t, $v_\theta(t, \cdot)$, $\nabla_\theta v_\theta(t, \cdot)$ *be bounded. If* $p_t(x) > 0$ *for all* $x \in \mathbb{R}^d$ *and* $t \in [0, 1]$, *then*

$$\nabla_\theta\mathcal{L}_{FM}(\theta) = \nabla_\theta\mathcal{L}_{CFM}(\theta). \tag{3.1.6}$$

Proof. We have

$$\begin{aligned}
&\nabla_\theta\mathbb{E}_{t\sim\mathcal{U}([0,1]),x\sim p_t}\|v_\theta(t, x) - u_t(x)\|^2 \\
&= \nabla_\theta\mathbb{E}_{t\sim\mathcal{U}([0,1]),x\sim p_t}\left[\|v_\theta(t, x)\|^2 - 2\langle v_\theta(t, x), u_t(x)\rangle + \|u_t(x)\|^2\right] \\
&= \nabla_\theta\mathbb{E}_{t\sim\mathcal{U}([0,1]),x\sim p_t}\left[\|v_\theta(t, x)\|^2 - 2\langle v_\theta(t, x), u_t(x)\rangle\right].
\end{aligned}$$

The first term we can be rewritten as

$$\begin{aligned}
\mathbb{E}_{t\sim\mathcal{U}([0,1]),x\sim p_t}\|v_\theta(t, x)\|^2 &= \int_0^1 \int_{\mathbb{R}^d} \|v_\theta(t, x)\|^2 p_t(x)\,\mathrm{d}x\,\mathrm{d}t \\
&= \int_0^1 \int_{\mathbb{R}^d} \int_{\mathbb{R}^d\times\mathbb{R}^d} \|v_\theta(t, x)\|^2 p_t(x \mid z)q(z)\,\mathrm{d}z\,\mathrm{d}x\,\mathrm{d}t
\end{aligned}$$

$$= \mathbb{E}_{t\sim\mathcal{U}([0,1]),z\sim q,x\sim p_t(\cdot|z)}\|v_\theta(t,x)\|^2.$$

The second term can be rewritten as

$$\begin{aligned}
&\mathbb{E}_{t\sim\mathcal{U}([0,1]),x\sim p_t}\langle v_\theta(t,x), u_t(x)\rangle \\
&= \int_0^1 \int_{\mathbb{R}^d} \left\langle v_\theta(t,x), \frac{\int_{\mathbb{R}^d\times\mathbb{R}^d} u_t(x\mid z)p_t(x\mid z)q(z)\,\mathrm{d}z}{p_t(x)} \right\rangle p_t(x)\,\mathrm{d}x\,\mathrm{d}t \\
&= \int_0^1 \int_{\mathbb{R}^d} \left\langle v_\theta(t,x), \int_{\mathbb{R}^d\times\mathbb{R}^d} u_t(x\mid z)p_t(x\mid z)q(z)\,\mathrm{d}z \right\rangle \mathrm{d}x\,\mathrm{d}t \\
&= \int_0^1 \int_{\mathbb{R}^d} \int_{\mathbb{R}^d\times\mathbb{R}^d} \langle v_\theta(t,x), u_t(x\mid z)\rangle p_t(x\mid z)q(z)\,\mathrm{d}z\,\mathrm{d}x\,\mathrm{d}t \\
&= \mathbb{E}_{t\sim\mathcal{U}([0,1]),z\sim q,x\sim p_t(\cdot|z)}\langle v_\theta(t,x), u_t(x\mid z)\rangle
\end{aligned}$$

Further, we get

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{t\sim\mathcal{U}([0,1]),z\sim q,x\sim p_t(\cdot|z)}\|v_\theta(t,x) - u_t(x\mid z)\|^2 \\
&= \nabla_\theta \mathbb{E}_{t\sim\mathcal{U}([0,1]),z\sim q,x\sim p_t(\cdot|z)} \left[\|v_\theta(t,x)\|^2 - 2\langle v_\theta(t,x), u_t(x\mid z)\rangle + \|u_t(x\mid z)\|^2\right] \\
&= \nabla_\theta \mathbb{E}_{t\sim\mathcal{U}([0,1]),z\sim q,x\sim p_t(\cdot|z)} \left[\|v_\theta(t,x)\|^2 - 2\langle v_\theta(t,x), u_t(x\mid z)\rangle\right].
\end{aligned}$$

This shows that (3.1.6) is fulfilled. □

Algorithm 1 Conditional Flow Matching

Input: Coupling q of the initial distribution μ and the target distribution ν, initial network $v_\theta(t,x)$, number of training iterations, batch size B
Output: trained model $v_\theta(t,x)$
  for training iterations **do**
    $(\boldsymbol{x_0}, \boldsymbol{x_1}) \leftarrow \{(x_0^i, x_1^i)\}_{i=0}^B$ with $(x_0^i, x_1^i) \sim q$
    $\boldsymbol{t} \leftarrow \{t_i\}_{i=0}^B$ with $t_i \sim U[0,1]$
    $\boldsymbol{x_t} \leftarrow \{x_{t_i}^i\}_{i=1}^B$ with $x_{t_i}^i \sim p_{t_i}(\cdot \mid x_0^i, x_1^i)$
    $\mathcal{L}_{CFM}(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^B \|v_\theta(t_i, x_{t_i}^i) - u_{t_i}(x_{t_i}^i \mid x_0^i, x_1^i)\|^2$
    $\theta \leftarrow \texttt{Update}(\theta, \nabla_\theta \mathcal{L}_{CFM}(\theta))$
  end for

3.2 Different Forms of Conditional Flow Matching

Although we established the theoretical framework for Conditional Flow Matching, we did not yet discuss how to select $(p_t(x \mid x_0, x_1))_{t\in[0,1]}$ in practice. While numerous options exist, we opt for the family of Gaussian distributions. The main reason for this is that we have a closed form for $(u_t(x \mid x_0, x_1))_{t\in[0,1]}$ which generates $(p_t(x \mid x_0, x_1))_{t\in[0,1]}$. The closed form of the marginal

$u_t(\cdot \mid x_0, x_1)$ is given in (2.1.6). A second reason is that the solution of the Dynamic Optimal Transport problem from $\mathcal{N}(x \mid x_0, \sigma^2 I)$ to $\mathcal{N}(x \mid x_1, \sigma^2 I)$ is a Gaussian probability path.

We choose

$$p_t(x \mid x_0, x_1) = \mathcal{N}(x \mid \mu_t, \sigma_t^2 I). \tag{3.2.1}$$

where $\mu : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is the time-dependent mean while $\sigma : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d_{>0}$ is the time dependent standard variation of the Gaussian distribution. While we will explore different choices for μ_t and σ_t, both should be differentiable with respect to t.

As previously mentioned we want $p_0(x \mid x_0, x_1) = \mathcal{N}(x \mid x_0, \sigma^2 I)$ and $p_0(x \mid x_1, x_1) = \mathcal{N}(x \mid x_1, \sigma^2 I)$. Therefore, μ_t should interpolate between x_0 and x_1, while σ_0 and σ_1 should have small values.

Remark 3.2.1. *By appropriately selecting μ_t and σ_t, we can recover the conditional probability paths used in Score Matching with SDEs [Son+21]. The conditional vector fields of CFM are the same as those used in the deterministic probability flow in [Son+21].*

3.2.1 Independent Conditional Flow Matching

A natural choice for μ_t is to just linearly interpolate between x_0 and x_1

$$p_t(x \mid x_0, x_1) = \mathcal{N}(x \mid tx_1 + (1-t)x_0, \sigma^2), \tag{3.2.2}$$

with σ chosen sufficiently small. By (2.1.6) $(p_t(x \mid x_0, x_1))_{t \in [0,1]}$ is generated by

$$u_t(x \mid x_0, x_1) = x_1 - x_0. \tag{3.2.3}$$

In fact one can show that the pair given by (3.2.2) and (3.2.3) transports $\mathcal{N}(x_0, \sigma^2)$ to $\mathcal{N}(x_1, \sigma^2)$ optimally in the sense of dynamical Optimal Transport [CGP16]. The name Independent Conditional Flow Matching (I-CFM) is due to the fact that we choose $q(x_0, x_1)$ to be the independent coupling of μ and ν, i.e. $q(A, B) = \mu(A)\nu(B)$ for all measurable sets $A, B \subseteq \mathcal{X}$. As it is easy to sample from $q(x_0, x_1)$, $p_t(x \mid x_0, x_1)$ and to compute $u_t(x \mid x_0, x_1)$, we can calculate the gradient descent on $\mathcal{L}_{\text{CFM}}$ efficiently. We further achieve that the unconditional probability path interpolates between μ and ν up to noise of the scale σ:

Proposition 3.2.2 (Proposition 3.3. in [Ton+24])**.** *The unconditional probability path p corresponding to q, $p_t(x \mid x_0, x_1)$ and $u_t(x \mid x_0, x_1)$ in the sense of* (3.1.3) *has boundary conditions $p_0 = \mu * \mathcal{N}(x \mid 0, \sigma^2)$ and $p_1 = \nu * \mathcal{N}(x \mid 0, \sigma^2)$, where $*$ is the convolution operator.*

As we let $\sigma \to 0$ we recover the boundary conditions μ and ν.

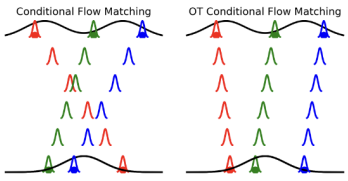

Figure 3.1: This image is taken from [Ton+24]. On the left we see the conditional flows from I-CFM and on the right from OT-CFM.

3.2.2 Optimal Transport Conditional Flow Matching

While I-CFM recovers dynamical Optimal Transport from $p_0(x \mid x_0, x_1)$ to $p_1(x \mid x_0, x_1)$, it does not recover dynamical Optimal Transport between μ and ν. Optimal Transport Conditional Flow Matching (OT-CFM), on the other hand, approximately recovers dynamical Optimal Transport (see Proposition 3.2.3).

We choose $p_t(x \mid x_0, x_1)$ and $u_t(x \mid x_0, x_1)$ to be the same as in I-CFM:

$$p_t(x \mid x_0, x_1) = \mathcal{N}(x \mid tx_1 + (1-t)x_0, \sigma^2)$$
$$u_t(x \mid x_0, x_1) = x_1 - x_0.$$

However, instead of the independent coupling we now choose $q(x_0, x_1) = \pi(x_0, x_1)$ to be the coupling which minimizes the 2-Wasserstein distance (2.2.2) between μ and ν.

The following Proposition shows that this restores approximate dynamical Optimal Transport:

Proposition 3.2.3 (Proposition 3.4. in [Ton+24])**.** *The results of Proposition 3.2.2 hold for q being the coupling which minimizes 2-Wasserstein distance. Furthermore, assuming μ and ν to be bounded densities and $\mu(x)\,\mathrm{d}x$, $\nu(x)\,\mathrm{d}x$ be compactly supported measures, we get that for $\sigma^2 \to 0$ the probability path p and the vector field u minimizes the dynamic Optimal Transport problem* (2.2.3) *between μ and ν.*

Thus, OT-CFM provides a fairly simple objective for solving the dynamic Optimal Transport problem given the static Optimal Transport coupling of (2.2.2) between μ and ν.

As OT-CFM learns the Optimal Transport path it gives us very straight paths. This is beneficial if we want to simulate the particles $(x_t)_{t\in[0,1]}$ by solving the ODE (2.1.1). For perfectly straight paths we would only need one step of the numerical Euler scheme to solve the ODE exactly.

Comparison of I-CFM and OT-CFM OT-CFM approximates dynamical Optimal Transport between μ and ν while I-CFM only achieves Optimal Transport of the conditional distribu-

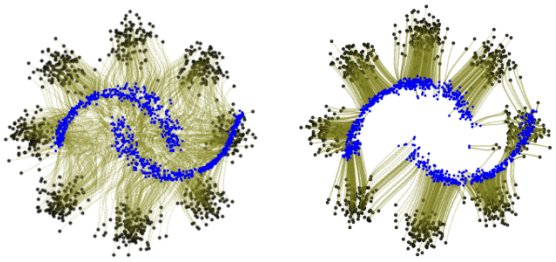

Figure 3.2: This image is taken from [Ton+24]. We see the learned sample paths (green) from moons (blue) to 8gaussians (black) using I-CFM (left) and OT-CFM (right)

.

tion path. This results in very straight transportation paths of samples for OT-CFM as we can see in Figure 3.1 and Figure 3.2 while the sample paths of I-CFM are more curved.

In Chapter 6, we demonstrate that OT-CFM performs exceptionally well in scenarios with numerous time points. This is because errors caused by suboptimal matching of data points tend to accumulate over time. To illustrate this, imagine a point cloud that splits into two distinct point clouds which then move independently in space. Ideally, after the split, a point should remain within its respective point cloud. However, I-CFM may couple points from different point clouds, causing the model to learn trajectories where points switch between clouds. Nevertheless, as we discuss in Section 3.3, even OT-CFM may not be optimal for this task in practice.

I-CFM requires less computational effort since it only calculates the independent coupling. In contrast, OT-CFM demands more computational effort, as it needs to approximate the Optimal Transport coupling for the 2-Wasserstein distance between μ and ν.

3.2.3 Schrödinger Bridge Conditional Flow Matching

Schrödinger Bridge Conditional Flow Matching (SB-CFM) is an entropic variant of OT-CFM. It can be used to learn a Schrödinger bridge between the source and target distributions using a Brownian motion as the reference process. We introduced the concept of the Schrödinger bridge in Section 2.2.3. For SB-CFM we define the conditional probability path as a Brownian bridge with diffusion scale σ between x_0 and x_1. That is

$$p_t(x \mid x_0, x_1) = \mathcal{N}(x \mid (1-t)x_0 + tx_1, t(1-t)\sigma^2). \tag{3.2.4}$$

By (2.1.6) this is generated by the conditional vector field

$$u_t(x \mid x_0, x_1) = \frac{1-2t}{2t(1-t)}(x - (tx_1 + (1-t)x_0)) + (x_1 - x_0). \tag{3.2.5}$$

Further, we choose the coupling $q = \pi_{2\sigma^2}$ to be the solution of entropic Optimal Transport problem with regularization parameter $\varepsilon = 2\sigma^2 > 0$ and cost $c(x, y) = \|x - y\|^2$. We saw in equation (2.2.8) that our choices generate the same marginal probabilities $p_t(x)\,\mathrm{d}x$ as the solution of the Schrödinger bridge problem (2.2.7) where the reference process is a Brownian motion with diffusion σ. This statement is also proved in Proposition 3.5 of [Ton+24].

3.2.4 Using General Time Intervals

To generalize Flow Matching to intervals $[t_0, t_1]$ we have to do small adaptions to the probability path. For I-CFM and OT-CFM we choose

$$p_t(x \mid x_{t_0}, x_{t_1}) = \mathcal{N}\left(x \,\middle|\, \frac{t - t_0}{t_1 - t_0} x_{t_1} + \frac{t_1 - t}{t_1 - t_0} x_{t_0}, \sigma^2\right),$$

which results in the conditional vector field

$$u_t(x \mid x_{t_0}, x_{t_1}) = \frac{x_{t_1} - x_{t_0}}{t_1 - t_0}.$$

For SB-CFM we get the conditional probability path

$$p_t(x \mid x_{t_0}, x_{t_1}) = \mathcal{N}\left(x \,\middle|\, \frac{t - t_0}{t_1 - t_0} x_{t_1} + \frac{t_1 - t}{t_1 - t_0} x_{t_0}, \sigma^2 (t - t_0)(t_1 - t)\right).$$

This is generated by the conditional vector field

$$u_t(x \mid x_{t_0}, x_{t_1}) = \frac{t_0 + t_1 - 2t}{2(t - t_0)(t_1 - t)}(x - \mu_t) + \frac{x_{t_1} - x_{t_0}}{t_1 - t_0}$$

with $\mu_t = \frac{t - t_0}{t_1 - t_0} x_{t_1} + \frac{t_1 - t}{t_1 - t_0} x_{t_0}$.

3.3 Calculating Optimal Transport Couplings Using Minibatch Optimal Transport

In Section 2.2 we saw that calculating the Optimal Transport coupling between large datasets can become computationally very complex and infeasible. Suppose we have samples from μ and ν and let $\hat{\mu}$ and $\hat{\nu}$ denote the empirical distributions induced by these samples. Then if we have many samples it may be hard to calculate the true Optimal Transport coupling between $\hat{\mu}$ and $\hat{\nu}$ directly. Even entropic Optimal Transport still has a complexity of $O(n^2)$, where n is the number of samples. This is too costly for large datasets. To speed up the computations we compute Optimal Transport on minibatches: Instead of computing the Optimal Transport coupling for the entire dataset, we calculate it on smaller batches of samples. This process is repeated multiple

times, and the results are averaged to approximate the overall Optimal Transport coupling. This strategy follows the method introduced in [GPC18] and further formalized in [Fat+21a]. Minibatch Optimal Transport gives us a good estimator and allows us to calculate gradients. However, minibatch Optimal Transport is not a metric. Let $X = \{x_0, \dots, x_n\}$ be the n samples at time t_0 such that $x_i \sim \mu$ and $Y = \{y_0, \dots, y_m\}$ the m samples at time t_1 such that $x_i \sim \nu$. Let $\mathcal{P}_l(X)$ (resp. $\mathcal{P}_l(Y)$) be the set of all subsets of X (resp. Y) with cardinality l. For each pair of $A = \{a_1, \dots, a_l\} \in \mathcal{P}_l(X)$ and $B = \{b_1, \dots, b_l\} \in \mathcal{P}_l(Y)$ we denote by $\pi_{A,B}$ the optimal coupling between the empirical distributions induced by A and B. As an example the empirical distribution for A is given by $\hat{p}_A = \frac{1}{l}\sum_{i=1}^{l} \delta_{a_i}$. Then we can define the averaged minibatch transport matrix for batch size l:

$$\tilde{\pi}_l(X, Y) = \binom{n}{l}^{-1} \binom{m}{l}^{-1} \sum_{A \in \mathcal{P}_l(X)} \sum_{B \in \mathcal{P}_l(Y)} \pi_{A,B}. \tag{3.3.1}$$

In practice it is too expensive to calculate the average of all batches of size l. Thus, we use a subsampled version

$$\tilde{\pi}_l^{(k)}(X, Y) := \frac{1}{k} \sum_{(A,B) \in D_k} \pi_{A,B}, \tag{3.3.2}$$

where D_k is a set of cardinality k whose elements are drawn uniformly from $\Gamma := \mathcal{P}_l(X) \times \mathcal{P}_l(Y)$. Depending on which Optimal Transport problem we want to approximate we can take $\pi_{A,B}$ to be either the exact Optimal Transport coupling from the p-Wasserstein distance (2.2.2) or the entropy regularized distance:

$$W^{\varepsilon}(\mu, \nu) = \min_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} \|x - y\|_2^p \, \mathrm{d}\pi(x, y) + \varepsilon \mathrm{KL}(\pi \mid \mu \otimes \nu).$$

In [Fat+20] it was shown for both Optimal Transport variants that the subsampled coupling $\tilde{\pi}_l^{(k)}$ converges in probability exponentionally fast to $\tilde{\pi}_l$ as $k \to \infty$. We set $k = 1$ which showed good experimental performance in training Machine Learning algorithms [GPC18], [Fat+21a], [Ton+24]. This is because Machine Learning algorithms train over many iterations. Therefore, the error of a single minibatch coupling does not matter much. Still, minibatch Optimal Transport tends to connect suboptimal samples. Thus, can be thought of a kind of regularization of the exact Optimal Transport. As the resulting coupling matrix $\tilde{\pi}_l^{(k)}$ is notably less sparse, this can lead to undesirable couplings and plans that can be close to a uniform distribution. Figure 3.3 shows that this problem might occur with clustered data. In [Fat+21b] it was proposed to use minibatch unbalanced Optimal Transport couplings to make the averaged transport coupling more robust. We introduced unbalanced Optimal Transport in Section 2.2.4. Unbalanced Optimal Transport is more robust to outliers and does not need to find a match for all samples. We want to pick up this idea here and test if this can bring advantages to Flow Matching. The definitions of the optimal minibatch plan $\tilde{\pi}_l$ (3.3.1) and the sampled optimal minibatch plan $\tilde{\pi}_l^{(k)}$ (3.3.2) stay the same. However, we choose the measure $\pi_{A,B}$ to be the optimal measure of the unbalanced Optimal Transport problem (2.2.10) or the regularized unbalanced Optimal Transport problem (2.2.11). In Figure 3.4 we see how balanced and unbalanced minibatch Optimal

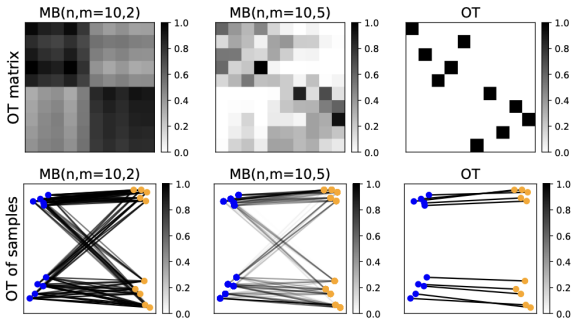

Figure 3.3: This image is taken from [Fat+21b]. We see Optimal Transport matrices, normalized by maximum value, between two dimensional distributions with $n = 10$ samples. The first row shows the minibatch Optimal Transport plans π for different minibatch sizes m. The second row shows the mass transport between samples. The rightmost picture shows the full Optimal Transport coupling.

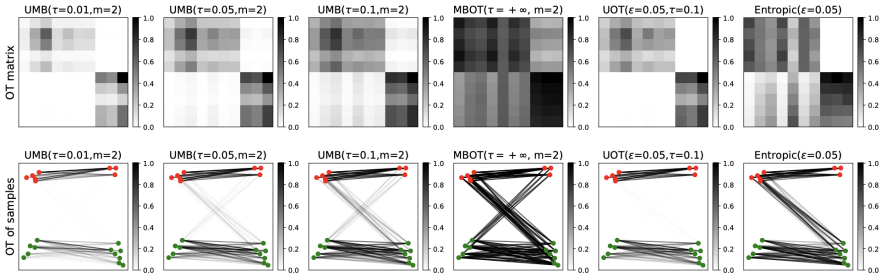

Figure 3.4: This image is taken from [Fat+21b]. We see Optimal Transport matrices, normalized by maximum value, between two dimensional distributions with $n = 10$ samples. The first row compares the unbalanced and balanced minibatch Optimal Transport plans π for different minibatch sizes m. The second row shows the mass transport between samples.

Transport with different parameters learn to transport clustered data. We want that samples are not likely to be transported in between clusters, i.e. diagonally. Unbalanced minibatch Optimal Transport (UMB in the Figure) shows superior results to the balanced minibatch version (MBOT). The learned coupling by MBOT is close to a uniform distribution, which means a lot of information is lost by the minibatch approximation.

Influence of the Minibatch Strategy on OT-CFM.

The first step of OT-CFM (Algorithm 1) requires us to sample a batch from the Optimal Transport coupling. In order to approximate this by minibatch Optimal Transport we use Algorithm 2. The function CalculateOTCoupling calculates either the Optimal Transport, regularized Optimal

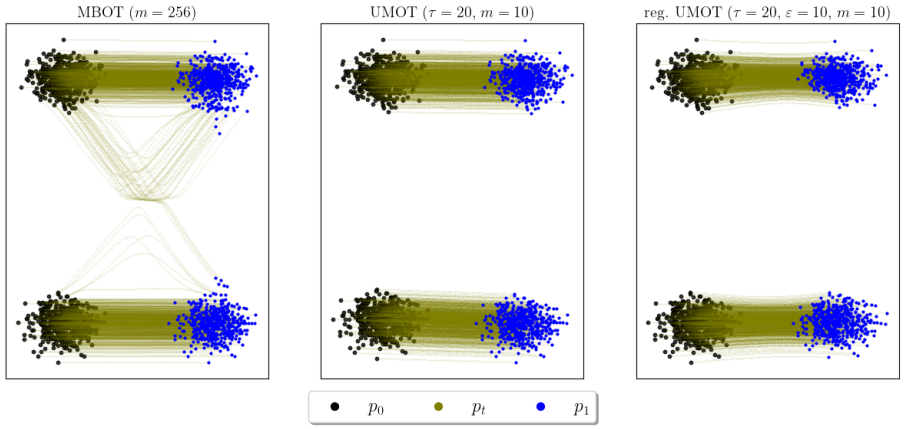

Figure 3.5: We compare the probability paths learned by OT-CFM using either minibatch Optimal Transport couplings (MBOT) or unbalanced minibatch Optimal Transport coupling (UMOT). For MBOT we use a batch size of 256 while for UMOT we use a batch size of only 10.

Transport or (regularized) unbalanced Optimal Transport map.

Algorithm 2 Calculate the Optimal Transport coupling q

Input: Initial distribution μ, target distribution ν, batch size B
Output: training batch $\{(x_0^i, x_1^i)\}_{i=1}^B$
$\boldsymbol{X} \leftarrow \{x_0^i\}_{i=1}^B$ with $x_0^i \sim \mu$
$\boldsymbol{Y} \leftarrow \{x_1^i\}_{i=1}^B$ with $x_1^i \sim \nu$
$\pi_{\boldsymbol{X},\boldsymbol{Y}} \leftarrow \text{CalculateOTCoupling}(\boldsymbol{X}, \boldsymbol{Y})$
$\boldsymbol{x} \leftarrow \{(x_0^i, x_1^i)\}_{i=1}^B$ with $(x_0^i, x_1^i) \sim \pi_{\boldsymbol{X},\boldsymbol{Y}}$
return $\boldsymbol{x}$

Interpolation between clustered data. We aim to investigate the influence of the approximation of the Optimal Transport coupling on the performance of Flow Matching. For this we interpolate between similiar distributions as used in Figure 3.3 and Figure 3.4. The objective is to compare the learned probability paths of OT-CFM for different methods to approximate the measure $q(x_0, x_1)\,\mathrm{d}(x_0, x_1)$. Figure 3.5 showcases the resulting learned probability paths p using minibatch Optimal Transport (MBOT), unbalanced minibatch Optimal Transport (UMOT) and regularized unbalanced minibatch Optimal Transport (reg. UMOT). Ideally, OT-CFM would learn perfectly straight paths connecting the samples drawn from p_0 and p_1.

Despite using a relatively large batch size ($m = 256$) for MBOT, it still generates suboptimal paths with significant curvature. This can be attributed to the minibatch approximation during training. All samples within a batch need to be matched, and it is possible that at time step 0, there are more samples from the upper cluster, while at time step 1, there are more from the

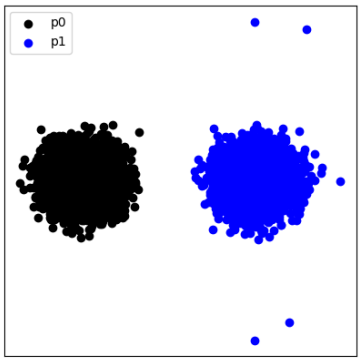

Figure 3.6: The data consists of 5000 initial and target samples, sampled from gaussian distributions. The black scatters represent the initial distribution while the blue ones represent the target distribution. The target data contains 5 outliers

lower cluster. This forces diagonal matching, leading to suboptimal paths.

In contrast, UMOT utilizes a considerably smaller batch size of only 10. Remarkably, it learns very straight paths. The regularized version's paths exhibit a slight inward bend but remain largely straight. When the data exhibits a clustered structure that persists over time, MBOT might result in suboptimal paths. In this scenario, UMOT offers a clear advantage.

Robustness to Outliers. One of the main reasons why unbalanced Optimal Transport is used in Machine Learning, is its promise of robustness to outliers and missing data [SPV23]. Here we want to test if we can also make Flow Matching more robust to outliers. For this we interpolate between two Gaussian distributions, where the target distribution contains some outliers, as can be seen in Figure 3.6. The performance of OT-CFM with MBOT, UMOT or reg. UMOT approximation is compared in Figure 3.7. We trained the methods for 10000 iterations. The figure shows where samples from the black distribution are transported to by the trained models. The probability path p is drawn in green. Ideally, the blue predictions should resemble the blue point cloud in Figure 3.6. We see that OT-CFM is impacted by the outliers and learns some of them. In contrast, using either the regularized or unregularized version of unbalanced minibatch Optimal Transport improves Flow Matching's robustness to outliers. It remains a modeling choice whether to include the outliers in the learned distribution or to focus on a distribution that disregards them.

In Chapter 7 we compare the different minibatch approximations on real single cell data.

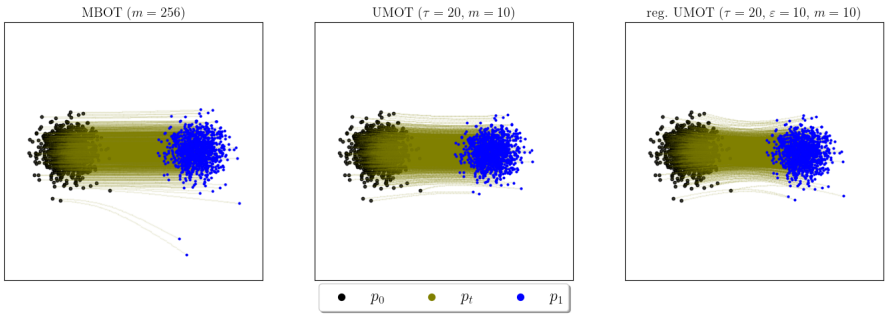

Figure 3.7: We compare the probability paths learned by OT-CFM using either minibatch Optimal Transport couplings or unbalanced minibatch Optimal Transport coupling. For MBOT and regularized UMB we use a batch size of 256 while for unregularized UMOT we use a batch size of only 10.

Computational Considerations. While calculating unbalanced Optimal Transport measures is computationally more expensive, the significant decrease in batch size with UMOT makes its overall complexity roughly equivalent to MBOT. Interestingly, the regularized UMOT variant is even five times faster than MBOT. However, UMOT introduces an additional parameter, τ (and ε for the regularized version), that requires careful tuning. This adds an extra step to the training process. Nevertheless, depending on the specific data, the benefits of UMOT can outweigh this additional overhead. In later chapters, we will refer to the OT-CFM method that uses UMOT to approximate the optimal coupling as UOT-CFM.

Chapter 4

Action Matching

Action Matching follows a similar idea to Flow Matching. As before we try to learn a CNF in a simulation-free manner, i.e. without simulating the ODE (2.1.1) during training. This method was proposed by Neklyudov et. al. [Nek+23b] and we follow their paper in this chapter. The main difference to Flow Matching is that we assume that the vector field u driving the ODE (2.1.1) is a gradient field in x, which means $u = \nabla_x s$ for a function $s : [0,1] \times \mathbb{R}^d \to \mathbb{R}$. We will derive an implicit objective to match the vector field u. In general Action Matching assumes that we can sample at all time points along the trajectory.

In this chapter we identify the density p_t with its measure $p_t(\mathrm{d}x) \mathrel{\widehat{=}} p_t(x)\,\mathrm{d}x$. It will be clear from the context whether we refer to the density or the measure.

4.1 The Action Matching Objective

We now assume the particles follow the ODE

$$\mathrm{d}x_t = \nabla_x s_t(x_t)\,\mathrm{d}t, \tag{4.1.1}$$

where $s : [0,1] \times \mathbb{R}^d \to \mathbb{R}$ and $x_0 \sim \mu, x_1 \sim \nu$. We call s the 'action'. The explanation for this name is given in Section 4.2. As before let p_t describe the density at time t of the particle x_t and let us assume that the continuity equation is satisfied, i.e.

$$\frac{\partial p_t}{\partial t} = -\nabla \cdot (p_t \nabla_x s_t).$$

Remark 4.1.1. *If instead of $\mathcal{X} = \mathbb{R}^d$ we choose a open bounded set $\mathcal{X} = \Omega$ such that $\partial\mathcal{X}$ is C^1, then we interpret the continuity equation* (2.1.3) *with a homogenous Neumann condition*

$\langle v, \eta_\Omega \rangle = 0$ *at the boundary* $\partial\Omega$*, where* η_Ω *is the outer normal.*[1]

Our aim is to match or rather learn the vector field $u = \nabla_x s$, such that the ODE (4.1.1) transports the source density μ to the target density ν.

Therefore, we parametrize the action by a neural network $s_\theta(t, x)$ with learnable parameter θ. In order to learn the true action we would like to minimize

$$\text{ACTION-GAP}(s, s_\theta) = \frac{1}{2} \int_0^1 \mathbb{E}_{x \sim p_t} \|\nabla s_t(x) - \nabla s_\theta(t, x)\|^2 \, \mathrm{d}t. \tag{4.1.2}$$

However, we do not have access to the true action so this objective is intractable. Because of that we will decompose the loss into an intractable constant, i.e. a part that does not depend on θ, and a part that only depends on s_θ, but not on the true action s. Later we will only optimize over the latter part.

Theorem 4.1.2 (Theorem 2.2 in [Nek+23b])**.** *Let the sample space* $\mathcal{X} \subset \mathbb{R}^d$ *be an open and bounded set such that* $\partial\mathcal{X}$ *is* C^1*. Let* $s : [0, 1] \times \mathcal{X} \to \mathbb{R}$ *be the true action and* $s_\theta : [0, 1] \times \mathcal{X} \to \mathbb{R}$ *a neural network with learnable parameter* θ*. The ACTION-GAP*(s, s_θ) *can be decomposed as a sum of an intractable constant* $\mathcal{K}$*, and a tractable term* $\mathcal{L}_{AM}(\theta)$*:*

$$\textit{ACTION-GAP}(s, s_\theta) = \mathcal{K} + \mathcal{L}_{AM}(\theta). \tag{4.1.3}$$

The Action Matching objective $\mathcal{L}_{AM}(s_\theta)$ *is given by*

$$\begin{aligned} \mathcal{L}_{AM}(\theta) = {} & \mathbb{E}_{x \sim p_0}[s_\theta(0, x)] - \mathbb{E}_{x \sim p_1}[s_\theta(1, x)] \\ & + \int_0^1 \mathbb{E}_{x \sim p_t} \left[\frac{1}{2} \|\nabla s_\theta(t, x)\|^2 + \frac{\partial s_\theta}{\partial t}(t, x) \right] \mathrm{d}t. \end{aligned} \tag{4.1.4}$$

Remark. *In practice it is no restriction to assume* $\mathcal{X}$ *open and bounded as we will only have finitely many samples.*

Proof. We will decompose the ACTION-GAP to see that $\mathcal{L}_{AM}(\theta)$ is an equivalent learning objective.

$$\begin{aligned} & \text{ACTION-GAP}(s, s_\theta) \\ & = \frac{1}{2} \int_0^1 \int_{\mathcal{X}} p_t(x) \|\nabla s_t(x) - \nabla s_\theta(t, x)\|^2 \, \mathrm{d}x \, \mathrm{d}t \end{aligned}$$

[1] Definition from [Eva22]: Let $\mathcal{X} \subset \mathbb{R}^d$ be open and bounded. We say that $\partial\mathcal{X}$ is C^1 if for each point $x^0 \in \partial\mathcal{X}$ there exists $r > 0$ and a C^1 function $\gamma : \mathbb{R}^{n-1} \to \mathbb{R}$ such that - upon relabeling and reorenting the coordinate axes if necessary - we have

$$\mathcal{X} \cap B(x^0, r) = \{x \in B(x^0, r) : x_n > \gamma(x_1, \dots x_{n-1})\}.$$

$$
\begin{aligned}
&= \frac{1}{2}\int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_\theta(t,x)\|^2 \,\mathrm{d}x\,\mathrm{d}t - \int_0^1 \int_{\mathcal{X}} p_t(x)\langle \nabla s_t(x), \nabla s_\theta(t,x)\rangle \,\mathrm{d}x\,\mathrm{d}t \\
&\quad + \frac{1}{2}\int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_t(x)\|^2 \,\mathrm{d}x\,\mathrm{d}t \\
&= \frac{1}{2}\int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_\theta(t,x)\|^2 \,\mathrm{d}x\,\mathrm{d}t - \int_0^1 \int_{\mathcal{X}} \langle p_t(x)\nabla s_t(x), \nabla s_\theta(t,x)\rangle \,\mathrm{d}x\,\mathrm{d}t + \mathcal{K} \\
&\overset{(1)}{=} \frac{1}{2}\int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_\theta(t,x)\|^2 \,\mathrm{d}x\,\mathrm{d}t + \int_0^1 \int_{\mathcal{X}} s_\theta(t,x)\left[\nabla \cdot (p_t(x)\nabla s_t(x))\right]\mathrm{d}x\,\mathrm{d}t \\
&\quad - \int_0^1 \int_{\partial\mathcal{X}} s_\theta(t,x) p_t(x)(\nabla s_t(x)\cdot \eta_{\mathcal{X}})\,\mathrm{d}S^{n-1}(x) + \mathcal{K} \\
&\overset{(2)}{=} \frac{1}{2}\int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_\theta(t,x)\|^2 \,\mathrm{d}x\,\mathrm{d}t + \int_0^1 \int_{\mathcal{X}} s_\theta(t,x)\left[\nabla \cdot (p_t(x)\nabla s_t(x))\right]\mathrm{d}x\,\mathrm{d}t + \mathcal{K} \\
&\overset{(3)}{=} \int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_\theta(t,x)\|^2 \,\mathrm{d}x\,\mathrm{d}t - \int_0^1 \int_{\mathcal{X}} s_\theta(t,x)\frac{\partial p_t}{\partial t}(x)\,\mathrm{d}x\,\mathrm{d}t + \mathcal{K} \\
&\overset{(4)}{=} \int_0^1 \int_{\mathcal{X}} p_t(x)\|\nabla s_\theta(t,x)\|^2 \,\mathrm{d}x\,\mathrm{d}t - \int_{\mathcal{X}} s_\theta(1,x)p_1(x)\,\mathrm{d}x + \int_{\mathcal{X}} s_\theta(0,x)p_0(x)\,\mathrm{d}x \\
&\quad + \int_0^1 \int_{\mathcal{X}} \frac{\partial s_\theta(t,x)}{\partial t} p_t(x)\,\mathrm{d}x\,\mathrm{d}t + \mathcal{K} \\
&= \mathcal{L}_{AM}(\theta) + \mathcal{K}
\end{aligned}
$$

In (1) we used the divergence theorem. S^{n-1} denotes the $n-1$-dimensional Hausdorff measure. In (2) we used the fact that due to the Neumann boundary conditions from Remark 4.1.1 we have $\nabla s_t(x)\cdot\eta_{\mathcal{X}} = 0$ on $\partial\mathcal{X}$. Here $\eta_{\mathcal{X}}$ is the outer normal. In (3) we used the continuity equation (2.1.3). In (4) we used integration by parts.

As $\mathcal{K}$ does not depend on θ the learning objectives ACTION-GAP(s, s_θ) and $\mathcal{L}_{AM}(\theta)$ are equivalent. □

This leads to an implicit objective to optimize. In order to approximate the expected values algorithmically, we will use Monte Carlo simulation. The algorithm for Action Matching is specified in Algoritm 3.

Remark. *The neural network s_θ parametrizes s. So we need to differentiate the network to access the vector field $u = \nabla_x s$.*

Remark. *One can show that we can interpret $\nabla_x s$ as learning the optimal transport map between two infinitesimally close distributions on the given curve $(p_t)_{t\in[0,1]}$. While we will not go further into this topic in this thesis, more information can be found in Appendix B in [Nek+23b].*

Algorithm 3 Action Matching

Input: Probability density path $(p_t)_{t\in[0,1]}$, initial model s_θ, number of training iterations, batch size B

Output: Trained model $s_\theta(t, x)$

for training iterations **do**

$\boldsymbol{x_0} \leftarrow \{x_0^i\}_{i=0}^B$ with $x_0^i \sim p_0$

$\boldsymbol{x_1} \leftarrow \{x_1^i\}_{i=0}^B$ with $x_1^i \sim p_1$

$\boldsymbol{t} \leftarrow \{t_i\}_{i=0}^B$ with $t_i \sim U[0,1]$

$\boldsymbol{x_t} \leftarrow \{x_{t_i}\}_{i=1}^B$ with $x_{t_i} \sim p_{t_i}$

$\mathcal{L}_{AM}(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^B \left[s_\theta(x_0^i, 0) - s_\theta(x_1^i, 1) + \frac{1}{2}\|\nabla s_\theta(x_{t_i}, t_i)\|^2 + \frac{\partial s_\theta(x_{t_i}, t_i)}{\partial t}\right]$

$\theta \leftarrow \texttt{Update}(\theta, \nabla_\theta \mathcal{L}_{AM}(\theta))$

end for

Applying Action Matching in Discrete Time. Initially, the learning objective $\mathcal{L}_{\text{AM}}(\theta)$ requires access to the complete trajectory of the dynamics. However, in practice, this is rarely feasible. Moreover, it complicates comparisons with Flow Matching, which only requires knowledge of the initial and target distributions.

Although the authors do not address this issue in their paper, they suggest in a talk[2] that the loss can be discretized. This suggestion can be supported by the law of large numbers. However, we encounter two practical challenges: Firstly, the time points at which we approximate the expected value of $\mathcal{L}_{\text{AM}}(\theta)$ are not independently drawn but are determined by the data. Secondly, we need to ensure that we have sufficient time points to achieve a good approximation.

In Section 6.3, we tested how Action Matching performs with different numbers of available time points. The results were disappointing, as we faced significant issues with exploding gradients. In the single-cell experiment in Chapter 7, we partially learned the paths, but the results were still unsatisfactory.

To determine whether the problem is produced by the discretization or the loss, we will also test a version called Independent Action Matching, based on I-CFM, which interpolates the data between discrete time points. This version is introduced in Chapter 6.

Why Gradient Fields Are Not Restrictive?

While assuming the vector field to be a gradient fields seems restrictive, we actually get a gradient field naturally:

Theorem 4.1.3 (Adapted from Theorem 13.8. and Remark 13.9. in [Vil09])**.** *Let M be a smooth complete Riemannian manifold, I an open interval and $p : I \to \mathcal{P}_2(M)$ be an absolutely continu-*

[2] for the talk see https://www.youtube.com/watch?v=AdesAB80oRM (last visited 22.08.2024)

ous curve, where $\mathcal{P}_2(M)$ denotes the set of probability measures with finite second moment. Let H_t be the Hilbert space generated by gradients of continuously differentiable, compactly supported ψ:

$$H_t := \overline{Vect(\{\nabla\psi; \psi \in C_c^1(M)\})}^{L^2(p_t;TM)}.$$

Then there exist a measurable vector field $v \in L^2(\mathrm{d}p_t\,\mathrm{d}t)$, which is $p_t(\mathrm{d}x)\,\mathrm{d}t$ almost everywhere unique. Further, $v_t \in H_t$ for all t, meaning the vector field really is tangent along the path, and p and v satistfy the continuity equation (2.1.3) *in the weak sense.*

4.2 Why Action Matching is Called Action Matching?

We follow [Nek+23a] in this part.

Let $\mathcal{X}$ be the sample space endowed with a scalar product $\langle\cdot,\cdot\rangle$ and let $\gamma_t : [0,1] \to \mathcal{X}$ be a continuous curve, which has a metric derivative. An action functional measures the cost of displacement along this curve. Actions can be defined for general Langrangians, i.e. functions which summarize the dynamics of the entire system. We, however, restrict ourselves to systems without a potential. Namely, the action is simply given as the time integral over the kinetic energy of the curve γ:

$$\mathcal{A}(\gamma) = \int_0^1 \frac{\langle\dot\gamma_t, \dot\gamma_t\rangle}{2}\,\mathrm{d}t.$$

In Action Matching we want to match the action of the curve $t \mapsto p_t$ which maps from $[0,1]$ to $\mathcal{P}_2(\mathcal{X})$. This argument relies on Otto Calculus which we do not want to formally introduce here (see for example [ABS21]). Still we try to outline the ideas informally. The tangent space to p is defined by $T_p^{W_2}\mathcal{P}_2(\mathcal{X}) = \{\dot p \mid \int \dot p\,\mathrm{d}x_t = 0\}$, where $\dot p$ is the time derivative of p. An element $\dot p$ from the tangent space is 'coupled' to a vector $v = \nabla_x s_{\dot p}$ if they satisfy

$$\dot p = -\nabla\cdot(p\nabla_x s_{\dot p}).$$

This condition resembles the continuity equation in Optimal Transport. Next we define Otto's scalar product for two elements $f, g \in T_p^{W_2}\mathcal{P}_2(\mathcal{X})$ by

$$\langle f, g\rangle := \int (\nabla_x s_f)\cdot(\nabla_x s_g)p\,\mathrm{d}x.$$

Then p has the action

$$\mathcal{A}[p] = \int_0^1 \frac{\langle\dot p_t, \dot p_t\rangle}{2}\,\mathrm{d}t = \int_0^1\int \frac{1}{2}\|\nabla_x s_t\|^2 p_t(x)\,\mathrm{d}x\,\mathrm{d}t.$$

Action Matching aims at minimizing the displacement cost between the true curve and the learned curve.

4.3 Comparison to Flow Matching

At last we compare Action Matching to Flow Matching. The first difference is that a priori Flow Matching aims to interpolate between two distributions so just assumes access to samples at the start and end points of the path, while Action Matching requires access to samples at all points in time. However, one can also apply Flow Matching to several time steps and approximate Action Matching by a finite number of fixed time steps. If we let the number of time steps we can sample from go to infinity, then OT-CFM and AM learn the same probability path $(p_t)_{t\in[0,1]}$ which is optimal in the sense of infinitesimal Optimal Transport.

In Flow Matching we needed to know or at least choose the marginal conditional probability path $p_t(x \mid x_0, x_1)$ and conditional vector field $u_t(x \mid x_0, x_1)$. Action Matching requires us to know the whole probability path but we learn the unconditional vector field directly.

Additionally, the vector fields learned by each method differ. While choosing the vector field u as a gradient field is not restrictive (and is almost everywhere unique when chosen this way), in Flow Matching, we may learn a different vector field where this restriction does not apply. Without the gradient field restriction, we can always add a divergence-free component w_t with $\nabla \cdot (p_t w_t) = 0$ to u_t, and the continuity equation is still satisfied. The specific vector field learned also depends on the particular Flow Matching method used.

Let u_t^* denote the vector field learned by Flow Matching and ∇s_t^* the one learned by Action Matching. The divergence-free component of u_t^* is identical to ∇s_t^*. As a result, ∇s_t^* has lower kinetic energy than u_t^*

$$\frac{1}{2}\int_0^1 \mathbb{E}_{x\sim p_t}\|\nabla s_t^*(x)\|\,\mathrm{d}t \leq \frac{1}{2}\int_0^1 \mathbb{E}_{x\sim p_t}\|u_t^*(x)\|\,\mathrm{d}t.$$

Thus, the particles moved by ∇s_t^* have less kinetic energy and move less. Consequently, Action Matching may be more robust to noisy data, as it is less prone to learning the noise in the data. In Section 6.4 and Chapter 7 we will see that this is indeed the case.

Moreover, the Flow Matching loss is a simple mean-squared error loss, which allows for fast regression. In contrast, backpropagating the Action Matching loss is more computationally complex because the loss includes the gradient of the neural network.

While the computational complexity of Action Matching remains stable regardless of the number of time points N included, Flow Matching's complexity scales linearly with the number of time points, as we need to compute $N-1$ couplings. This does not hold for I-CFM as it just uses the independent coupling. We further compare these two methods in the experiments detailed in Chapter 6 and Chapter 7.

Chapter 5

Stochastic Variants

In the previous chapters, we assumed the particles $(x_t)_{t\in[0,1]}$ follow a deterministic ODE. We focused on transporting particles in an energy-efficient manner, for example by methods related to Optimal Transport, which results in straight-line trajectories of the particles. However, natural processes often include stochastic elements that cannot be described by an ODE. The stochastic process known as Brownian motion, $(W_t)_{t\geq 0}$, characterizes the random motion of particles in a liquid. To incorporate randomness in the trajectories, we add noise induced by Brownian motion to the driving process. This can be described by the following Stochastic Differential Equation (SDE), which we will explain in more detail shortly:

$$\mathrm{d}X_t = u_t(X_t)\,\mathrm{d}t + g(t)\,\mathrm{d}W_t. \tag{5.0.1}$$

While the ODE transports every particle starting at $X_0 = x_0$ to the same target $X_1 = x_1$, the SDE allows a particle starting at $X_0 = x_0$ to reach various different locations.

Stochastic dynamics are essential for describing many natural processes, such as the behavior of single cells, which we will encounter later. The gene expression dynamics of single cells involve inherent randomness and branching behaviors that deterministic dynamics cannot capture.

Additionally, Stochastic Differential Equations can enhance learning by enabling a more comprehensive exploration of the state space. Cutting-edge generative models, such as diffusion models [Son+21], utilize SDEs to introduce noise into training data. These models often outperform deterministic processes, particularly in high-dimensional settings. We aim to achieve similar results with our stochastic variants. This hypothesis is tested experimentally in Section 6.4, where we compare deterministic and stochastic models on artificial high-dimensional data, and in Chapter 7, where we evaluate the models on real single-cell data across various dimensions.

In this chapter we will introduce the methods entropic Action Matching and $[SF]^2M$, which is the stochastic variant of Flow Matching. Both methods aim at learning the drift term u_t

of the SDE. However, $g(t)$ needs to be chosen or fine-tuned manually. To learn $g(t)$ we would need access to trajectories of particles, i.e. we would need a time series structure of the data where we can follow specific particles over time. After we have learned u_t we can solve the SDE numerically to simulate trajectories of particles. Both methods avoid having to solve the SDE during training.

5.1 Preliminaries

5.1.1 SDEs and Diffusion Processes

We assume that the stochastic process $(X_t)_{t\in[0,1]}$ takes values in $\mathbb{R}^d$ and can be represented by the Itô-SDE:

$$\mathrm{d}X_t = u_t(X_t)\,\mathrm{d}t + g(t)\,\mathrm{d}W_t, \tag{5.1.1}$$

where $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ and $g : [0,1] \to \mathbb{R}_+$. It would also be possible to choose $g(t)$ such that it takes values in the space of positive definite diagonal $d \times d$-matrices. This can be useful if we want to describe processes whose noise scales vary over different dimensions. For ease of notation we restrict ourselves to $g : [0,1] \to \mathbb{R}_+$. The time marginal $X_t \in \mathbb{R}^d$ of the stochastic process $(X_t)_{t\in[0,1]}$ defined above describes the state of the particle at time t. $\mathrm{d}W_t$ is the increment of a d-dimensional Wiener process. We informally can think of this increment as $W_{t+\mathrm{d}t} - W_t$. $g(t)$ is the diffusion coefficient at time t and u_t is a smooth vector field, which is also called the drift of the SDE. The term $\mathrm{d}X_t$ describes the change in the particle's state at time t; for example, in single-cell mRNA data, it represents changes in gene expression over time. The drift term $u_t(X_t)$ depends on the temporal and spatial information of the particle, indicating the direction of movement at time t. The term $g(t)$ scales Brownian motion, accounting for the inherent randomness in the particle's movement at time t. More formally, the SDE (5.1.1) is defined as

$$X_{t_2} - X_{t_1} = \int_{t_1}^{t_2} u_t\,\mathrm{d}_t + \int_{t_1}^{t_2} g(t)\,\mathrm{d}W_t.$$

The stochastic integral

$$\int_{t_1}^{t_2} g(t)\,\mathrm{d}W_t$$

can be defined as the limit of Riemannian sums. Assume that $g(t)$ is a càdlag function, meaning it is right continuous and has left limits, and is locally bounded. If g were stochastic, it would also need to be adapted to the filtration. Let P be a partition of $[0,T]$ defined by $0 = t_0 < t_1 < \cdots < t_n = T$ and let

$$|P| = \max_{i=0,\ldots n-1}(t_{i+1} - t_i)$$

For any sequence of partitions $\{P_n\}_{n\in\mathbb{N}}$ of the time interval $[0,T]$, such that the maximal mesh width $|P_n| \to 0$ as $n \to \infty$, the Itô-Integral is defined as:

$$\int_0^t g(t)\,\mathrm{d}W_t = \lim_{n\to\infty} \sum_{[t_i,t_{i+1}]\in P_n} g(t_i)(W_{t_{i+1}} - W_{t_i}). \tag{5.1.2}$$

Remark 5.1.1. *The definition of the Itô stochastic integral is unique. It is crucial to use the value of $g(t)$ at the left border of the interval. Using another value within the interval $[t_i, t_{i+1}]$ would change the value of the integral.*

Next let $X_0 \sim \mu$. Moving this density by the SDE produces a path of probability distributions $p : [0,1] \times \mathbb{R}^d \to \mathbb{R}_+$. This probability path is characterized by its initial condition $p_0 = \mu$ and the Fokker-Planck equation:

$$\frac{\partial p_t}{\partial t} = -\nabla \cdot (p_t u_t) + \frac{g(t)^2}{2}\Delta p_t. \tag{5.1.3}$$

In the degenerate case $g(t) = 0$ the SDE (5.1.1) becomes just an ODE and the Fokker-Planck equation (5.1.3) recovers the continuity equation (2.1.3).

5.1.2 Probability Flow ODE

Song et. al. [Son+21] showed that we can restore the marginal distributions $(p_t)_{t\in[0,1]}$ of a SDE by solving an ODE which we call the Probability Flow ODE. In order to sample from p_t it is thus sufficient to solve this ODE. It is given by

$$\mathrm{d}x_t = \left[u_t(x_t) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x_t)\right] \mathrm{d}t. \tag{5.1.4}$$

Transporting an initial distribution μ along (5.1.4) yields the same probability path $(p_t)_{t\in[0,T]}$ as transporting it along our initial SDE (5.1.1).

Let $f_t(x) = u_t(x) - \frac{1}{2}g(t)^2\nabla_x \log p_t(x)$. We can parametrize $f_t(x)$ by a neural network $f_\theta(t,x)$ with learnable weights θ. We then can learn the vector field f by the previous methods as it is just the drift of the Probability Flow ODE . If we have access to $f_t(x)$, $g(t)$ and the score function $\nabla_x \log p_t(x)$ we can restore the original SDE:

$$\mathrm{d}X_t = \left[f_t(X_t) + \frac{1}{2}g(t)^2 \nabla \log p_t(X_t)\right] \mathrm{d}t + g(t)\,\mathrm{d}W_t. \tag{5.1.5}$$

5.1.3 Numerically Solving the SDE

Given the SDE (5.1.1) we want to simulate particles $(X_t)_{t\in[0,1]}$ which evolve according to this SDE. For this we need a numerical solution to the SDE. We use the Euler-Maruyama method,

which is an extension of the Euler method for ODEs. Let $Y_0 \sim \mu$. Let $0 = t_0 < t_1 < \cdots < t_n = 1$ be a partition of the interval $[0, 1]$ of mesh width $\Delta t = \frac{1}{n}$. We can define recursively for $0 \leq i \leq n - 1$

$$Y_{i+1} = Y_i + u_{t_i}(Y_i)\Delta t + g(t_i)\Delta W_i \tag{5.1.6}$$

with $\Delta W_i = W_{t_{i+1}} - W_{t_i}$. By the properties of the Wiener process the random variables ΔW_i are independent and identical distributed Gaussian random variables with expectation 0 and variance Δt. The Euler-Maruyama method has a strong convergence rate of $\frac{1}{2}$. To define this thoroughly let Y_T^h be the numerically solution of an SDE on the interval $[0, T]$ with step length h and let y_T be the real solution at time T. Then there exist constants C and $h_0 > 0$ such that for all $h \in (0, h_0)$ we have

$$\mathbb{E}\left[\left|y_T - Y_T^h\right|\right] \leq Ch^{\frac{1}{2}}.$$

The proof for this can be found in [Klo+92].

5.2 Entropic Action Matching

Entropic Action Matching is the extension of Action Matching to learn SDEs given by (5.1.1). Again we follow [Nek+23b]. As before we assume that the vector field u is a gradient field. That is $u_t(x) = \nabla_x s_t(x)$ with $s : [0, 1] \times \mathbb{R}^d \to \mathbb{R}$. We call s 'entropic action'. Further we have access to the whole probability path $(p_t)_{t\in[0,1]}$. Entropic Action Matching parametrizes the action s by a neural network s_θ with learnable weight θ. We will derive a learning objective to match the true and the paramterized action functions. Even though the diffusion coefficient $g(t)$ is fixed and not learned, it has an influence on the dynamics. While we derived the unentropic Action Matching loss $\mathcal{L}_{AM}$ by using the continuity equation, we now will use its stochastic counterpart - the Fokker-Planck equation (5.1.3).

As before it can be shown that it is no restriction to choose the vector field as a gradient field, which moves particles in time in such a way that the marginal probabilities are p_t. In Theorem 4.1.3 we saw this for the deterministic case. Together with the Probability Flow ODE (5.1.4) we get the following Proposition:

Theorem 5.2.1 (Proposition 3.1. in [Nek+23b])**.** *Consider an absolutely continuous density evolution $p : I \to \mathcal{P}_2(\mathbb{R}^d)$ with I an open interval, and suppose $g(t)$ is given. Then, there exists an unique (up to a constant) function $s_t(x)$, such that the vector field $u_t = \nabla_x s_t(x)$ and p satisfy the Fokker-Planck equation* (5.1.3)*.*

We now parametrize the entropic action by a neural network $s_\theta(t, x) : [0, 1] \times \mathbb{R}^d \to \mathbb{R}$ with learnable parameter θ. The desired learning objective is the same as in the deterministic case:

$$\text{E-ACTION-GAP}(s, s_\theta) = \frac{1}{2}\int_0^1 \mathbb{E}_{x\sim p_t}\|\nabla s_t(x) - \nabla s_\theta(t, x)\|^2 \,\mathrm{d}t. \tag{5.2.1}$$

As this is intractable we again decompose this objective into a intractable constant and a tractable learning objective:

Theorem 5.2.2 (Theorem 3.2 in [Nek+23b]). *Let the sample space $\mathcal{X} \subset \mathbb{R}^d$ be an open and bounded set such that $\partial\mathcal{X}$ is C^1. Let $s : [0,1] \times \mathcal{X} \to \mathbb{R}$ be the true entropic action and $s_\theta : [0,1] \times \mathcal{X} \to \mathbb{R}$ the neural network with learnable parameter θ. The E-ACTION-GAP(s, s_θ) can be decomposed as a sum of an intractable constant $\mathcal{K}$, and a tractable term $\mathcal{L}_{eAM}(s_\theta)$:*

$$\textit{E-ACTION-GAP}(s, s_\theta) = \mathcal{K} + \mathcal{L}_{eAM}(s_\theta). \tag{5.2.2}$$

The entorpic Action Matching objective $\mathcal{L}_{AM}(s_\theta)$ is given by

$$\begin{aligned}\mathcal{L}_{eAM}(s_\theta) = \mathbb{E}_{x\sim p_0}[s_\theta(0,x)] - \mathbb{E}_{x\sim p_1}[s_\theta(1,x)] \\ + \int_0^1 \mathbb{E}_{x\sim p_t}\left[\frac{1}{2}\|\nabla s_\theta(t,x)\|^2 + \frac{\partial s_\theta}{\partial t}(t,x) + \frac{g(t)^2}{2}\Delta s_\theta(t,x)\right] \mathrm{d}t.\end{aligned} \tag{5.2.3}$$

The proof of this theorem is similiar to Theorem 4.1.2 and will be skipped. The main difference is that we use the Fokker-Planck equation instead of the continuity equation.

5.2.1 The Hutchinson Trace Estimator

The learning objective (5.2.3) requires the Laplacian of the parameterized action, which is the trace of the Hessian. In high dimensions, calculating the Hessian and its trace can be computationally intensive and storage demanding, as the Hessian is a $d \times d$ matrix. The neural network s_θ has several hidden layers, each containing even more hidden units. Even in relatively small dimensions, computing this Hessian can be expensive. Additionally, since the Laplacian appears in the loss function, backpropagating through it can lead to an unstable training process, even in lower dimensions.

Instead of calculating the entire Hessian we can use matrix-vector products of the Hessian of $\nabla_x s_\theta$ and vectors v_i, $i = 1, \ldots, n$. This product can be calculated efficiently by autodifferentiation in the popular machine learning frameworks (PyTorch, TensorFlow, JAX). Let $A \in \mathbb{R}^{d\times d}$ be a positive semi-definite matrix like the Hessian. The exact trace of A can be computed as

$$\sum_{i=1}^{d} e_i^T (Ae_i)$$

where e_i is the i-th canonical basis vector of $\mathbb{R}^d$. However, in high dimensions, this requires significant computation, which is impractical to compute in each training step in a machine learning algorithm.

Instead, we use the Hutchinson trace estimator [Hut90], as detailed in [Sko21]. The Hutchinson estimator is given by:

$$\mathrm{tr}_H(A) = z^T A z,$$

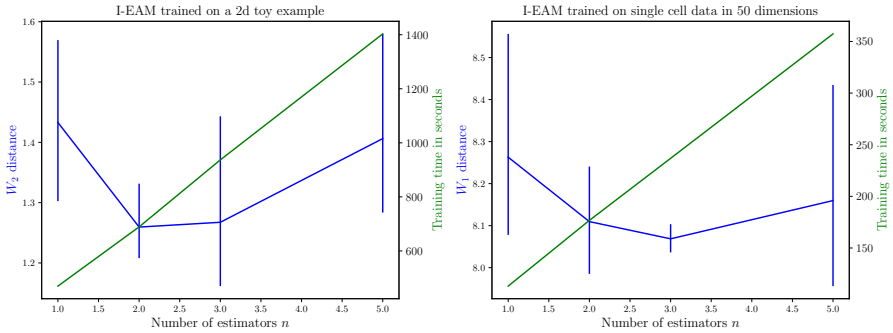

Figure 5.1: Results of fine-tuning the number of estimators n for I-EAM. Both experiments were run for three runs. The blue plot shows the average W_2 or W_1 distance between the predicted and observed distributions over the three runs. The vertical blue lines are error plots, indicating the minimum and maximum distance over the three runs. The green line displays the average computational time.

where z is a Rademacher random variable, i.e. $z \sim U(\{-1,1\}^d)$. While this is an unbiased estimator it can be very noisy. Therefore, we usually average over n independent trials in practice:

$$\mathrm{tr}_{H^{(n)}}(A) = \frac{1}{n}\sum_{i=1}^{n} z_i^T A z_i \tag{5.2.4}$$

and z_i for $i = 1, \ldots, n$ are independent Rademacher variables. It is easy to see that this estimator remains unbiased. To measure how good the estimator performs we can look at its variance:

$$\mathrm{Var}(\mathrm{tr}_{H^{(n)}}(A)) = \frac{4}{n}\sum_{1\le i<j\le d} |A_{ij}|^2.$$

Thus the variance decreases linearly with the number of trials n.

Choosing the number of estimators n. To determine a suitable number of estimators n, we tuned this hyperparameter for two experiments that we will conduct later. The other hyperparameters are chosen in the same way as in Chapter 6. The details are outlined in Section 6.1. For computational stability, we use I-EAM, a version of EAM that interpolates the data between discrete time steps, ensuring a more stable training objective. This method is introduced in Section 6.1. We fine-tuned n using the two dimensional toy example which we introduce in Chapter 6 and the single cell data from Chapter 7. We ran both experiments three times. Figure 5.1 shows the results, indicating that the average computational complexity (green line) increases linearly with the number of estimators n; a result which was also expected. The blue line displays the average W_2 or W_1 distance between the predicted and observed distributions over the three runs. The vertical blue lines are error plots, indicating the minimum and maximum Wasserstein

distance over the three runs. It is desirable that the average distance is low but also the error is not too large. The results suggest that $n = 2$ or $n = 3$ might be good choices. In this thesis, we use $n = 3$ as it provides more stability for the EAM method when dealing with dynamics in discrete time.

However, we observe two surprising results. Firstly, the error of entropic Action Matching increases when using $n = 5$ estimators. A possible explanation for this is that a less accurate Laplacian regularizes entropic Action Matching. This is further supported by the observation that entropic Action Matching becomes very unstable with $n = 5$, as indicated by the large error bars. In Chapter 6, we demonstrate that entropic Action Matching is generally an unstable method.

Moreover, the significant difference between the estimators with $n = 1$ and $n = 2$ is unexpected. During training, we already average over the data from one batch, so it would be reasonable to expect that a single estimator would yield fairly good results. In fact, using just one estimator is a common practice when training CNFs. This suggests that the Laplacian term in the Action Matching objective might contribute to an unstable loss function, and that using more than one estimator helps to stabilize it.

5.2.2 Connection to Entropic Optimal Transport

In Section 2.2, we discussed the Schrödinger bridge problem. It searches for the most likely measure $\boldsymbol{D}^* \in \mathcal{P}(C([0,1];\mathcal{X}))$, such that its time marginals D_0^* and D_1^* agree with μ and ν and with the condition that $\boldsymbol{D}^*$ should be as close as possible to the law of a Brownian motion with diffusion $g(t)$. We want to show that entropic Action Matching is connected to entropic Optimal Transport. We follow [CGP21] in this section.

Chen et. al. [CGP21] show that the Schrödinger bridge problem can be formulated as a stochastic control problem and that the marginal paths $(D_t^*)_t$ can be restored by solving the fluid dynamic formulation:

Definition 5.2.3. The fluid dynamic formulation of entropic Optimal Transport is given by

$$\begin{aligned} &\inf_{p,u} \int_{\mathbb{R}^n} \int_0^1 \frac{1}{2}\|u(t,x)\|^2 p(t,x)\,\mathrm{d}t\,\mathrm{d}x, \\ &\frac{\partial p}{\partial t} + \nabla \cdot (up) - \frac{g(t)}{2}\Delta p = 0, \\ &p(0,x) = \mu(x), \quad p(1,y) = \nu(y) \end{aligned}$$

with $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ and $p : [0,1] \to \mathcal{P}(\mathbb{R}^d)$.

We get that $D_t^* = p_t$ for the optimal solution $(p_t)_{t\in[0,1]}$ of the fluid dynamic problem. The fluid dynamic formulation can be written as a saddle-point problem by introducing a space-time

dependent Lagrange multiplier $s(t,x)$. After partial integration the Lagrangian is given by:

$$L(s,u,p) = \int_0^1 \int_{\mathbb{R}^n} \frac{1}{2}\|u_t(x)\|^2 p_t(x) - \left(\frac{\partial s_t(x)}{\partial t} + \langle u_t(x), \nabla s_t(x)\rangle - \frac{g(t)^2}{2}\Delta s_t(x)\right) p_t(x)\,\mathrm{d}x\,\mathrm{d}t + \int_{\mathbb{R}^n} s_1(x)p_1(x) - s_0(x)p_0(x)\,\mathrm{d}x.$$

Then the fluid dynamic formulation is equivalent to the saddle-point problem

$$\inf_{p,u}\sup_{s} L(s,u,p).$$

In the Action Matching setting we assume p to be known, so the saddle point simplifies to

$$\inf_{u}\sup_{s} L(s,u,p).$$

[BB00] show that the optimality condition for the vector field u_t is $u_t(x) = \nabla_x s_t(x)$, where s is the Langrangian. Thus, it remains to solve

$$\sup_{s} L(s,\nabla s,p).$$

Maximizing this is equivalent to minimzing the entropic Action Matching loss. Thus entropic Action Matching solves the fluid dynamic formulation of entropic Optimal Transport, where the probability path $(p_t)_{t\in[0,1]}$ is known.

5.3 Simulation Free Score and Flow Matching

We proceed by examining a generalized form of Flow Matching called $[SF]^2M$, extending Flow Matching to stochastic dynamics as described in Tong et al. [Ton+23]. This approach combines the learning objectives of Score Matching [Son+21] with Flow Matching, thereby extending Score Matching to arbitrary source distributions and broadening Flow Matching to stochastic dynamics. Before going into the details of this method, we give a brief overview of Score Matching.

5.3.1 Score Matching

Score based models were introduced to facilitate more efficient sampling from distributions with complicated density functions, denoted by $p(x)$, which are challenging to sample from directly. The key idea is to learn the gradient of the log-likelihood, also called the score function, denoted by $\nabla_x \log p(x)$. The score function eliminates the need to compute the normalizing constant. We

would like to learn a neural network $s_\theta \approx \nabla_x \log p(x)$ with learnable weights θ, approximating the score function, by optimizing the objective

$$\frac{1}{2}\mathbb{E}_{x\sim p}\left[\|\nabla_x \log p(x) - s_\theta(x)\|\right]. \tag{5.3.1}$$

However, in practice this is infeasable as the true score function is mostly unavailable. Estimating the score function from the data would be a non-parametric estimation problem. Instead, by simply using partial integration we get the objective in [HD05]:

$$\mathbb{E}_{x\sim p}\left[\operatorname{tr}(\nabla_x s_\theta(x)) + \frac{1}{2}\|s_\theta(x)\|_2^2\right]. \tag{5.3.2}$$

This can be learned more easily. As (5.3.2) depends on the computation of $\operatorname{tr}(\nabla_x s_\theta(x))$ this approach is still not scalable to deep neural networks and high dimensional data. To overcome this problem Sliced Score Matching [Son+20] and Denoising Score Matching [Vin11], [SE19] were introduced. We will give a short summary of Denoising Score Matching.

Denoising Score Matching. Instead of directly estimating the score for the original data distribution, Denoising Score Matching adds noise to the data and estimates the score function of the perturbed data. This is especially beneficial in regions of low data density. For better results we perturb the data with different scales of noise. For a given noise scale σ we define the perturbed density as

$$q_\sigma(\tilde{x}) = \int \mathcal{N}(\tilde{x} \mid x, \sigma^2 I) p(x)\,\mathrm{d}x. \tag{5.3.3}$$

We get $q_{\sigma_i}(\tilde{x} \mid x) = \mathcal{N}(\tilde{x} \mid x, \sigma_i^2 I)$. We can match the noisy score function by the objective

$$\frac{1}{2}\mathbb{E}_{x\sim p}\left[\|\nabla_x \log q_\sigma(x) - s_\theta(x)\|\right]$$

This loss can be shown to be equivalent to the conditional loss

$$\frac{1}{2}\mathbb{E}_{x\sim p, \tilde{x}\sim q_{\sigma_i}(\cdot|x)}\left[\|\nabla_x \log q_\sigma(\tilde{x} \mid x) - s_\theta(x)\|\right].$$

Suppose we have the noise scales $\sigma_1, \ldots, \sigma_L$. We want to estimate the score functions of all the perturbed densities $q_{\sigma_1}, \ldots, q_{\sigma_L}$. To train a network $s_\theta(x, \sigma_i)$ for all noise scales we can use a weighted sum of the conditional objectives. The sum is weighted by $\lambda(\sigma_i) \in \mathbb{R}_{\geq 0}$. Then the objective is given by

$$\frac{1}{L}\sum_{i=1}^{L} \lambda(\sigma_i)\mathbb{E}_{x\sim p}\mathbb{E}_{\tilde{x}\sim q_{\sigma_i}(\cdot|x)}\left[\|s_\theta(\tilde{x}, \sigma_i) - \nabla_x \log q_{\sigma_i}(\tilde{x} \mid x)\|^2\right]. \tag{5.3.4}$$

If the minimum noise scale $\sigma_{\min}$ is small enough we get $s_\theta(x, \sigma_{\min}) \approx \nabla_x \log p(x)$. To recover samples $\tilde{x}_t$ approximately drawn from p_t using the learned score function s_θ, one can apply a procedure called annealed Langevin dynamics [SE19].

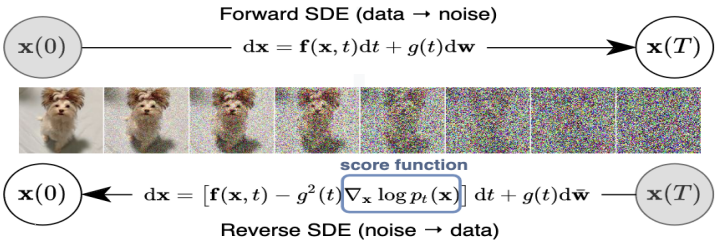

Figure 5.2: Score Matching with SDEs which was introduced by Song et. al. in 2021. The figure is taken from their paper [Son+21]

Score Matching with SDEs. Score Matching with SDEs, introduced by Song et al. [Son+21], extends the principles of Denoising Score Matching by employing a continuum of noise scales, achieved through an SDE framework. We will follow their framework in this section. Specifically, a diffusion process $(X_t)_{t=0}^T$ is constructed, such that $X(0) \sim \mu$ and $X(T) \sim p_T$. Here we have access to i.i.d. samples from μ and p_T is a distribution we can easily sample from, usually we choose it to be a Gaussian distribution. $(X_t)_{t=0}^T$ is modeled by an Itô-SDE:

$$\mathrm{d}X_t = f(X_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}W_t, \tag{5.3.5}$$

where $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ represents the drift and $g : \mathbb{R} \to \mathbb{R}$ represents the diffusion coefficient. Both f and g are chosen and not learned. The SDE perturbs the data into a form that allows for easy sampling.

To generate samples we then reverse the diffusion process. We aim to transport samples from $X_T \sim p_T$ to $X_0 \sim \mu$. It was shown in [And82] that under some conditions on the forward diffusion process, the reverse-time SDE of (5.3.5) is given by

$$\mathrm{d}X_t = \left[f(X_t, t) - g(t)^2 \nabla_x \log p_t(X_t)\right] \mathrm{d}t + g(t)\,\mathrm{d}\bar{W}_t, \tag{5.3.6}$$

where we now move from T to 0 and $\bar{W}$ is the standard Brownian motion backwards in time from T to 0. The proof is based on deriving reverse-time Kolmogorov equations and can be found in [And82]. The only unknown in (5.3.6) is the score function $\nabla_x \log p_t(x)$. We learn this by the conditional objective

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}([0,T])}\mathbb{E}_{X_0\sim p_0}\mathbb{E}_{X_t\sim p_{0,t}(\cdot|X_0)}\left[\lambda(t)\|s_\theta(X_t, t) - \nabla_x \log p_{0,t}(X_t \mid X_0)\|_2^2\right], \tag{5.3.7}$$

where $\lambda : [0, T] \to \mathbb{R}_+$ is a positive weighting function and $p_{s,t}(X_t \mid X_s)$ denotes the transition kernel from X_s to X_t. This loss is the continuous version of the one in Denoising Score Matching (5.3.4).

The commonly utilized SDEs include the Variance Exploding SDE, Variance Preserving SDE, and sub-Variance Preserving SDE, each serving specific purposes within the framework. The Variance

Exploding SDE (VE SDE) always yields a process with exploding variance when $t \to \infty$. The VE SDE can be thought of as the continuous version of the above Denoising Score Matching. For each of N noise scales the perturbation kernel $p_{\sigma_i}(x \mid x_0)$ represents the distribution of x_i in the following Markov chain:

$$x_i = x_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2} z_{i-1},$$

for $i = 1, \dots, N$, $z_{i-1} \sim \mathcal{N}(0, I)$ and $\sigma_0 = 0$. The continuous time version of this is given by the VE SDE:

$$\mathrm{d}X_t = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\,\mathrm{d}W_t.$$

The VP SDE (Variance preserving SDE) yields a process with a fixed variance of one if the initial distribution has a variance of one. It is given by

$$\mathrm{d}X_t = -\frac{1}{2}X_t\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}W_t,$$

where $\beta : [0, T] \to (0, 1)$ is a function of positive noise scales. This method is a continuous version of Denoising Diffusion Probabilistic Models [HJA20], which we did not introduce here.

The sub-VP SDE is inspired by the VP SDE. It has its name by the fact that the variance of the stochastic process $(X_t)_{t\in[0,1]}$ defined by the sub-VP SDE is bounded by the variance of the stochastic process defined by the VP SDE at every intermediate time step when using the same initial distribution and the same $\beta(t)$. It is given by

$$\mathrm{d}X_t = -\frac{1}{2}\beta(t)X_t\,\mathrm{d}t + \sqrt{\beta(t)(1 - e^{-2\int_0^t \beta(s)\,\mathrm{d}s})}\,\mathrm{d}W_t.$$

5.3.2 The $[SF]^2M$-Method

We consider the evolution of particles according to the SDE (5.1.1), with marginal densities p_t. We saw that the Probability Flow ODE (5.1.4) with the vector field

$$f_t(x) = u_t(x) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x)$$

yields identical marginal distributions. Additionally, from (5.1.5) we know that the SDE can be reconstructed given knowledge of f, g and $\nabla_x \log p$. Hence, we use Flow Matching to learn the vector field f and Score Matching to learn the score function $\nabla_x \log p$, while g is assumed to be known. We parametrize both using neural networks: $v_\theta : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ for the vector field and $s_\theta : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ for the score function. The natural objective is the loss function

$$\mathcal{L}_{U[SF]^2M}(\theta) = \mathbb{E}_{t\sim U([0,1]),x\sim p_t}\left[\|v_\theta(t, x) - f_t(x)\|^2 + \lambda(t)^2\|s_\theta(t, x) - \nabla_x \log p_t(x)\|^2\right], \quad (5.3.8)$$

where $\lambda : [0, 1] \to \mathbb{R}_+$ are positive weights, which need to be chosen manually.

To learn f and $\nabla_x \log p$ both Flow Matching and Score Matching use conditional objectives. In Chapter 3 we assume that the probability path $(p_t)_{t\in[0,1]}$ can be written as a mixture of the conditional probability path $(p_t(\cdot \mid z))_{t\in[0,1]}$. We will generalize the objects from Chapter 3 to the stochastic setting. We suppose that the stochastic process $\boldsymbol{X} = (X_t)_{t\in[0,1]}$, which we want to learn, can be written as a mixture of conditional random variables $(Y_t^z)_{t\in[0,1]}$ indexed by a latent variable $z \in \mathcal{Y}$, where $\mathcal{Y}$ is a latent space. We usually choose $\mathcal{Y} = \mathbb{R}^d \times \mathbb{R}^d$. The marginals X_t then can be written as

$$X_t = \int Y_t^z q(z)\,\mathrm{d}z. \tag{5.3.9}$$

Suppose that Y_t^z is defined by the SDE

$$\mathrm{d}Y_t^z = u_t(Y_t^z \mid z)\,\mathrm{d}t + \sigma_t\,\mathrm{d}W_t \tag{5.3.10}$$

with initial distribution $p_0(x \mid z)$. Further, let $f_t(x \mid z)$ denote the drift of the corresponding Probability Flow ODE. We have the following theorem:

Theorem 5.3.1 (Theorem 3.1. in [Ton+23])**.** *Let*

$$f_t(x) = \mathbb{E}_{z\sim q}\left[\frac{f_t(x \mid z)p_t(x \mid z)}{p_t(x)}\right], \tag{5.3.11}$$

$$\nabla \log p_t(x) = \mathbb{E}_{z\sim q}\left[\frac{p_t(x \mid z)}{p_t(x)}\nabla \log p_t(x \mid z)\right]. \tag{5.3.12}$$

Suppose that $u_t(x \mid z), \nabla \log p_t(x \mid z)$ and their first derivatives are continuous in x and uniformly continuous in z. Then the ODE $\mathrm{d}x_t = f_t(x)\,\mathrm{d}t$ generates the marginals p_t, where p_t is the density of X_t, from initial condition p_0 and its score function is given by (5.3.12)*. The SDE*

$$\mathrm{d}\tilde{X}_t = \left[f_t(\tilde{X}_t) + \frac{1}{2}\sigma_t^2 \nabla \log p_t(\tilde{X}_t)\right]\mathrm{d}t + \sigma_t\,\mathrm{d}W_t \tag{5.3.13}$$

generates the Markovization of $\boldsymbol{X}$, i.e. the Markov process with the same infinitesimal transition kernel.

Processes $\boldsymbol{X} = (X_t)_{t\in[0,1]}$ which have the form (5.3.9), are not necessarily Markovian and may not be generated by any SDE. Thus, the solution of (5.3.13), $\tilde{\boldsymbol{X}}$, must not be the same process as $\boldsymbol{X}$, however, they have the same infinitesimal transition kernel. If $\boldsymbol{X}$, on the other hand, is defined by a SDE and thus a Markov process, then $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$ are indistinguishable, i.e. $X_t = \tilde{X}_t$ almost surely for all $t \in [0, 1]$.

Based on Flow and Score Matching we now can define the conditional objective to solve $[SF]^2M$

$$\mathcal{L}_{[SF]^2M}(\theta) = \mathbb{E}_{t,z,x}\|v_\theta(t,x) - f_t(x \mid z)\|^2 + \mathbb{E}_{t,z,x}\lambda^2(t)\|s_\theta(t,x) - \nabla \log p_t(x \mid z)\|^2, \tag{5.3.14}$$

with $t \sim U([0,1])$, $z \sim q(\cdot)$ and $x \sim p_t(\cdot \mid z)$. Further $\lambda : [0,1] \to \mathbb{R}_+$ is a weight function which needs to be chosen. This is equivalent to the unconditional loss:

Theorem 5.3.2 (Theorem 3.2. in [Ton+23]). *Let $q, p_t(\cdot \mid z) \in L^2(\mathbb{R}^d)$ and f_t, $\nabla \log p_t$, $v_\theta(t,\cdot)$, $s_\theta(t,\cdot)$, $\nabla_\theta v_\theta(t,\cdot)$ and $\nabla_\theta s_\theta(t,\cdot)$ be bounded. If $p_t(x) > 0$ for all $x \in \mathbb{R}^d$ and $t \in [0,1]$, then*

$$\nabla_\theta \mathcal{L}_{U[SF]^2M} = \nabla_\theta \mathcal{L}_{[SF]^2M}. \tag{5.3.15}$$

Proof. We show this seperately for the Flow Matching and the Score Matching loss. For the Flow Matching loss we already showed the statement in Theorem 3.1.2. As we have

$$w_t(x) = \mathbb{E}_{z\sim q}\left[\frac{w_t(x \mid z)p_t(x \mid z)}{p_t(x)}\right]$$

for both $w_t(x) = f_t(x)$ and $w_t(x) = \nabla \log p_t(x)$ we just can repeat the proof of Theorem 3.1.2 for $\nabla \log p_t$. □

The algorithm for training is given in Algorithm 4.

Analogue to the Conditional Flow Matching framework we choose $z = (x_0, x_1)$. Further we choose $p_t(x \mid x_0, x_1)$ as in SB-CFM and therefore

$$f_t(x \mid x_0, x_1) = \frac{1-2t}{t(1-t)}(x - (tx_1 + (1-t)x_0)) + (x_1 - x_0), \tag{5.3.16}$$

$$\nabla \log p_t(x \mid x_0, x_1) = \frac{tx_1 + (1-t)x_0 - x}{\sigma^2 t(1-t)}. \tag{5.3.17}$$

We take $\sigma = 1$. As we want to transport the distribution μ to ν, we choose the mixing density q from (5.3.9) to be a coupling of μ and ν.

In Section 3.2.3 we chose $q = \pi_{2\sigma^2}$ for SB-CFM, where $\pi_{2\sigma^2}$ is the solution of entropic Optimal Transport with regularization parameter $2\sigma^2$. We saw that with this choice SB-CFM recovers the marginals of the Schrödinger bridge. When we now choose $q = \pi_{2\sigma^2}$ in the stochastic setting we recover the Schrödinger bridge to the reference process σW, where W is a Brownian motion. To make this more precise let v_θ^* and s_θ^* be the minimizers of $\mathcal{L}_{[SF]^2M}(\theta)$. Then the Schrödinger bridge is defined by the SDE

$$\mathrm{d}\tilde{X}_t = \left[v_\theta^* + \frac{1}{2}\sigma_t^2 s_\theta^*\right]\mathrm{d}t + \sigma_t\,\mathrm{d}W_t,$$

i.e. the Schrödinger bridge $\boldsymbol{D}^* \in \mathcal{P}(C([0,1];\mathbb{R}^d))$ is the law of the stochastic process $(\tilde{X}_t)_{t\in[0,1]}$. Therefore we will choose $q = \pi_{2\sigma^2}$. This of course only holds true when we know the exact distributions μ and ν. Thus, in practice we only recover an empirical Schrödinger bridge. Suppose we have n samples from each μ and ν, let $\boldsymbol{D}^*$ be the Schrödinger bridge between μ and ν and $\hat{\boldsymbol{D}}_n$ the Schrödinger bridge recovered from n samples. Then Stromme [Str23] gives the following bound:

$$\mathrm{KL}(\hat{\boldsymbol{D}}_n \mid \boldsymbol{D}^*) \in O\left(\frac{1}{\sqrt{n}}\right).$$

If we have n samples from each μ and ν, then calculating the entropic Optimal Transport has complexity $O(n^2)$. This might be too large in a machine learning application. Thus as for OT-CFM we can use a minibatch approximation of entropic Optimal Transport.

Algorithm 4 Algorithm for $[SF]^2M$

Input: Coupling q of the initial distribution μ and the target distribution ν, initial networks $v_\theta(t,x)$ and $s_\theta(t,x)$, number of training iterations, batch size B

Output: trained model $v_\theta(t,x)$

for training iterations **do**

$(\boldsymbol{x_0}, \boldsymbol{x_1}) \leftarrow \{(x_0^i, x_1^i)\}_{i=0}^B$ with $(x_0^i, x_1^i) \sim q$

$\boldsymbol{t} \leftarrow \{t_i\}_{i=0}^B$ with $t_i \sim U([0,1])$

$\boldsymbol{x_t} \leftarrow \{x_{t_i}^i\}_{i=1}^B$ with $x_{t_i} \sim p_{t_i}(\cdot \mid x_0^i, x_1^i)$

$$\mathcal{L}_{[SF]^2M}(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \|v_\theta(t_i, x_{t_i}^i) - f_{t_i}(x_{t_i}^i \mid x_0^i, x_1^i)\|^2 + \lambda^2(t_i)\|s_\theta(t_i, x_{t_i}^i) - \nabla \log p_{t_i}(x_{t_i}^i \mid x_0^i, x_1^i)\|^2$$

$\theta \leftarrow \texttt{Update}(\theta, \nabla_\theta \mathcal{L}_{[SF]^2M}(\theta))$

end for

Chapter 6

Experiments

We want to test Action Matching against Conditional Flow Matching and our stochastic methods against their deterministic counterparts. We want to understand better what kind of trajectories are learned, how the different methods behave in high dimensions and how well they scale if we use several time points or in the case of Action Matching how poorly our method behaves if we only give it acess to a limited number of time steps.

6.1 General Setup

Let d denote the dimensionality of the data, i.e. $\mathcal{X} \subseteq \mathbb{R}^d$. As a short recap Table 6.1 shows the methods and some of their assumptions in comparison. I-AM and I-EAM are introduced in the next section.

	I-CFM	OT-CFM	$[SF]^2M$	AM	I-AM	EAM	I-EAM
Requires knowledge of the entire probability path $(p_t)_t$?	False	False	False	True	False	True	False
Requires us to choose a suitable conditional probability path $(p_t(\cdot \mid z))_t$?	True	True	True	False	True	False	True
Can model stochastic dynamics?	False	False	True	False	False	True	True
Restores a form of OT or the Schrödinger bridge?	False	True	True	True	False	True	False

Table 6.1: We compare assumptions and capabilities of the introduced methods.

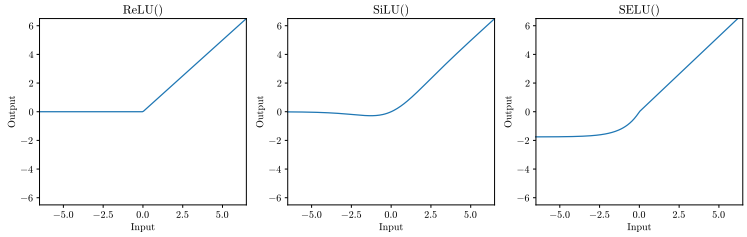

Figure 6.1: The plot shows the different activation functions we use in the neural networks.

6.1.1 Model Architectures

We start by explaining the model architectures we chose and how to set the hyperparameters.

Action Matching. For Action Matching we parametrize the action $s : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}$ by a neural network s_θ. The parametrized vector field from (2.1.1) is then given by $u_\theta = \nabla_x s_\theta$. As the action is time dependent we model it with a neural network with $d+1$ input nodes and one output node. We use 4 hidden layers in the network with 512 hidden nodes per layer. This is the same choice the authors used for Action Matching.[1] In our experiments using less hidden nodes resulted in worse predictions. The Action Matching model has especially problems to capture curvature. We tried to use more layers in the neural network to improve this. However, we could see no significant change in performance. Therefore, we use 4 layers. The activation functions we use depend on the task at hand. The first architecture (MLP ReLU) uses only ReLU activation functions, i.e.

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0. \end{cases}$$

The second architecture (MLP SiLU) uses the ReLU activation function after the first layer and the SiLU activation function after the second and third layer, i.e.

$$f(x) = x * \sigma(x),$$

where $\sigma(x)$ is the logistic sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Figure 6.1 shows both the ReLU and the SiLU activation functions. MLP SiLU is the same architecture the authors used in [Nek+23b].

The MLP ReLU architecture has difficulties with learning more complex dynamics, however, it protects us from exploding gradients. The MLP SiLU architecture allows to learn more flexible dynamics. On the down side it is very prone to exploding gradients, especially when we discretize

[1] For the code see: https://github.com/necludov/jam/tree/main, last accessed 05.09.2024

time in Action Matching. As the loss function depends on the gradient of the neural network, the exploding gradients problem could not be solved by using gradient clipping.

In some experiments we will interpolate between an initial distribution μ and a target distribution ν without any information on the path which connects those two. We want to introduce I-AM, which uses the conditional trajectories from I-CFM to learn the Action Matching loss. I-AM thus assumes conditional Optimal Transport, similar to I-CFM, meaning for known x_0 and x_1 we transport $\mathcal{N}(x_0, \sigma)$ optimally to $\mathcal{N}(x_1, \sigma)$ by

$$p_t(x \mid x_0, x_1) = \mathcal{N}(x \mid tx_1 + (1-t)x_0, \sigma^2).$$

The choice of σ is explained later. The interpolation between data points is simple and linear. However, it ensures conditional Optimal Transport. While one could use a nonlinear interpolation, this would result in a higher displacement cost at the particle level. Therefore, we choose to consider only the linear interpolation. However, Neklyudov et al. explore the impact of learning the interpolation path using a neural network in [Nek+23a]. To mix the conditional sample paths we use the independent coupling. One could also think about using another measure here.

Entropic Action Matching. For Entropic Action Matching (EAM) we use the same network setup. In the loss function we use the Hutchinson trace estimator (see Section 5.2.1) to estimate the Laplacian of the action. Experimentally we could see that using $n = 3$ trials for this estimator yields good results without blowing up the computational time too much.

We denote by I-EAM the entropic variant of I-AM for the cases where we only have access to the initial and target distribution.

If not otherwise indicated we use $g(t) = 0.25$. This is the same value which is used in [Ton+23] for $[SF]^2M$.

Conditional Flow Matching. For Conditional Flow Matching, we base the network architecture on the one used by the authors [Ton+24][1]. We need to parametrize the vector field $u : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}^d$. Thus, we use a network with $d + 1$ input nodes and d output nodes. We again use a network with 4 hidden layers and 512 hidden nodes per layers. While CFM already shows good performance with less hidden nodes per layer, we wanted the architecture to be comparable to the Action Matching architecture. Tong et. al. [Ton+24], however, only use 64 hidden nodes. As activation function we use SELU which is also used by [Ton+24]:

$$f(x) = \begin{cases} \text{scale} * \alpha * (\exp(x) - 1) & \text{for } x < 0 \\ \text{scale} * x & \text{for } x \geq 0. \end{cases}$$

[1]The code to this paper can be found on github: https://github.com/atong01/conditional-flow-matching, last accessed 05.09.2024

Here α and scale are predefined constants. A plot of the SELU activation function can be found in Figure 6.1.

To calculate the Optimal Transport coupling in OT-CFM and the entropic Optimal Transport coupling in SB-CFM we use a minibatch approximation as described in Section 3.3. We will also use the unbalanced Optimal Transport minibatch approximation (UMOT) for OT-CFM which is described in Section 3.3. We will refer to this method as UOT-CFM. The batch size to calculate the minibatch approximation will be chosen later.

For SB-CFM we choose $\sigma = 1$ in the marginal conditional probability path $p_t(x \mid x_0, x_1)$, which is the value used in [Ton+24]. For I-CFM and OT-CFM we choose σ later.

$[SF]^2M$ For $[SF]^2M$ we need to parametrize both the vector field $u : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}^d$ and the score function $\nabla \log p : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}^d$ by neural networks. We parametrize the vector field by v_θ and the score function by s_θ. For both we use the same architecture as for Conditional Flow Matching.

We use SB-CFM to match the vector field. As before we choose $\sigma = 1$ in the conditional probability path. We use the same diffusion coefficient $g(t) = 0.25$ as the authors [Ton+23]. Further, in Chapter 7, we fine-tune $g(t)$ only as a constant, rather than as a time-dependent function, for the single-cell data. We find that $g(t) = 0.25$ gives optimal results for the embroyoid body data.

Optimizer

For all methods except $[SF]^2M$ we use the optimizer Adam [KB15]. We experimentially tested how the methods perform for different learning rates. In Figure 6.2 we plot an example of the fine-tuning process in which we use the two dimensional toy data from Section 6.2. We see the W_2-distance between the observed and predicted data for I-AM, I-CFM and OT-CFM. Even though I-AM behaves slightly better for a higher learning rate, we use a learning rate of 10^{-4}. Only for unrestricted Action Matching, i.e. the original Action Matching which requires access to the whole probability path $(p_t)_t$, we use a learning rate of 0.0002, as this has empirically proven to be more effective.

For $[SF]^2M$ we use AdamW with a learning rate of 0.0001 as an optimizer. This is the same optimizer which Tong et. al. use [Ton+23]. Similiar to Adam, AdamW uses adaptive estimates for the first and second moments of the gradients. However, Adam uses L_2 regularization in its algorithm, while AdamW uses weight decay regularization [LH19].

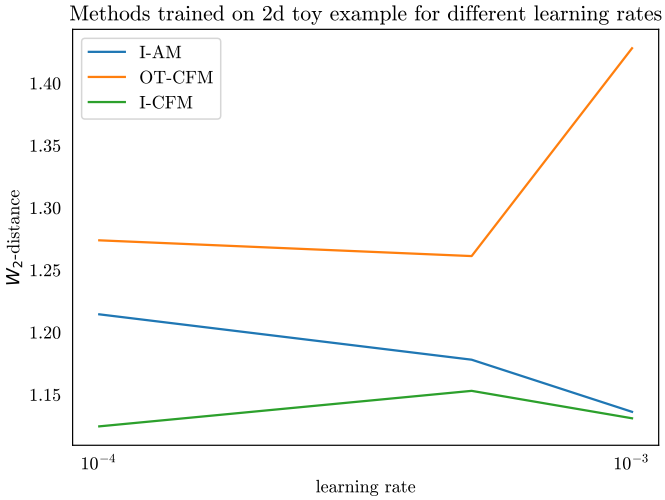

Figure 6.2: We fine-tuned the learning rate based on the methods I-AM, OT-CFM and I-CFM for two dimensional data. The data is the same as we use in Section 6.2. The W_2-distance is averaged over three runs of the methods.

Batch Size and Other Hyperparameters

As the next step, we tune the batch size, which determines the number of training samples processed simultaneously before the model's parameters are updated. We will calculate the minibatch Optimal Transport approximation for OT-CFM, UOT-CFM, SB-CFM, and $[SF]^2M$ on each batch of training data. Thus, the choice of batch size is particularly critical. A batch size that is too small can lead to a inaccurate minibatch approximation, while a batch size that is too large increases computational time

We focus on fine-tuning the parameter for OT-CFM, as it is the most critical method. As previously discussed, calculating the Optimal Transport coupling has a time complexity of $O(n^3 \log(n))$.

Figure 6.3 (right) illustrates the relationship between the W_2 error and computational time (in seconds) as a function of batch size. The data used in this example is the five dimensional dataset from Section 6.4. As expected, the W_2 error consistently decreases as the batch size increases. This is because a more accurate minibatch OT coupling better approximates the dynamic OT solution obtained after solving OT-CFM, thereby minimizing the W_2 error. Surprisingly, the computational time is slightly higher for a batch size of 64 compared to 128 due to the increased number of iterations required to process the data. Afterwards, computational time increases consistently as expected.

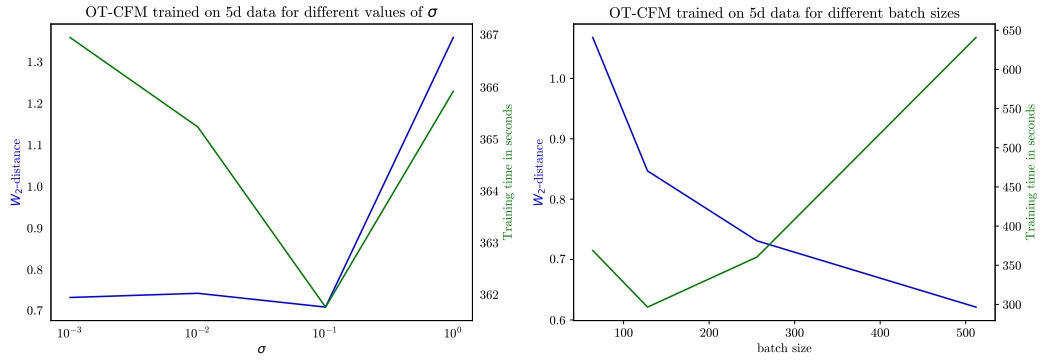

Figure 6.3: We fine-tune σ and the batch size for OT-CFM. The plot shows the results on the 5 dimensional data from Section 6.4 (Gaussian to two Gaussians). The blue line plots the W_2 error while the green line plots the computational time in seconds. Both metrics are averaged over three runs of OT-CFM.

In our experiments, we selected a batch size of 256, which balances a relatively low W_2 error with reasonable computational time. This batch size will be used for all methods except UOT-CFM. As discussed in Section 3.3, UOT-CFM performs well with a significantly smaller batch size compared to OT-CFM. Yet, its computational time for the same batch size is higher. For UOT-CFM we use a batch size of 50. This is higher than in Section 3.3 to account for more complex data.

Finally, we fine-tuned the parameter σ for the conditional probability paths in OT-CFM, I-CFM, I-AM and I-EAM, specifically $p_t(x \mid x_0, x_1) = \mathcal{N}(x \mid tx_1 + (1-t)x_0, \sigma^2)$. Figure 6.3 shows an example of fine-tuning OT-CFM using the same data as used for determining the batch size. A value of $\sigma = 0.1$ appears optimal for both the W_2 error and computational time. This is the same parameter value used in [Ton+24]. We will use $\sigma = 0.1$ for OT-CFM and I-CFM, as well as for I-AM and I-EAM.

6.1.2 Solving the ODEs or SDEs

Our methods only learn the drift of an ODE or SDE. In order to access the learned distribution at a time point t, we have to solve the ODE or SDE. We use the Euler scheme for the ODE (see Section 2.1.1) and the Euler-Maruyama scheme for the SDE (see Section 5.1.3). For the Euler scheme we choose a step width of 0.01, while for the Euler-Maruyama scheme we choose a step width of 0.0001 as this method has only a strong convergence rate of $\frac{1}{2}$.

6.1.3 Performance Metrics

In our examples we want to measure how well the true distribution at a time point t is predicted from the intial data at time point t_0. That is we want to measure the difference or distance between the true or observed distribution and the predicted distribution. One quantity to measure this difference is the Kullback Leibler divergence (see Section 2.2, (2.2.5)), which is, however, not a distance. Further it requires the distributions to have joint supports, otherwise it always takes the value ∞ and does not allow for any comparisons. Optimal Transport allows us to define another notion of difference. The p-Wasserstein distance is a metric. In the following we will use either the 1-Wasserstein or 2-Wasserstein metrics (see Section 2.2). We saw in Section 2.2 that approximating the Optimal Transport loss of two distributions by samples suffers of a curse of dimensionality. This problem can be mitigated by using an entropy regularized version of Optimal Transport. We tried to use both the Wasserstein and the entropy regularized Wasserstein distance. In our experiments the entropy regularized Wasserstein distance was consistently slightly higher than the Wasserstein distance. The qualitative results of the experiments, however, were the same. Thus, we only display the unregularized Wasserstein distance in the evaluation of our experiments. As it is computationally too complex to calculate the exact Wasserstein distance we calculate the Wasserstein distance on many minibatches over which we average. Here we choose the minibatch size to be 256.

Additionally, we visually examine plots of the learned trajectories in two dimensions. Interestingly, the models that perform best in terms of Wasserstein distance do not always produce the most visually convincing trajectories. One possible explanation is that Optimal Transport can be heavily influenced by outliers. Learning the outliers reduces the transport distance required, but this may lead to suboptimal trajectories. To address this, we considered using unbalanced Optimal Transport (Section 2.2) as an evaluation metric, which is more robust to outliers. However, using unbalanced Optimal Transport would require fine-tuning the parameter τ, which determines how far we are willing to transport mass, involving significant computational effort. Moreover, identifying outliers is not always straightforward. Therefore, we decided to use balanced Optimal Transport for the evaluations.

6.2 Toy Examples in 2d

In this section we want to interpolate between an initial and a target distribution in two dimensions and observe what paths are learned. As initial distribution we take the moons distribution and as target distribution we take the distribution of 8gaussians. In Figure 6.4 we can see samples of the initial and target distributions. We want to compare the following methods: I-CFM, OT-CFM, UOT-CFM, SB-CFM, $[SF]^2M$, I-AM and I-EAM. We let all of the methods run for 1000 iterations, that is we train each model for 1000 epochs on the training set of 10000 samples of the initial and target distribution. In each epoch we shuffle the data so we have different

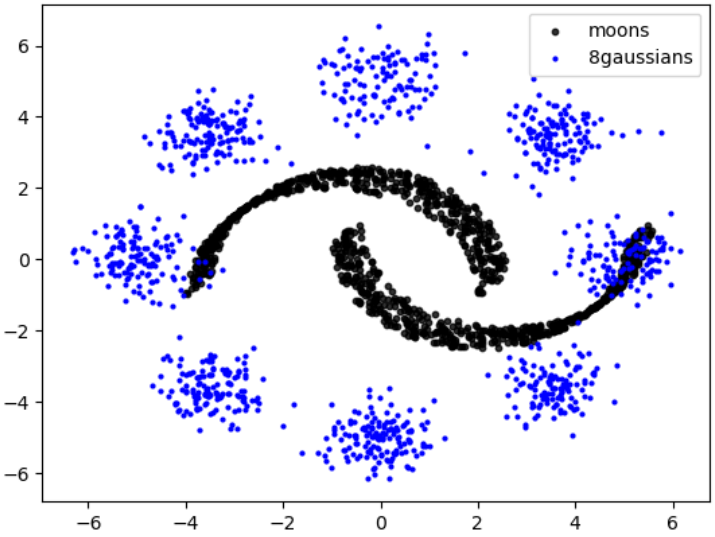

Figure 6.4: We choose the initial distribution μ as moons and the target distribution ν as 8gaussians

pairs of initial and target samples. In Figure 6.5 we can see the learned trajectories and target distributions of the data. For the stochastic methods we colored a few learned paths in red as the other paths are not well distingushable. It is desirable that the predicted target distribution of 8gaussians looks roughly like the true distribution in Figure 6.4. Further, we prefer straight probability paths as they can be learned more easily with the Euler scheme. As for qualitative measures, Table 6.2 shows the W_2-distance between the real and predicted target distribution and the computational time of each method is shown in Table 6.3.

Deterministic Flow and Action Matching Examining Figure 6.5, we observe that OT-CFM, UOT-CFM, and SB-CFM generate very straight paths, while I-CFM produces curved paths. The paths learned by I-AM are similar to those of I-CFM but appear slightly straighter. The 8gaussians distribution predicted by OT-CFM, UOT-CFM and SB-CFM shows some inconsistencies, such as holes in the distribution. For example some points are predicted to be on the moon distribution instead of being transported to their corresponding Gaussian centers. Visually, the target distribution produced by I-AM appears more convincing, with fewer holes and rounder distribution centers that more closely resemble a Gaussian distribution. Although the prediction of I-CFM reflects the actual data structure, it is too noisy.

Despite this, Table 6.2 shows that OT-CFM and SB-CFM perform best, while I-AM and UOT-CFM perform significantly worse, with I-CFM performing the worst. Considering the computational times in Table 6.3, OT-CFM emerges as a good choice for W_2 performance, and I-AM for visual performance. UOT-CFM is the slowest method among the deterministic methods, even

	moons → 8gaussians
I-CFM	1.8±0.36
OT-CFM	1.65±0.47
UOT-CFM	1.78±0.44
SB-CFM	1.66±0.46
$[SF]^2M$	1.68±0.45
I-AM	1.77±0.38
I-EAM	1.74±0.48

Table 6.2: W2-distance of predicted target distribution vs the true distribution 8gaussians. Red indicates the best perormance and blue the second best.

with a batch size of 50 compared to 256 for the other methods. When comparing W_2-distances and computational time, UOT-CFM is less advantageous than OT-CFM.

Stochastic vs. Deterministic We have two methods that model stochastic dynamics: $[SF]^2M$ and I-EAM. Both stochastic methods visually produce good predictions of the target distribution. Unlike the deterministic methods, the predicted clusters look very round and close to Gaussian distributions. However, the Gaussians predicted by $[SF]^2M$ seem to have a lower variance than the 8gaussians distribution, while the predictions of I-EAM accurately reflect the variance. Due to their stochastic nature, these methods produce noisy paths, with the noisiness depending on the diffusion coefficient $g(t)$. However, the sample paths of I-EAM sometimes bend undesirably. While $[SF]^2M$ performs slightly worse than SB-CFM for W_2-distance, I-EAM slightly outperforms its deterministic counterpart. In terms of W_2-distance and computational time, $[SF]^2M$ outperforms I-EAM, which has high computational complexity due to the Hutchinson trace operator with $n = 3$ estimators.

Remark 6.2.1. *As mentioned in Section 6.1.3, the most visually convincing model is not necessarily the best performing in terms of W_2-distance. While I-EAM and $[SF]^2M$ produced visually appealing predictions, their W_2 performance metrics were worse than those of OT-CFM or SB-CFM.*

Conclusion Visually the stochastic methods outperform the deterministic ones. However, they are more complex to train and as we need to apply the Euler-Maruyama scheme to solve the SDE, they are also computationally more complex to solve. In terms of W_2-distance CFM methods like OT-CFM and $[SF]^2M$ can outperform the methods using the Action Matching loss.

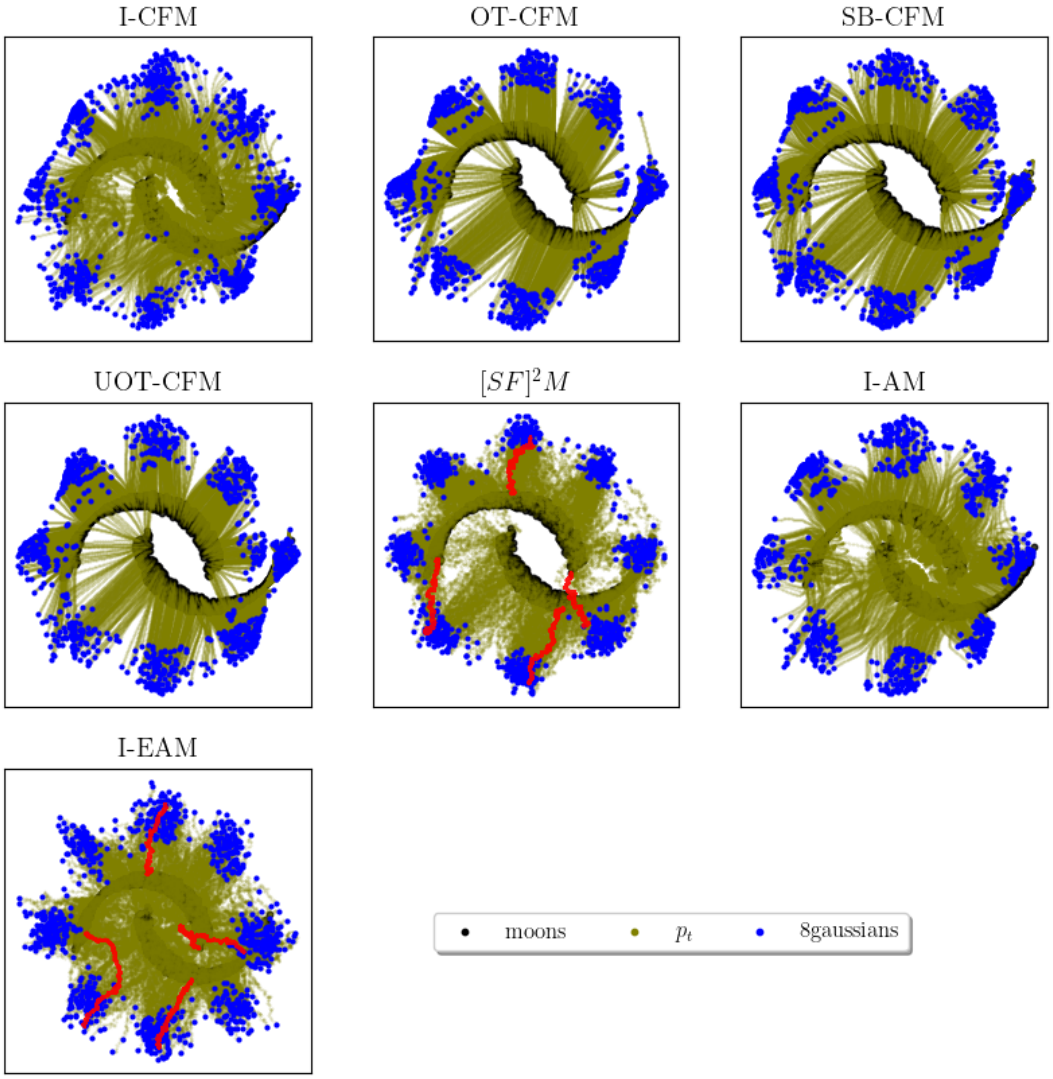

Figure 6.5: The learned paths of each of the methods. Here moons is the initial distribution. The blue dots are the predicted samples of the target distribution using the respective method. The green line indicates the learned path between the two distributions.

	moons → 8gaussians
I-CFM	258.24±4.57
OT-CFM	576.11±7.67
UOT-CFM	7820.36±2209.49
SB-CFM	576.31±8.1
$[SF]^2M$	779.5±29.03
I-AM	684.19±29.81
I-EAM	10572.3±2185.81

Table 6.3: Computational time in seconds needed to train the models

6.3 Learning Dynamics with Multiple Time Points

The concept of Action Matching is based on full knowledge of the trajectories. While the authors of the method argue that even with a finite number of time steps a satisfactory convergence to the true action can be achieved, this claim should be verified. We attempt to analyse this assertion on the basis of the development of a Gaussian distribution with a bend. Let us define the two-dimensional dynamics over the interval $t \in [0, 3]$, by $p_t(x) = \mathcal{N}(x \mid \mu_t; 0.1 \cdot I_2)$, where $I_2 \in \mathbb{R}^{2\times 2}$ is the identity matrix and

$$\mu_t = \begin{cases} (5t, 5t)^T & \text{for } t \leq 1 \\ (5t, 10 - 5t)^T & \text{for } t > 1. \end{cases}$$

In Figure 6.6 we see how different models predict the dynamics. OT-CFM with 4 time points is a very good approximation of the true dynamics. We use this dynamics as it is fairly simple but still has one bend which we want to capture in the predictions. Theoretically, we restore the exact dynamics if we have access to data at time points 0, 1 and 3 and just connect the distributions at those time points by straight lines. We will investigate what happens if we add data at more time points than necessary. Do the predictions stay good or do they get more noisy?

Unrestricted Action Matching We begin by training with unrestricted Action Matching, which utilizes the entire trajectory in the training process. Using an MLP SiLU architecture with ReLU and SiLU activation functions and a learning rate of 0.0002, this method effectively learns the dynamics. When tested at time $t = 2$, the model achieves an average W_2-distance of 0.44 with a standard deviation of 0.32 across three runs. However, comparing it to OT-CFM and I-CFM in Table 6.4 models using only three time points can outperform Action Matching. Notably, unrestricted Action Matching slightly overshoots the target distribution, as illustrated in Figure A.1 in the Appendix.

Discretizing Time

In practical scenarios, we typically have access only to a discretized version of the entire trajectory. Thus, it is essential to evaluate our methods with varying numbers of time points. To assess model performance, we exclude a small time interval $(1.9, 2.1)$ from training and then evaluate the models at time point $t = 2$. The remaining time points used for training are evenly spaced across the interval $[0, 3]$.

Unconstrained Action Matching Although it is suggested that Action Matching works well with time discretization, we encountered significant challenges in practice. Using the same MLP-SiLU architecture and learning rate of 0.0001, the model experienced instability when discretizing the loss. This behavior was evident even with a considerable number of time points, stabilizing only when using 50 to 100 time points. However, even then the predicted dynamics did not really represent the actual dynamics. A plot of the learned trajectores can be found in Figure A.2 in the Appendix. Gradient clipping did not mitigate the issue due to the dependence of the loss on the model's gradient.

To prevent model instability, we employed the MLP ReLU architecture, which exclusively uses ReLU activation functions. However, this architecture failed to capture the dynamics, resulting in poor performance. A plot of the learned trajectores can be found in Figure A.3 in the Appendix.

Consequently, we cannot confirm that Action Matching performs well on discretized data. To explore whether the loss function offers any advantages, we compared the performance of I-CFM, OT-CFM, and I-AM.

I-CFM, OT-CFM and I-AM To evaluate these methods, we trained each on datasets containing 2 (initial and target data only), 3, 4, 9, 10, and 15 time points. The time point $t = 1$, where the distribution curve bends, is included in the datasets with 4 and 10 time points. The 9-point dataset was chosen specifically because it includes time points at $t = 1.875$ and $t = 2.25$, which are close to the test time at $t = 2$. Additionally, training with 15 time points allowed us to observe how the models perform when provided with more data than necessary. Table 6.4 clearly shows that both I-CFM and OT-CFM outperform I-AM. This is also evident from the learned probability paths in Figure 6.6, where some paths, diverge significantly from the others for I-AM. In contrast, the paths learned by I-CFM and OT-CFM are quite similar. To accurately learn the dynamics we need to have access to the time points at which curvature appears. For instance, using only three time points in training, the model learns the bend at $t = 1.5$ as we can see in Figure 6.6. With four and ten time points, the time point $t = 1$ at which the curvature occurs is included in the training data, allowing for effective learning of the dynamics.

However, Table 6.4 indicates that increasing the number of time points does not necessarily yield better results. Only three time points are needed to effectively predict the test data, while

four are sufficient to capture the dynamics. Adding more time points can introduce extra noise, potentially degrading performance. Therefore, it is optimal to use the minimum number of time points necessary to capture the dynamics. Interestingly, using nine time points yields very good results, despite the relatively high number. This may be because the dataset includes two time points close to $t = 2$.

While OT-CFM performs best overall, Figure 6.7 shows that its computational complexity increases significantly with more time points, unlike I-CFM and I-AM, which exhibit only slight increases. With too many time points it could become infeasabile to calculate the OT-CFM model. This slow computational time results from the fact that OT-CFM has to calculate the minibatch Optimal Transport couplinng for each pair of consecutive time points during training.

Remark about I-AM I-AM appears to be very sensitive to data irregularities. It is applied in a discrete setting, where we interpolate the distributions between time steps to stabilize the models. However, time steps are often unevenly spaced. If the number of interpolated data points is not proportional to the distance between consecutive time points, the I-AM model produces poor results and tends to become unstable. To address this, we ensured that the interpolated samples are proportional to the time distances.

Despite this adjustment, practical data may be more densely available in some time regions than in others, making the Action Matching loss unsuitable in such cases. Conversely, the Flow Matching models did not exhibit this problem and appear to be more robust.

Conclusion In conclusion, Action Matching demonstrates limited scalability in learning dynamics with multiple time points. Curvature in particular is a major problem, as it cannot be learnt well with ReLU activation functions alone. However, the use of other activation functions makes the model very vulnerable to exploding gradients. Also the simpler Flow Matching loss outperforms the Action Matching loss, as evidenced by the performance difference between I-CFM and I-AM. Although OT-CFM is computationally intensive, it is better suited to the task. For scenarios with many time points, I-CFM might be preferable in terms of computational efficiency. While both OT-CFM and I-CFM are sensitive to noise of excessive time points, which are not needed to describe the dynamics, the effect is stronger for I-CFM. If it is computationally feasabile OT-CFM should be preferred.

6.4 Higher-Dimensional Experiments

This subsection investigates how our proposed methods behave in settings with increasing dimensionality. We are particularly interested in the following:

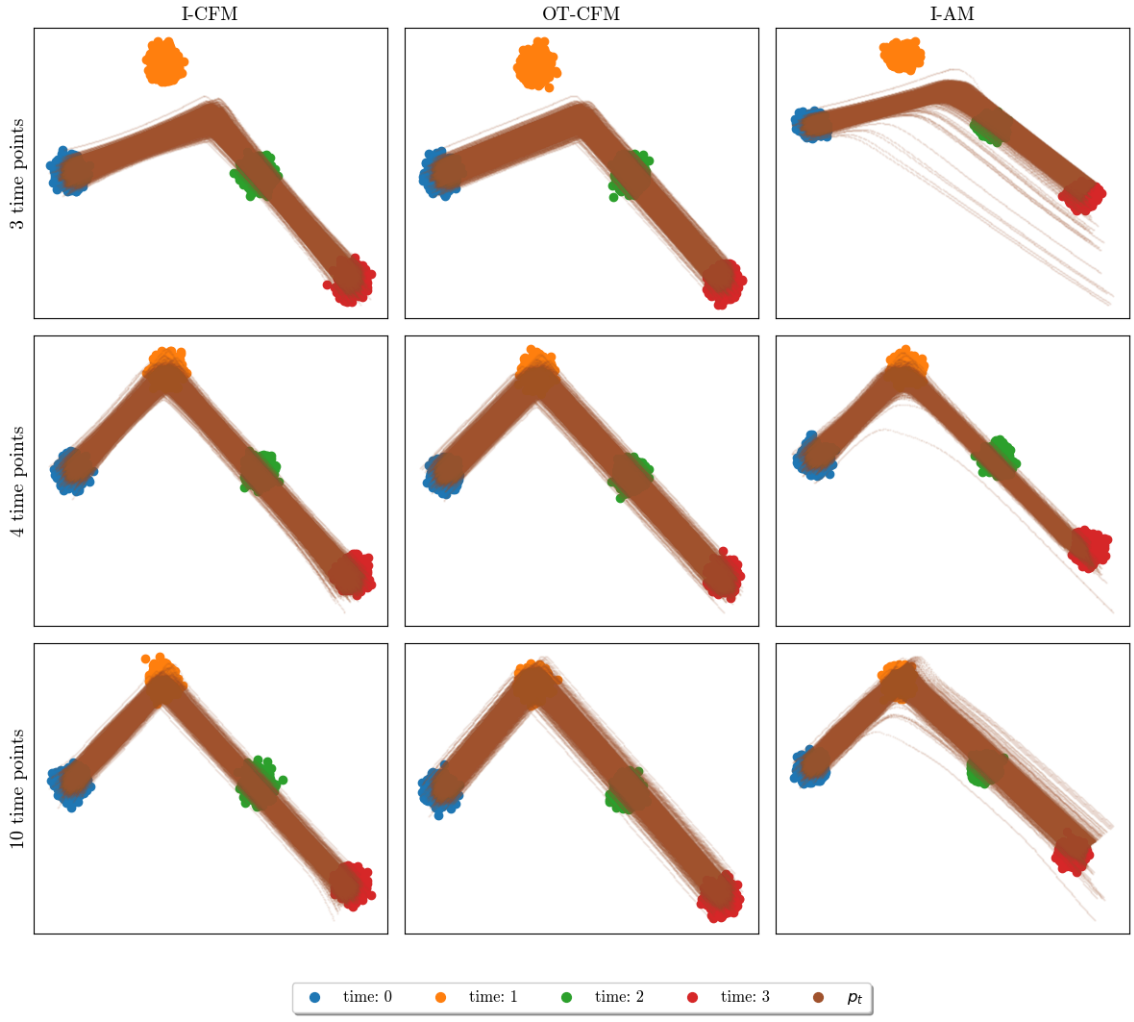

Figure 6.6: We evaluate the models' trajectories based on varying numbers of training time points. The four colored scatter clusters correspond to the actual data distribution at time points 1, 2, 3, and 4. The brownish path represents the trajectory learned by the model.

Number of time points	I-CFM	OT-CFM	I-AM
2	3.3±0.02	3.35±0.07	3.35±0.01
3	0.21±0.1	0.21±0.09	0.74±0.13
4	0.24±0.02	0.2±0.07	0.54±0.1
9	0.24±0.09	0.17±0.01	0.37±0.03
10	0.27±0.07	0.28±0.09	0.43±0.15
15	0.35±0.12	0.2±0.1	0.38±0.11

Table 6.4: W_2-distance of OT-CFM and I-AM tested on left out time point $t = 2$ using an increasing number of time points during training.

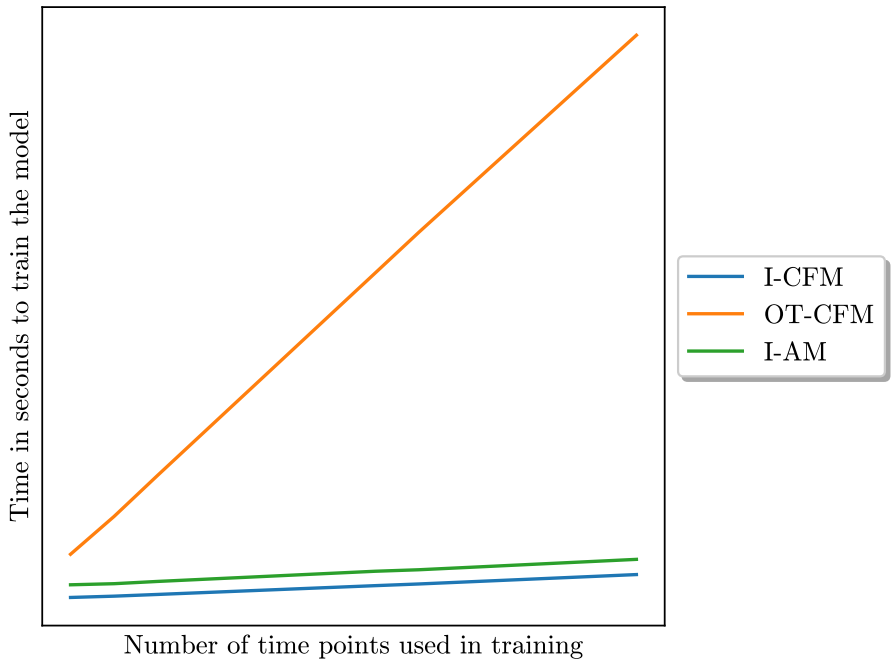

Figure 6.7: Computational time in seconds needed to train the models for 10000 iterations. In each iteration we trained on a batch size of 256.

- **Impact of Dimensionality on Performance:** Does the implicit loss of Action Matching (AM) differ from the mean squared loss of CFM? Do stochastic methods lead to better performance in high dimensions?
- **Computational Complexity:** How does the computational complexity of the methods scales in high dimensions?
- **Behavior on Different Data Types:** How do the methods perform on truly high-dimensional data versus data lying on a low-dimensional manifold?

We interpolate between an initial distribution μ and a target distribution ν. Since we thus lack pre-defined trajectories for Action Matching, we use I-AM and I-EAM (see Section 6.1). We compare those methods to OT-CFM and $[SF]^2M$. As OT-CFM consistently showed better results than I-CFM, we only show the results for OT-CFM here.

6.4.1 Truly Higher Dimensional Data

In our first example we want to explore the performance on truly high dimensional data, i.e. data which cannot be projected on a lower dimensional manifold without significant information loss. For this we choose μ as a standard Gaussian distribution. As target distribution ν we use a Gaussian mixture consisting of $\nu_1 = \mathcal{N}(\boldsymbol{\mu_1}, 0.25I)$ and $\nu_2 = \mathcal{N}(\boldsymbol{\mu_2}, 0.25I)$ with $\boldsymbol{\mu_1} = (-1, \dots, -1)^T$ and $\boldsymbol{\mu_2} = (1, \dots, 1)^T$. This creates a high-dimensional dataset with simple features. Figure 6.8 shows μ and ν in two dimensions. We evaluate the methods in dimensions 2, 5, 10, 50, 100, 500 and 1000.

Benefits of Stochasticity: One motivation for introducing stochastic variants (EAM and $[SF]^2M$) was the potential for better performance in higher dimensions due to enhanced exploration. Table 6.5 confirms this hypothesis for $[SF]^2M$ only. In high dimensions $[SF]^2M$ can consistently outperform the deterministic Flow Matching method OT-CFM, even without fine-tuning the hyperparameter $g(t)$ specifically for this task. I-EAM on the other hand cannot outperform I-AM but rather performs similar in high dimensions.

Flow Matching vs. Action Matching: Another observation is that Flow Matching outperforms Action Matching in this example across all dimensions.

Computational Time: Table 6.6 compares the computational time of each method. Entropic Action Matching is the most expensive due to the use of the Hutchinson trace estimator with $n = 3$ trials for estimating the model's Laplacian. However, its time remains relatively stable

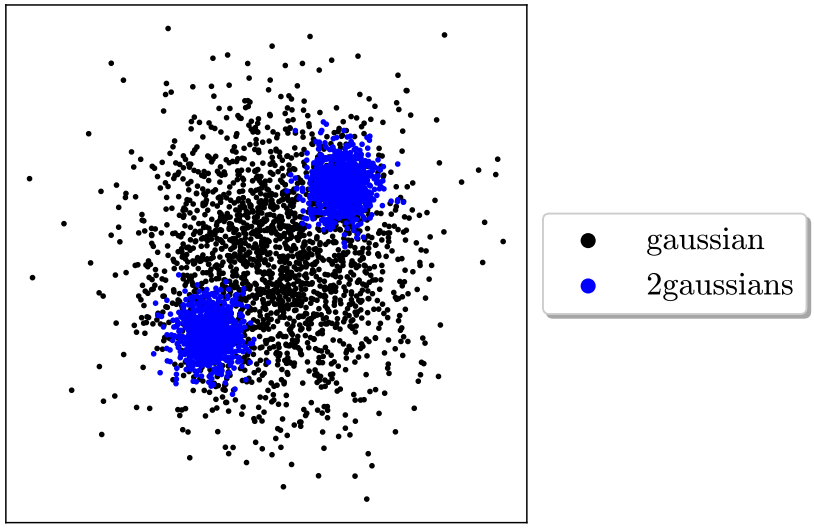

Figure 6.8: We choose the initial distribution μ as a Gaussian and the target distribution ν as the mixture of two Gaussians.

across dimensions. Action Matching also exhibits stable complexity. Both OT-CFM and $[SF]^2M$ show increasing complexity with dimensionality. This is especially true for $[SF]^2M$, which becomes significantly slower in dimensions 500 and 1000 due to fitting two models simultaneously.

6.4.2 Data on a Low-Dimensional Manifold

This section explores how our methods perform when data resides in a high-dimensional space but has a much lower intrinsic dimensionality. That means the data lies on a low-dimensional manifold with some noise. This scenario is common in practice and the single cell data we explore in Chapter 7 is an example thereof.

Experimental Setup: We assume the intrinsic dimension of the data is two. For this we consider the initial distribution 8gaussians and the target distribution moons. We used those two distributions already in Section 6.2 and their distribution is displayed in Figure 6.4. The first two dimensions of the data represent these distributions, while additional dimensions are filled either with Gaussian noise with variance 0.1 or are simply dummy dimensions which are set to zero. This essentially embeds the original 2 dimensional data in a higher-dimensional space (5, 10, 50, 100, and 500 dimensions). We repeat each experiment three times.

Dimension	OT-CFM	$[SF]^2M$	I-AM	I-EAM
2	0.36±0.02	0.47±0.03	0.37±0.01	0.64±0.25
5	0.7±0.02	0.83±0.07	0.75±0.04	1.04±0.35
10	1.25±0.08	1.24±0.04	1.24±0.06	1.41±0.15
50	3.25±0.08	2.96±0.09	3.19±0.12	3.57±0.91
100	4.72±0.15	4.16±0.1	4.99±0.28	4.77±0.15
500	13.78±0.32	10.7±0.15	14.88±0.57	14.88±0.34
1000	27.63±0.26	21.61±0.29	28.46±0.27	28.52±0.26

Table 6.5: We compare the approximated W_2-distance between the predicted target distribution and the true target distribution over different dimensions. Here we used a standard Gaussian as initial and a mixed Gaussian as target distribution. The results for each method were averaged over three runs, with the standard deviation also shown. The best result for each dimension is highlighted in red while the second best is highlighted in blue.

Dimension	OT-CFM	$[SF]^2M$	I-AM	I-EAM
2	322.41±2.71	360.61±4.64	170.24±4.42	813.28±21.02
5	344.01±1.1	383.87±0.33	170.66±0.38	825.24±0.59
10	351.72±2.44	391.85±1.81	170.54±0.78	824.28±2.33
50	357.49±1.19	398.15±1.57	171.36±0.14	830.39±0.76
100	358.19±0.89	398.46±0.61	171.91±0.48	830.8±2.87
500	360.16±2.91	435.06±1.19	171.25±0.18	831.49±4.36
1000	363.66±1.28	495.18±1.17	172.48±0.11	832.4±1.37

Table 6.6: We compare the time in seconds the methods needed to train over different dimensions. Here we used a standard Gaussian as initial and a mixed Gaussian as target distribution.

Performance Evaluation: We measure performance by approximating the W_2-distance between the predicted target distribution and the true target distribution. We use a validation set of 10,000 samples. To isolate the performance on the underlying manifold, we calculate the W_2-distance between the true and predicted two dimensional distributions. Specifically, we project the data onto its first two dimensions and calculate the error based only on this projection.

Impact of Noise and Dummy Dimensions: Tables 6.8 and 6.7 compare performance across dimensions. We can observe the following:

- Regardless of noise or dummy dimensions, deterministic methods seem as good as or better than their stochastic counterparts.
- Action Matching is robust to noise dimensions and its performance is not becoming significantly worse in high dimensions. However, it performs poorly with dummy dimensions.
- Action Matching displays a higher variance in its performance across multiple runs.
- OT-CFM can outperform all other methods with dummy dimensions. However, it is more sensitive if we use noise instead of dummy dimensions.

Higher-Dimensional Embedding of 50 Dimensional Data: In practice the intrinsic dimension of the data is usually higher than two. Thus we also embed 50 dimensional data into 100 and 500 dimensional spaces. The intrinsic data is the same we used in the truly high-dimensional experiments. The other dimensions are filled with gaussian noise with variance 0.1. To evaluate we calculate the minibatch W_2 approximation between the true and predicted 50 dimensional distributions. Table 6.9 compares the performance of the methods. We observe that $[SF]^2M$ performs better in 100 dimensional space, while OT-CFM performs better in 500 dimensional space. This aligns with our observations in Table 6.5, where $[SF]^2M$ outperformed OT-CFM in 50 dimensions. However, as the number of noise dimensions increases, $[SF]^2M$ becomes less robust. Stochastic methods inherently explore all dimensions, including those containing noise. This focus on irrelevant information leads to increased error in higher-dimensional settings.

Further, entropic Action Matching performs surprisingly well in this setting, while I-AM performs worst in 500 dimensions.

Conclusion: Deterministic versus Stochastic Methods

To conclude this chapter we can summarize our observations:

- In truly high-dimensional data, $[SF]^2M$ outperforms all other methods.

Dimension	OT-CFM	$[SF]^2M$	I-AM	I-EAM
2	0.6±0.02	0.66±0.03	0.67±0.08	0.72±0.07
5	0.59±0.02	0.66±0.01	0.62±0.01	0.67±0.02
10	0.62±0.0	0.68±0.03	0.68±0.06	0.61±0.0
50	0.66±0.02	0.74±0.0	0.67±0.12	0.68±0.05
100	0.72±0.01	0.77±0.02	0.66±0.04	0.75±0.11
500	0.76±0.03	0.77±0.04	0.68±0.05	0.84±0.03

Table 6.7: We compare the approximated W_2-distance between the predicted target distribution and the true target distribution over different dimensions. In this example the data lies on a 2-dimensional manifold where all other dimensions consist of gaussian noise. The results for each method were averaged over three runs, with the standard deviation also shown. The best result for each dimension is highlighted in red while the second best is highlighted in blue.

Dimension	OT-CFM	$[SF]^2M$	I-AM	I-EAM
2	0.59±0.04	0.64±0.01	0.65±0.06	0.69±0.07
5	0.59±0.02	0.66±0.01	0.67±0.07	0.75±0.13
10	0.58±0.01	0.74±0.07	1.24±0.63	0.72±0.09
50	0.65±0.03	0.74±0.02	5.97±2.23	75.18±42.01
100	0.63±0.02	0.84±0.01	10.35±0.68	21.05±3.33
500	0.64±0.03	0.89±0.03	15.63±5.16	2.46±0.01

Table 6.8: We compare the approximated W_2-distance between the predicted target distribution and the true target distribution over different dimensions. In this example the data lies on a 2-dimensional manifold where all other dimensions are set to 0. The results for each method were averaged over three runs, with the standard deviation also shown. The best result for each dimension is highlighted in red while the second best is highlighted in blue.

Dimension	OT-CFM	$[SF]^2M$	I-AM	I-EAM
100	3.39±0.12	2.94±0.07	3.29±0.05	3.19±0.11
500	3.2±0.09	3.3±0.08	3.37±0.13	3.19±0.16

Table 6.9: We compare the approximated W_2-distance between the predicted target distribution and the true target distribution over different dimensions. In this example the data lies on a 50-dimensional manifold where all other dimensions consist of Gaussian noise. The results for each method were averaged over three runs, with the standard deviation also shown. The best result for each dimension is highlighted in red while the second best is highlighted in blue.

- When learning a lower-dimensional data manifold in a high dimensional space, stochastic methods are not beneficial and may even decrease performance. Action Matching here performs well with noisy dimensions but also displays unstable behaviors (e.g., with dummy dimensions). In general the performance of all methods decreases for additional noise dimensions.

Because of these observations it is desirable to bring the data down or at least closer to its intrinsic dimension. Thus, we recommend using dimensionality reduction techniques such as Principal Component Analysis (PCA) or Diffusion Maps before applying the methods in practice.

We will apply the methods to more complex single-cell data. Here, $[SF]^2M$ is expected to continue performing well while offering the ability to predict different trajectories from the same initial data.

Chapter 7

Single-Cell Data

In modern biology single cell analysis is a crucial technology. Historically, biological experiments relied on ensemble measurements, averaging responses across large populations of cells and thereby masking the heteoregeinity within individual cells. This made it impossible to understand unique behaviors of cells. Single-cell RNA sequencing (scRNA-seq) allows the random sampling of the entire transcriptome - comprising 20 to 30 thousand different mRNA molecules - revealing the complex molecular landscape of individual cells. This wealth of information provides a gateway to understanding fundamental biological processes such as cell differentiation and state transitions. This might help us to understand and combat diseases.

Nevertheless, despite the profound insights that scRNA-seq offers, the technique remains limited by its static nature: Each measurement represents a snapshot in time and excludes the observation of individual trajectories of cells. Instead, cells are destroyed when measured and transribed. Consequently, inferring the trajectories of individual cells from such high-dimensional, static data poses a very complex computational challenge.

To make this more mathematically rigorous let n be the number of cells and d the number of genes. Then the single cell data is given by

$$\mathcal{X} = \{x_i^t \mid i = 1, \dots, n; t \in I\} \subset \mathbb{R}^d,$$

where $I = \{t_1, \cdots, t_T\}$ denotes the different time points at which single-cell measurements were made.

Single-cell data often requires preprocessing. We typically begin by filtering cells based on their library size, which indicates the number of unique mRNA molecules detected in each cell. Extremely small or large library sizes often indicate data errors. Capturing RNA from single cells is inherently noisy, so lowly expressed genes might not be detected. Since these genes are observed in only a few cells, we lack sufficient information about them and, therefore, remove

them.

We can also filter for highly-variable genes (HVGs), which are genes whose expression levels vary significantly across individual cells. These genes may play important roles in cell differentiation, development, disease progression, or response to stimuli.

7.1 A Short History of Trajectory Inference

Trajectory Inference tries to capture the dynamics of cells from their static snapshots. The field is very active and numerous methods have been proposed in recent years. Since static snapshots capture gene expressions at a single point in time, they lack information about the dynamic processes that led to those patterns - often many dynamics could have led to the observed data. This necessitates making assumptions to infer a unique trajectory. By 2019, over 70 methods had been developed to address this challenge [Sae+19].

Most methods begin by reducing data dimensionality using techniques like Principal Component Analysis (PCA) or t-SNE. This aims to capture the variations of gene expressions in a lower-dimensional space. However, methods differ in the assumptions they make about the underlying dynamics and how they represent those dynamics to infer trajectories.

One approach utilizes pseudotime ordering, which assigns a relative time value to each cell based on its gene expression profile. Cells with similar gene expressions are assumed to be at similar stages in the process and are assigned similar pseudotimes. This method requires only one static snapshot. A popular method, Monocle, constructs a minimum spanning tree and identifies the longest path, assigning pseudotime based on this path [Tra+14].

Another set of methods leverages RNA velocity analysis [LM+18], [Ber+20]. RNA velocity estimates the rate of change in gene expression by comparing the abundance of unspliced pre-mRNA and spliced mature mRNA molecules within a single snapshot. Unspliced pre-mRNA cannot yet be translated into a protein, as it contains parts that need to be removed through a process called splicing. The goal is to infer the direction of change in gene expression. It is hoped to infer the direction of change. While appealing due to its single-snapshot requirement, RNA velocity methods have shown limitations in reliability, particularly with noisy data [Zhe+23].

Methods utilizing multiple time points also exist. Waddington-OT [Sch+19] interpolates between empirical distributions at different time points, viewing these distributions as points in Wasserstein space. This primarly focuses on the probability density path and does not offer detailed insights into how individual cells move in space.

Another multi-timepoint method by Hashimoto et al. [HGJ16] assumes a linearly separable potential function and uses a Recurrent Neural Network to sample trajectories.

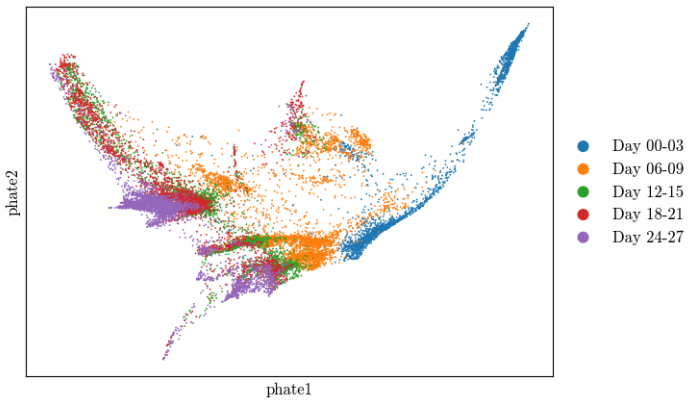

Figure 7.1: The embryoid body dataset projected into two dimensions with the method PHATE [Moo+19].

TrajectoryNet [Ton+20] uses dynamical Optimal Transport to learn the vector field of the ODE which drives the movement of cells in gene space. Yet, the method relaxes the dynamical OT constraint. Instead of requiring the dynamics to perfectly fit the target distribution, it only penalizes deviations from this target, offering more flexibility. However, TrajectoryNet relies on simulating an ODE during training, limiting its ability to handle high-dimensional datasets effectively.

In contrast both Conditional Flow Matching and Action Matching learn this driving vector field in a simulation-free way. Thereby, they describe the movement of individual cells, while being scalable to high-dimensional data.

7.2 Embroyoid Body Data.

We evaluate the methods on a differentiating embroyoid body (EB) scRNA-seq time course. Figure 7.1 shows this data projected into two dimensions using the dimensionality reduction method PHATE [Moo+19]. The data is collected from a developing human embryo system and consists of five time points (Day 0 - Day 24). We took the preprocessed data as in [Ton+20].

For our experiments we first apply a dimensionality reduction method (see below) and then normalize the data. To evaluate the performance we train the methods on all but one intermediate time points and then calculate the 1-Wasserstein distance between the validation data and the model's predictions. We take the W_1-distance here as it is the standard in the literature. An overview over the Wasserstein distance and Optimal Transport can be found in Section 2.2. We also run the methods on all data points and then predict the last data point. We test the methods I-CFM, OT-CFM, UOT-CFM, $[SF]^2M$, AM, EAM and I-AM. The results for I-EAM can be

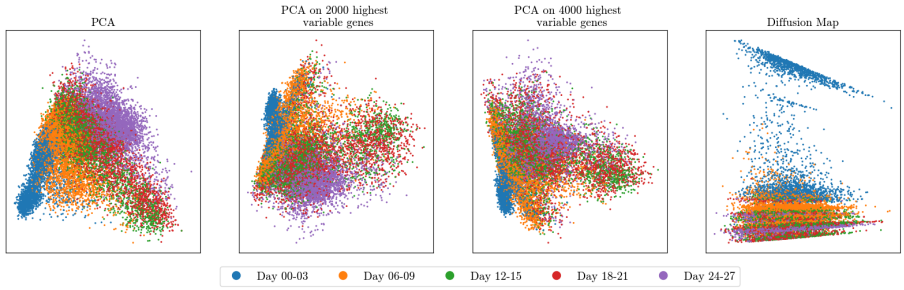

Figure 7.2: The EB dataset projected into two dimensions using either PCA on all genes, PCA on highly variable genes or Diffusion Maps on all genes.

found in Appendix B.

7.3 Dimensionality Reduction Method

While there is a variety of dimensionality reduction methods, we wanted to use one which is iterative in the sense that adding an additional dimension leaves the other dimensions unchanged. Two methods which fulfill this requirement are Prinicipal Component Analysis (PCA) and Diffusion Maps. In Figure 7.2 we can see the two dimensional projections of those two methods. We use the PCA decomposition because it is a common approach in scRNA-seq data analysis. scRNA-seq data is sparse and very noisy. Especially the technical noise cannot be explained in a low dimensional space. If we apply PCA on the whole gene space it is hard to explain a lot of variance. In our case the most important principal component only explains around 2.8% variance, while the second component only explains 1.3%. Even the first 500 principal components together only explain around 33% of variance. Thus it is desirable to include a feature selection step before applying PCA. For this we want to filter for highly variable genes (hvg). Those genes tend to capture the majority of variability in the system, e.g. differences between different cell types. On the other hand we reduce the noise in the data massively. Figure 7.3 shows the variance captured when applying PCA onto the 2000 or 4000 highest variable genes. We also compare it with the case where we did not use feature selection.

For the experiments we will use PCA on the 4000 highest variable genes.

7.4 Experimental Results

We test the methods on 2, 5, 10, 50, 100 and 500 PCA dimensions. For AM we use the MLP ReLU model architecture, whereas for I-AM we use the MLP SiLU model architecture. The

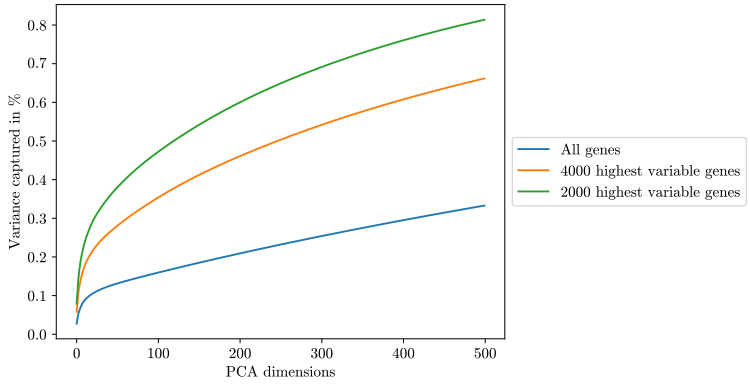

Figure 7.3: We applied PCA to either all genes or only the 2000 or 4000 highest variable genes. The plot shows the number of principal components vs the variance they explain cumulatively.

models are trained by excluding one intermediate time point from the dataset and then testing on that excluded time point. Additionally, we tested the methods by training them on all time points and then predicting the final time point. We refer to the distribution of the observed data in the embroyoid body dataset as the empirical distribution. Before we start we fine-tune τ for UOT-CFM and $g(t)$ for $[SF]^2M$. For entropic Action Matching we will use the same value for $g(t)$ as for $[SF]^2M$. All other hyperparameters are chosen as explained in Section 6.1.

Fine-tuning UOT-CFM. UOT-CFM is introduced in Section 3.3. It approximates the Optimal Transport coupling in OT-CFM by an unbalanced minibatch Optimal Transport coupling. The hyperparameter τ in unbalanced Optimal Transport penalizes deviation at the marginals from the initial and target distribution. As discussed in Section 2.2, adjusting the penalization parameter τ is challenging. Since the batch size can be kept small, effective training is possible even without regularization. Thus, we focus on the unregularized version here, where we only finetune τ and set $\varepsilon = 0$.

Ideally, we would tune τ for each dimension. However, for demonstration purposes we will only tune it for one dimensionality. Figure 7.3 shows that the variance ratio curve flattens after 100 dimensions. Thus, we fine-tune τ in 100 dimensions. Therefore, we train UOT-CFM with different τ values on the entire single cell dataset and compare its predictions to the observed samples at the last time point. Figure 7.4 presents the results. The dashed line shows the error of OT-CFM. UOT-CFM outperforms OT-CFM, and $\tau = 10$ appears to be the optimal parameter.

We will use this τ value across all dimensions and time points. Because this is a simplification, UOT-CFM performs mainly well on the data it was fine-tuned on. In 500 dimensions, we encountered numerical difficulties when calculating the approximated unbalanced Optimal Transport map with $\tau = 10$ and chose $\tau = 20$ instead.

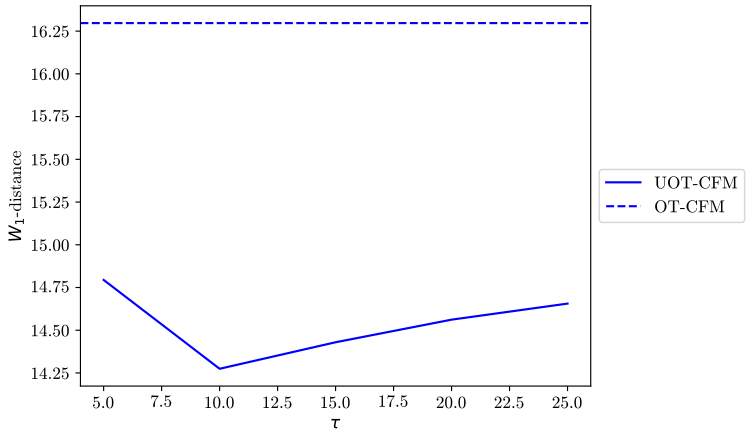

Figure 7.4: The solid blue line shows the W_1 distance between the empirical distribution and the distribution predicted by the UOT-CFM model with different values of τ. The dashed line indicates the corresponding error for the OT-CFM method.

Finetuning $[SF]^2M$. We use the same experimental setup as in the previous paragraph to determine the optimal value of $g(t)$ for $[SF]^2M$. We only tune $g(t)$ on the space of constant functions, which means $g(t) = g \in \mathbb{R}_+$. Figure 7.5 shows that $[SF]^2M$ outperforms OT-CFM for all tested values, with $g(t) \approx 0.25$ being optimal. Thus, we use the same value for $g(t)$ as in [Ton+23]. While we fine-tuned $g(t)$ with respect to the last time point, in the following we will predict the data also at timesteps 1, 2 and 3. For these earlier time steps, a slightly smaller $g(t)$ might be beneficial as the variance increases over time. Additionally, in smaller dimensions, a thorough exploration of the space is less critical. We will see that $[SF]^2M$ primarily performs well in large dimensions.

Experimental Results. The results of the experiments can be found in Tables 7.1 - 7.4. While we display I-AM in the results, to test the Action Matching loss with more dense in time samples, we excluded I-EAM in the results for clarity. I-EAM performed poorer than I-AM in low dimensions and comparably in high dimensions, while having a significantly higher computational time. The results for I-EAM can be found in Appendix B.

Action Matching vs. Flow Matching. Flow Matching consistently outperforms Action Matching. Both Action Matching and entropic Action Matching exhibit instability, with occasional large errors and high variance over three runs. Neklyudov et al. used entropic Action Matching and interpolated between time steps using mixtures of data to increase the number of available data points in [Nek+23b], but they did not specify their exact procedure. We attempted a similar approach with I-AM. Although we could not replicate the authors' results and used the deterministic variant, we found that I-AM often performs quite well. In higher dimensions,

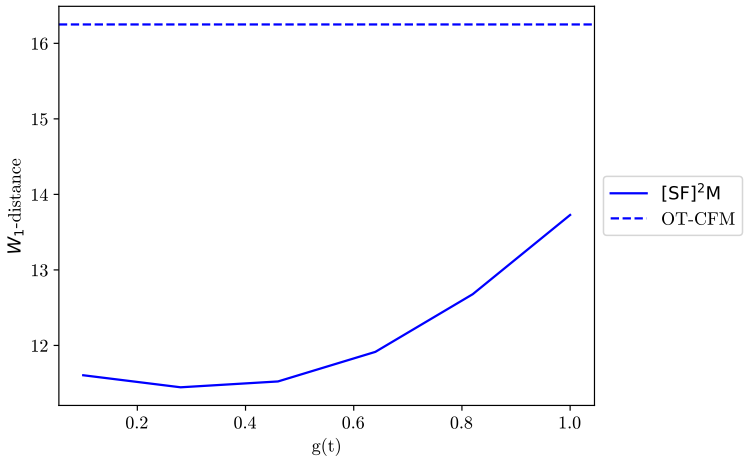

Figure 7.5: The solid blue line shows the W_1-distance between the empirical distribution and the distribution predicted by the $[SF]^2M$ model. The x-axis depicts varying values of $g(t)$ for the $[SF]^2M$ model. The dashed line indicates the corresponding error for the OT-CFM method.

it consistently outperforms OT-CFM and is only surpassed by the stochastic method $[SF]^2M$, which has significantly higher computational costs.

As observed in Section 6.4, I-AM excels in settings with many noise dimensions. The single cell data also contains a substantial amount of noise, likely not fully removed by PCA, especially for the higher dimensions. In Section 6.3, we noted that in two dimensions, I-CFM and OT-CFM consistently outperformed I-AM. While OT-CFM still outperforms I-AM in low-dimensional settings, I-AM surprisingly outperforms I-CFM consistently.

Stochastic vs. Deterministic. As entropic Action Matching shows a very volatile behavior, we focus on $[SF]^2M$. We see that it consistently outperforms the other methods in dimensions 50, 100 and 500. This confirms the results from the higher dimensional experiments in Section 6.4.

UOT-CFM vs. OT-CFM. We fine-tuned UOT-CFM in dimension 100. In 50 and 100 dimensions we can see that UOT-CFM slightly outperforms OT-CFM. Especially the further we move in time. For the other dimensions OT-CFM mostly performs a bit better. However, the hyperparameter τ depends heavily on the dimension as it can be interpreted as a transport radius. For smaller dimensions we would prefer a smaller value of τ while for higher dimensions we would prefer a larger τ.

Dimension	I-CFM	OT-CFM	UOT-CFM	$[SF]^2M$	AM	EAM	I-AM
2	0.43±0.09	0.33±0.01	0.35±0.06	0.43±0.06	7.48±9.08	10.48±7.3	0.41±0.04
5	1.07±0.08	0.95±0.04	0.94±0.05	1.02±0.01	1.28±0.28	1.17±0.16	1.05±0.01
10	2.24±0.05	2.22±0.06	2.23±0.02	2.14±0.03	3.08±0.55	3.77±1.22	2.21±0.01
50	6.48±0.03	6.55±0.02	6.48±0.01	6.03±0.01	7.56±0.2	7.8±1.11	6.33±0.0
100	9.99±0.02	10.12±0.02	10.1±0.01	8.85±0.01	10.52±0.28	10.85±0.22	9.69±0.08
500	24.37±0.05	24.52±0.01	24.95±0.01	21.24±0.0	25.45±0.23	27.11±1.79	23.99±0.08

Table 7.1: We compare the W_1-distance of the empirical and predicted distributions at time point 1. We trained the model on all but this time points. The best model is highlighted in red, while the second best is highlighted in blue.

Dimension	I-CFM	OT-CFM	UOT-CFM	$[SF]^2M$	AM	EAM	I-AM
2	0.51±0.09	0.42±0.04	0.61±0.07	0.81±0.01	1.22±0.5	0.66±0.05	0.5±0.04
5	1.28±0.14	1.04±0.02	1.08±0.09	1.26±0.02	1.76±0.13	1.55±0.34	1.13±0.03
10	2.09±0.09	2.0±0.04	2.09±0.06	2.2±0.06	3.15±0.21	2.93±0.5	2.02±0.03
50	7.62±0.13	7.5±0.02	7.35±0.03	6.49±0.02	7.36±0.13	7.83±0.37	6.99±0.02
100	12.25±0.03	12.0±0.04	11.66±0.05	9.67±0.01	11.41±0.94	12.34±1.38	10.76±0.07
500	30.0±0.07	29.55±0.08	30.8±0.05	23.2±0.03	35.79±3.0	37.49±11.54	26.2±0.25

Table 7.2: We compare the W_1-distance of the empirical and predicted distributions at time point 2. We trained the model on all but this time points. The best model is highlighted in red, while the second best is highlighted in blue.

Computational Complexity. The time in seconds needed to train the models can be seen in Table 7.5. While I-AM showed a good performance, it is also extremly fast to train. Further, its computational time stays stable over the dimensions.

Comparing High and Low Dimensional Predictions.

We aim to visually compare our models. Specifically, we examine the paths the models learned in the first two dimensions. For this analysis, we use models trained on all time points. We also want to compare the final points predicted by the models, starting from the observed initial samples. Figure 7.6 shows the initial and target distributions we aim to interpolate between. The data we use is the normalized PCA data.

Figure 7.7 presents the results of the models trained with 2, 5, or 100 dimensional data. The plots display the projections of the first two dimensions of the models' predictions. Specifically, the black scatters show the first two dimensions of the empirical initial distribution given to the models, while the blue scatters show the first two dimensions of the learned target disributions. The red lines illustrate examples of the first two dimensions of learned sample paths between

Dimension	I-CFM	OT-CFM	UOT-CFM	$[SF]^2M$	AM	EAM	I-AM
2	0.69±0.16	0.47±0.03	0.63±0.04	1.0±0.11	11.52±9.94	20.77±33.93	0.52±0.04
5	1.21±0.11	1.15±0.04	1.13±0.09	1.75±0.13	1.89±0.22	3.88±3.23	1.14±0.07
10	2.13±0.11	2.19±0.02	2.21±0.03	2.43±0.05	3.75±0.81	4.0±0.88	2.04±0.01
50	8.78±0.1	8.77±0.22	8.26±0.09	6.97±0.02	8.04±0.53	9.88±2.37	7.49±0.1
100	14.12±0.2	13.58±0.04	12.62±0.13	10.14±0.01	24.03±19.87	13.62±2.82	11.47±0.15
500	33.07±0.03	32.3±0.04	35.76±0.37	24.26±0.01	75.78±42.13	40.31±3.24	29.65±0.88

Table 7.3: We compare the W_1-distance of the empirical and predicted distributions at time point 3. We trained the model on all but this time points. The best model is highlighted in red, while the second best is highlighted in blue.

Dimension	I-CFM	OT-CFM	UOT-CFM	$[SF]^2M$	AM	EAM	I-AM
2	0.57±0.14	0.35±0.06	0.51±0.13	0.8±0.07	1.43±0.79	0.75±0.34	0.46±0.18
5	1.21±0.12	1.02±0.1	1.0±0.02	1.18±0.06	1.06±0.03	1.07±0.01	0.98±0.16
10	2.87±0.51	2.06±0.04	2.2±0.05	2.15±0.05	2.4±0.18	2.41±0.19	2.4±0.06
50	15.25±0.63	11.55±1.04	9.15±0.55	7.41±0.06	7.9±0.52	8.2±0.1	9.97±0.8
100	19.71±1.51	16.74±0.99	14.38±0.1	11.35±0.04	16.81±6.55	15.16±1.88	14.22±0.53
500	42.98±0.89	40.59±0.29	43.13±0.19	27.53±0.03	49.87±10.83	43.86±6.71	40.09±8.5

Table 7.4: We compare the W_1-distance of the empirical and predicted distributions at the last time point. We trained the model on all including this time points. The best model is highlighted in red, while the second best is highlighted in blue.

Dimension	I-CFM	OT-CFM	UOT-CFM	$[SF]^2M$	AM	EAM	I-AM
2	31.57±18.29	265.25±3.52	1004.03±24.25	282.73±8.88	70.57±0.82	263.93±3.01	44.75±0.11
5	20.93±0.03	267.73±6.96	1031.34±10.38	301.5±44.25	70.0±0.01	259.9±0.09	44.24±0.16
10	20.93±0.06	265.81±2.2	1031.87±13.2	280.91±4.43	69.91±0.09	259.42±0.69	44.11±0.03
50	20.95±0.03	263.54±3.03	1049.12±17.06	296.72±0.42	70.25±0.47	261.36±2.26	44.18±0.03
100	20.96±0.06	264.92±0.95	1047.3±3.06	323.03±1.26	69.99±0.01	260.15±0.07	44.85±0.72
500	20.99±0.04	289.49±0.99	694.15±5.79	516.9±1.18	70.04±0.18	268.83±1.3	45.05±1.36

Table 7.5: Time in seconds to train each of the models on all available data points.

the initial and target distribution. Background scatters show the two dimensional empirical distributions of the single-cell data over time, as seen in Figure 7.2. Each row presents the results of one specific model trained in 2, 5 and 100 dimensions. Each column presents the results of all methods for this dimension.

What do we want to compare? In our comparison, we want to answer the following questions:

- Which models predict target distributions (blue scatters) similar to the empirical target distribution in Figure 7.6? Does the accuracy of the two-dimensional predictions degrade when the model is trained on higher-dimensional data? Specifically, do the predictions get noisier?
- How does the dimension of the model influence the sample paths? Do they remain consistent across different dimensions, or do they exhibit significant variation?

First we want that the two dimensional model predicts the two dimensional target distribution visually. Secondly, we evaluate the consistency of the models for higher dimensions, namely whether a higher-dimensional model still predicts the first two dimensions satisfactorily. For the sample paths we primarily check for consistency. Specifically, whether the first two dimensions of the sample paths, depicted in red in Figure 7.7, appear similar across models of different dimensions based on the same method.

Results. Generally, we observe that the projections of the models in two dimensions become noisier as the number of training dimensions increases, which is intuitively expected. For example, the OT-CFM model predicts a distribution in two dimensions that closely resembles the empirical target distribution. In five dimensions, it still learns similar paths and a comparable target distribution, albeit slightly noisier. However, in 100 dimensions, the OT-CFM model predicts a very noisy target distribution, with many data points far from the empirical distribution's support.

I-AM shows a similiar behavior to OT-CFM, wheras I-CFM's predictions are already very noisy for the five dimensional model.

In contrast, the AM and EAM models predict distributions that already deviate more from the empirical distribution in two dimensions, and they also predict many samples far from the actual support in the 100-dimensional models.

The $[SF]^2M$ and UOT-CFM models perform somewhat better; $[SF]^2M$ mainly predicts the densest region of the empirical target distribution. Although these predictions also become

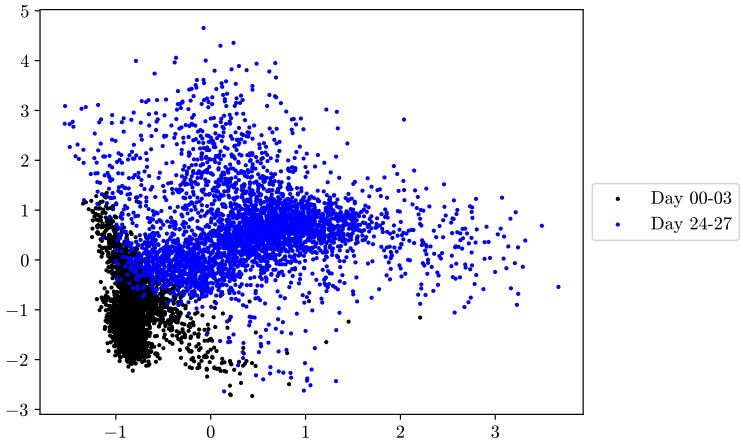

Figure 7.6: Initial and target distribution of the single cell data in two dimensions.

noisier in higher dimensions, they remain on the empirical distribution's support, and the dense region is mostly preserved.

Since we only fine-tuned UOT-CFM in 100 dimensions, the hyperparameter τ may not be optimal for the lower-dimensional cases. Nevertheless, the predictions from the two-dimensional UOT-CFM model are still convincing. Moreover, the predictions from the 100-dimensional model are significantly less noisy than those from OT-CFM and lie mostly on the support of the empirical target distribution.

The learned sample paths appear consistent across different dimensions. However, as the predictions become noisier, the sample paths also become noisier for higher-dimensional models. For example, they may move into directions in which they do not move in lower dimensions.

Conclusion

Our observations on the single-cell data confirm many of our previous findings. Notably, $[SF]^2M$ performs exceptionally well in high dimensions and shows the most consistency in its predictions across different data dimensions. I-AM performs well on noisy data, as predicted in Section 4.3, and is very fast to train. Visually, its predictions are not consistent. The advantages of UOT-CFM over OT-CFM in terms of Wasserstein distance were evident only with the specific data, on which UOT-CFM was fine-tuned. This highlights the importance of thorough fine-tuning. Further, UOT-CFM required significantly more computational time. However, as discussed in the previous section, UOT-CFM's predictions demonstrated greater consistency across different data dimensions compared to those of the OT-CFM model. This underscores the advantages of UOT-CFM in certain contexts.

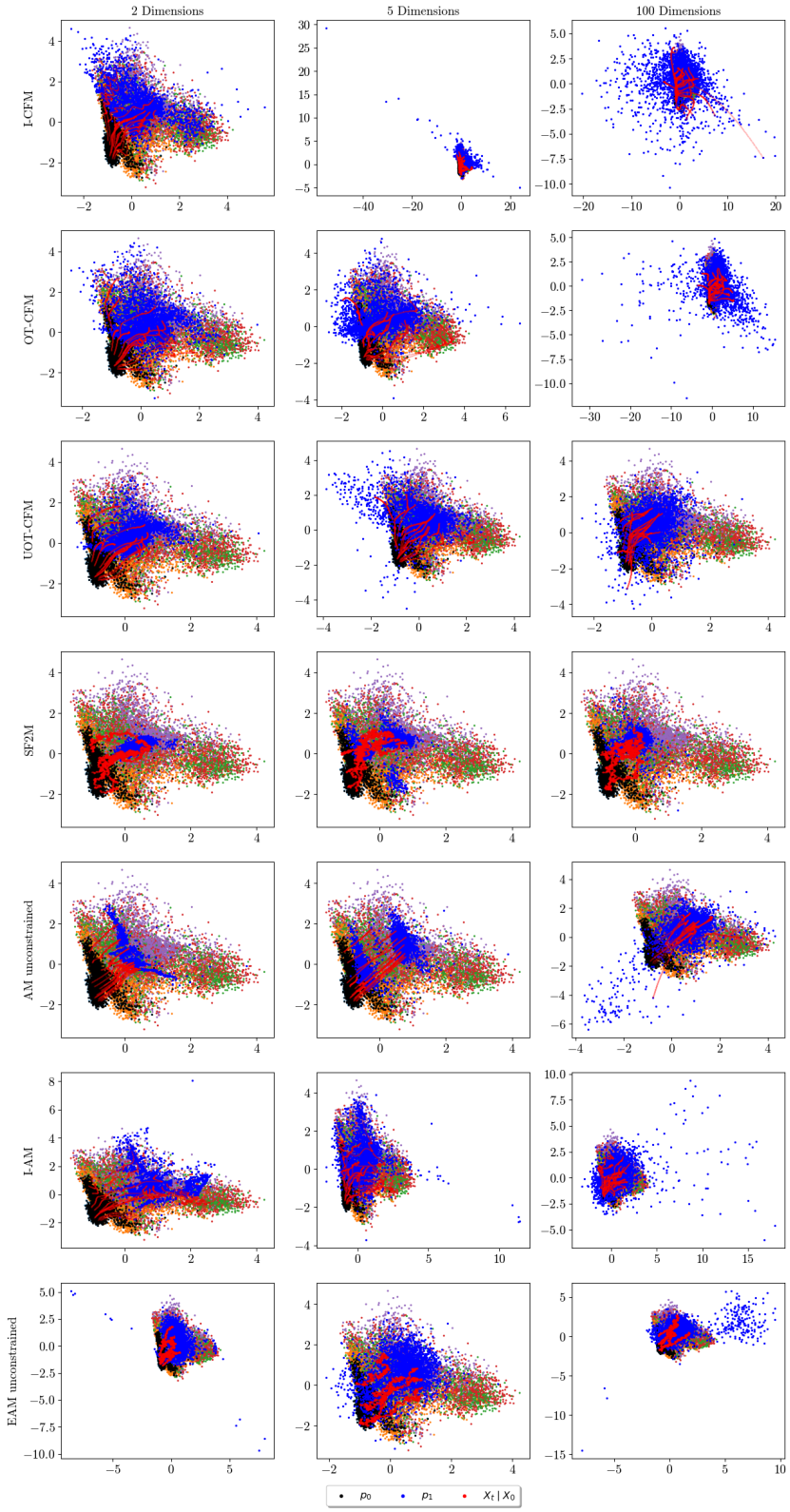

Figure 7.7: We plot the first two dimensions of the sample paths of the different models trained in dimensions 2, 5 or 100. The black scatters indicate the empirical initial distribution given to the models, while the blue scatters represent the learned target disributions. The red lines illustrate examples of learned sample paths between the initial and target distribution. Background scatters show the empirical distributions of the single-cell data over time, as seen in Figure 7.2.

Chapter 8

Conclusion and Outlook

Conclusion

In this thesis, we investigated Flow and Action Matching methods for learning deterministic and stochastic dynamics from snapshot data. Initially, we found that Action Matching in its original form is highly unstable. Discretizing the loss function in time often leads to exploding gradients. The only workaround we found was to use ReLU activation functions exclusively in the neural network architecture. However, this approach has limitations as it struggles to capture curvature in the dynamics. On the other hand, the adapted method, I-AM, demonstrated improved results when applied carefully. It appears particularly effective for data residing on a low-dimensional manifold embedded in a higher-dimensional space with substantial noise. This is what we also expected in Section 4.3. Additionally, I-AM has low computational time, making it advantageous in scenarios where quick computation is necessary and the data is noisy.

Furthermore, we observed that $[SF]^2M$ outperformed other methods in high-dimensional settings. In lower dimensions, $[SF]^2M$ performed worse than OT-CFM based on the Wasserstein metric. However, as demonstrated in Section 6.2 and Chapter 7, $[SF]^2M$ visually performes well in low dimensions, producing a convincing target distribution. This can be attributed to the diffusion term which acts as a regularizer and enhances model stability by exploring more of the space. Nonetheless, in low dimensions, deterministic methods excel in terms of the Wasserstein distance, while $[SF]^2M$ inherently has some error from the diffusion coefficient. Additionally, as discussed in Section 6.4, $[SF]^2M$ benefits from projecting the data onto a lower-dimensional manifold that aligns with the data's intrinsic dimension. Conversely, entropic Action Matching did not prove beneficial, likely due to the instability of the discretized loss and the overly simplistic data inference in I-EAM.

Finally, we introduced UOT-CFM, which replaces the minibatch Optimal Transport approx-

imation of OT-CFM with a minibatch unbalanced Optimal Transport approximation. Initial experiments in Chapter 3 showed visually convincing results. However, subsequent experiments revealed its high computational complexity and the necessity for careful fine-tuning of the hyperparameter τ. It only outperformed OT-CFM for the single-cell data case it was fine-tuned on, based on the Wasserstein distance. However, learning outliers might be rewarded in terms of the Wasserstein distance. Visually, as seen in Figure 7.7, UOT-CFM appeared more consistent than OT-CFM as the model dimension increased. It is important to consider whether the data structure justifies the use of UOT-CFM, which might be the case in scenarios with several data clusters evolving differently or when there are many outliers that should not be learned.

Outlook

Our adaptation of Action Matching involved interpolating data in a simple linear manner. Future work could explore alternative interpolation processes to improve the performance of Action Matching and entropic Action Matching. A similar concept was examined in [Nek+23a], where an additional neural network is used to learn the best interpolation process. Additionally, actions with different potentials can be matched using this method. Further research could thoroughly explore and compare these approaches with the methods introduced in this thesis.

In this thesis, we selected $g(t)$ as a known constant. Although the snapshot data does not provide sufficient information to learn $g(t)$, future work could consider varying $g(t)$ over time. Another potential improvement is to set $g(t)$ larger at the beginning of the training to promote bifurcations and then let $g(t) \to 0$ later in the training, as suggested in [Hug+22].

Finally, UOT-CFM did not fully approximate dynamic unbalanced Optimal Transport. Future research could extend Flow Matching to approximate the dynamic unbalanced Optimal Transport Problem more accurately.

Bibliography

[ABS21] Luigi Ambrosio, Elia Brué and Daniele Semola. *Lectures on optimal transport.* Vol. 130. Unitext. Springer, Cham, 2021.

[ABVE23] Michael S Albergo, Nicholas M Boffi and Eric Vanden-Eijnden. 'Stochastic interpolants: A unifying framework for flows and diffusions'. In: *arXiv preprint arXiv:2303.08797* (2023).

[And82] Brian DO Anderson. 'Reverse-time diffusion equation models'. In: *Stochastic Processes and their Applications* 12.3 (1982), pp. 313–326.

[ANWR17] Jason Altschuler, Jonathan Niles-Weed and Philippe Rigollet. 'Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration'. In: *Advances in neural information processing systems* 30 (2017).

[BB00] Jean-David Benamou and Yann Brenier. 'A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem'. In: *Numer. Math.* 84.3 (2000), pp. 375–393.

[Ber+20] Volker Bergen et al. 'Generalizing RNA velocity to transient cell states through dynamical modeling'. In: *Nature biotechnology* 38.12 (2020), pp. 1408–1414.

[Bre91] Yann Brenier. 'Polar factorization and monotone rearrangement of vector-valued functions'. In: *Comm. Pure Appl. Math.* 44.4 (1991), pp. 375–417.

[CGP16] Yongxin Chen, Tryphon T Georgiou and Michele Pavon. 'On the relation between optimal transport and Schrödinger bridges: A stochastic control viewpoint'. In: *Journal of Optimization Theory and Applications* 169 (2016), pp. 671–691.

[CGP21] Yongxin Chen, Tryphon T Georgiou and Michele Pavon. 'Stochastic control liaisons: Richard sinkhorn meets gaspard monge on a schrodinger bridge'. In: *Siam Review* 63.2 (2021), pp. 249–313.

[Cha+21] Laetitia Chapel et al. 'Unbalanced optimal transport through non-negative penalized linear regression'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 23270–23282.

[Che+18] Ricky T. Q. Chen et al. 'Neural ordinary differential equations'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems.* 2018, 6572–6583.

[Chi+18] Lenaïc Chizat et al. 'Scaling algorithms for unbalanced optimal transport problems'. In: *Math. Comput.* 87.314 (2018), pp. 2563–2609.

[Cla+21] Christian Clason et al. 'Entropic regularization of continuous optimal transport problems'. In: *Journal of Mathematical Analysis and Applications* 494.1 (2021), p. 124432.

[Cut13] Marco Cuturi. 'Sinkhorn distances: Lightspeed computation of optimal transport'. In: *Advances in neural information processing systems* 26 (2013).

[Dud68] R. M. Dudley. 'The speed of mean Glivenko-Cantelli convergence'. In: *Ann. Math. Statist.* 40 (1968), pp. 40–50.

[Eva22] Lawrence C Evans. *Partial differential equations*. Vol. 19. American Mathematical Society, 2022.

[Fat+20] Kilian Fatras et al. 'Learning with minibatch Wasserstein : asymptotic and gradient properties'. In: *The 23rd International Conference on Artificial Intelligence and Statistics*. Vol. 108. PMLR, 2020, pp. 2131–2141.

[Fat+21a] Kilian Fatras et al. 'Minibatch optimal transport distances; analysis and applications'. In: *arXiv preprint arXiv:2101.01792* (2021).

[Fat+21b] Kilian Fatras et al. 'Unbalanced minibatch optimal transport; applications to domain adaptation'. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 3186–3197.

[FG15] Nicolas Fournier and Arnaud Guillin. 'On the rate of convergence in Wasserstein distance of the empirical measure'. In: *Probab. Theory Related Fields* 162.3-4 (2015), pp. 707–738.

[Föl88] Hans Föllmer. 'Random fields and diffusion processes'. In: *École d'Été de Probabilités de Saint-Flour XV–XVII, 1985–87*. Ed. by Paul-Louis Hennequin. Springer Berlin Heidelberg, 1988, pp. 101–203.

[GPC18] Aude Genevay, Gabriel Peyré and Marco Cuturi. 'Learning generative models with sinkhorn divergences'. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2018, pp. 1608–1617.

[Gra+19] Will Grathwohl et al. 'FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models'. In: *7th International Conference on Learning Representations, ICLR 2019*. 2019.

[HD05] Aapo Hyvärinen and Peter Dayan. 'Estimation of non-normalized statistical models by score matching.' In: *Journal of Machine Learning Research* 6.4 (2005).

[HGJ16] Tatsunori Hashimoto, David Gifford and Tommi Jaakkola. 'Learning Population-Level Diffusions with Generative RNNs'. In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. 2016, pp. 2417–2426.

[HJA20] Jonathan Ho, Ajay Jain and Pieter Abbeel. 'Denoising diffusion probabilistic models'. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[Hug+22] Guillaume Huguet et al. 'Manifold interpolating optimal-transport flows for trajectory inference'. In: *Advances in neural information processing systems* 35 (2022), pp. 29705–29718.

[Hut90] M.F. Hutchinson. 'A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines'. In: *Communications in Statistics - Simulation and Computation* 19.2 (1990), pp. 433–450.

[Hyn18] RJ Hyndman. *Forecasting: principles and practice*. OTexts, 2018.

[KB15] Diederik Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *International Conference on Learning Representations*. 2015.

[Klo+92] Peter E Kloeden et al. *Stochastic differential equations*. Springer, 1992.

[LGL23] Xingchao Liu, Chengyue Gong and Qiang Liu. 'Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow'. In: *The Eleventh International Conference on Learning Representations*. 2023.

[LH19] Ilya Loshchilov and Frank Hutter. 'Decoupled Weight Decay Regularization'. In: *7th International Conference on Learning Representations, ICLR 2019*. 2019.

[Lip+22] Yaron Lipman et al. 'Flow matching for generative modeling'. In: *arXiv preprint arXiv:2210.02747* (2022).

[LM+18] Gioele La Manno et al. 'RNA velocity of single cells'. In: *Nature* 560.7719 (2018), pp. 494–498.

[Lé13] Christian Léonard. 'A survey of the Schr\"odinger problem and some of its connections with optimal transport'. In: *arXiv preprint arXiv:1308.0215* (2013).

[Moo+19] Kevin R. Moon et al. 'Visualizing structure and transitions in high-dimensional biological data'. In: *Nature Biotechnology* 37.12 (Dec. 2019), pp. 1482–1492.

[Nek+23a] Kirill Neklyudov et al. 'A computational framework for solving Wasserstein Lagrangian flows'. In: *arXiv preprint arXiv:2310.10649* (2023).

[Nek+23b] Kirill Neklyudov et al. 'Action matching: learning stochastic dynamics from samples'. In: *Proceedings of the 40th International Conference on Machine Learning*. 2023.

[Onk+21] Derek Onken et al. 'Ot-flow: Fast and accurate continuous normalizing flows via optimal transport'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10. 2021, pp. 9223–9232.

[PC+19] Gabriel Peyré, Marco Cuturi et al. 'Computational optimal transport: With applications to data science'. In: *Foundations and Trends in Machine Learning* 11.5-6 (2019), pp. 355–607.

[Qua+20] Alessio Quaglino et al. 'SNODE: Spectral Discretization of Neural ODEs for System Identification'. In: *8th International Conference on Learning Representations, ICLR 2020*. 2020.

[RM15] Danilo Rezende and Shakir Mohamed. 'Variational inference with normalizing flows'. In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.

[Sae+19] Wouter Saelens et al. 'A comparison of single-cell trajectory inference methods'. In: *Nature biotechnology* 37.5 (2019), pp. 547–554.

[Sch+19] Geoffrey Schiebinger et al. 'Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming'. In: *Cell* 176.4 (2019), pp. 928–943.

[SE19] Yang Song and Stefano Ermon. 'Generative modeling by estimating gradients of the data distribution'. In: *Advances in neural information processing systems* 32 (2019).

[Séj+19] Thibault Séjourné et al. 'Sinkhorn divergences for unbalanced optimal transport'. In: *arXiv preprint arXiv:1910.12958* (2019).

[SH+97] Jürgen Schmidhuber, Sepp Hochreiter et al. 'Long short-term memory'. In: *Neural Comput* 9.8 (1997), pp. 1735–1780.

[Sko21] Maciej Skorski. 'Modern analysis of hutchinson's trace estimator'. In: *2021 55th Annual Conference on Information Sciences and Systems (CISS)*. IEEE. 2021, pp. 1–5.

[Son+20] Yang Song et al. 'Sliced score matching: A scalable approach to density and score estimation'. In: *Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 574–584.

[Son+21] Yang Song et al. 'Score-Based Generative Modeling through Stochastic Differential Equations'. In: *9th International Conference on Learning Representations, ICLR 2021*. 2021.

[SPV23] Thibault Séjourné, Gabriel Peyré and François-Xavier Vialard. 'Unbalanced optimal transport, from theory to numerics'. In: *Handbook of Numerical Analysis* 24 (2023), pp. 407–471.

[Str23] Austin Stromme. 'Sampling from a Schrödinger bridge'. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 4058–4067.

[SW09] Galen R Shorack and Jon A Wellner. *Empirical processes with applications to statistics*. SIAM, 2009.

[Ton+20] Alexander Tong et al. 'TrajectoryNet: a dynamic optimal transport network for modeling cellular dynamics'. In: *Proceedings of the 37th International Conference on Machine Learning*. 2020.

[Ton+23] Alexander Tong et al. 'Simulation-free Schröodinger bridges via score and flow matching'. In: *arXiv preprint arXiv:2307.03672* (2023).

[Ton+24] Alexander Tong et al. 'Improving and generalizing flow-based generative models with minibatch optimal transport'. In: *Trans. Mach. Learn. Res.* 2024 (2024).

[Tra+14] Cole Trapnell et al. 'The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells'. In: *Nature biotechnology* 32.4 (2014), pp. 381–386.

[Vil03] Cédric Villani. *Topics in optimal transportation*. Vol. 58. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2003.

[Vil09] Cédric Villani. *Optimal transport*. Vol. 338. Grundlehren der mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Old and new. Springer-Verlag, Berlin, 2009.

[Vin11] Pascal Vincent. 'A connection between score matching and denoising autoencoders'. In: *Neural computation* 23.7 (2011), pp. 1661–1674.

[WB19] Jonathan Weed and Francis Bach. 'Sharp asymptotic and finite-sample rates of convergence of empirical measures in Wasserstein distance'. In: *Bernoulli* 25.4A (2019), pp. 2620–2648.

[Wen+23] Qingsong Wen et al. 'Transformers in Time Series: A Survey'. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023*. ijcai.org, 2023, pp. 6778–6786.

[Yan+20] Hanshu Yan et al. 'On Robustness of Neural Ordinary Differential Equations'. In: *8th International Conference on Learning Representations, ICLR 2020*. 2020.

[Zhe+23] Shijie C Zheng et al. 'Pumping the brakes on RNA velocity by understanding and interpreting RNA velocity estimates'. In: *Genome biology* 24.1 (2023), p. 246.

Appendix A

Learning Dynamics with Multiple Time Points

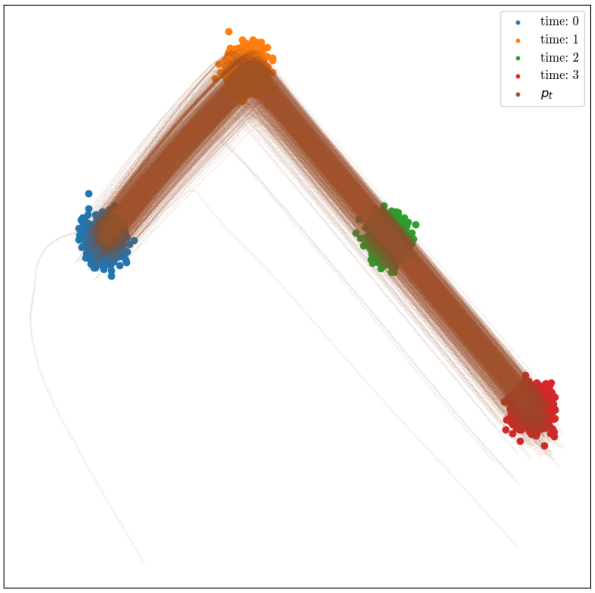

Figure A.1: Learned trajectory of the unrestricted Action Matching model. We see that it slightly overshoots the target distribution.

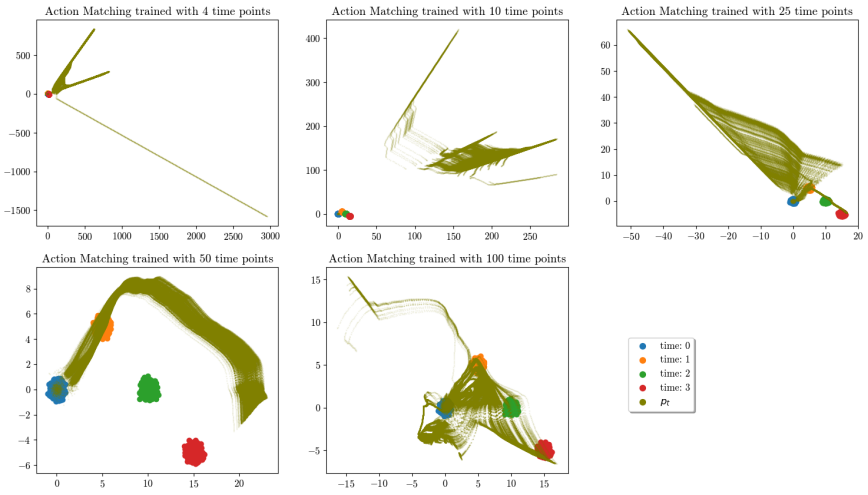

Figure A.2: Learned trajectory of Action Matching model using the MLP SiLU architecture. We discretized time and gave Action Matching only access to a varying amount of time points. We used 10000 samples at each point in time. Even with 50 or 100 time points Action Matching did not learn the dynamics well.

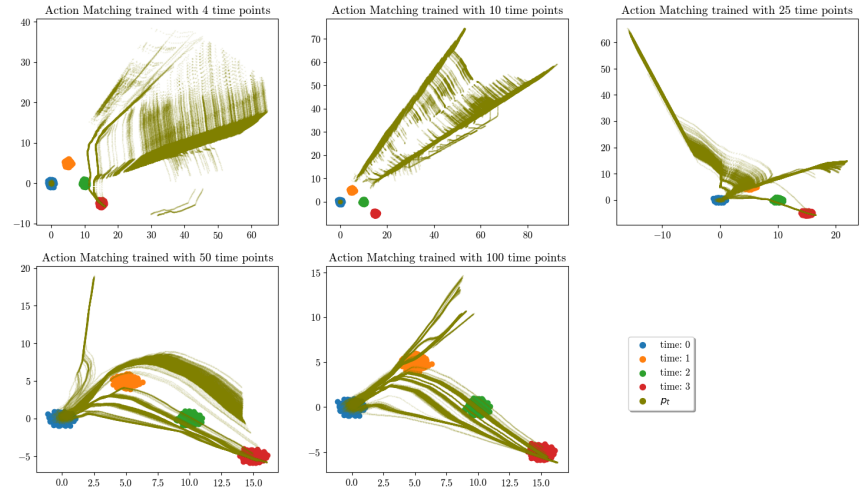

Figure A.3: Learned trajectory of Action Matching model using the MLP ReLU architecture. We discretized time and gave Action Matching only access to a varying amount of time points. We used 10000 samples at each point in time. Even with 50 or 100 time points Action Matching did not learn the dynamics well.

Appendix B

Single Cell Data

Here we state the results for the I-EAM model on the embroyoid body data from Chapter 7.

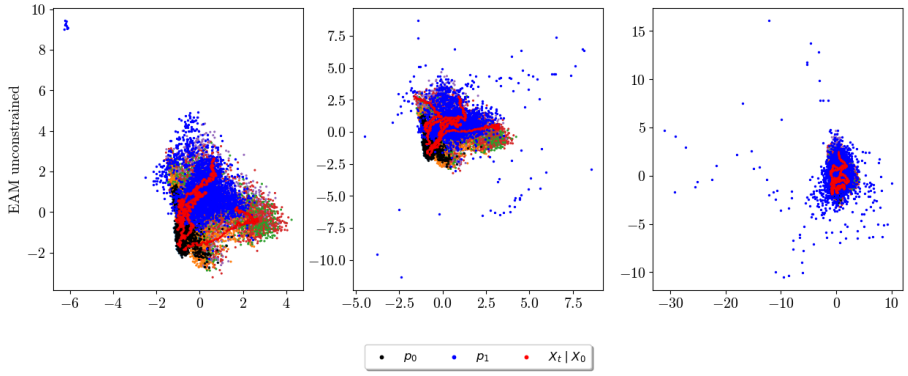

Figure B.1: We plot the first two dimensions of the sample paths of the I-EAM model trained in dimensions 2, 5 or 100. The black scatters indicate the empirical initial distribution given to the models, while the blue scatters represent the learned target disributions. The red lines illustrate examples of learned sample paths between the initial and target distribution. Background scatters show the empirical distributions of the single-cell data over time, as seen in Figure 7.2.

Dimension	time point 1	time point 2	time point 3
2	0.43±0.03	0.58±0.06	0.68±0.05
5	1.06±0.03	1.24±0.03	1.37±0.07
10	2.3±0.02	2.62±0.08	3.52±0.15
50	6.43±0.02	7.23±0.1	7.81±0.18
100	9.82±0.03	11.02±0.16	11.45±0.02
500	24.12±0.04	26.54±0.1	29.26±0.19

Table B.1: We compare the W_1-distance of the empirical and predicted distributions of I-EAM at time points 1, 2 and 3. We trained the model on all but this time points.

Dimension	W_1-distance	training time in seconds
2	0.51±0.09	206.0±11.51
5	1.15±0.15	196.6±0.11
10	3.59±0.15	201.37±8.26
50	9.57±0.39	198.95±2.98
100	14.08±0.86	198.08±0.49
500	37.34±0.6	208.68±14.85

Table B.2: We compare the W_1-distance of the empirical and predicted distributions at the last time point for I-EAM. We trained the model on all including this time points. The second column shows the training time of the model in seconds.