IMPERIAL

Imperial College London

Department of Computing

Training Energy-Based Models by Learning to Sample with Discrete Flows

Project Report

Author:
Moritz Elias Hauschulz

Supervisor:
Yingzhen Li
Co-Supervisor:
Zijing Ou
Second Marker:
Tolga Birdal

Submitted in partial fulfillment of the requirements for the MSc degree in Advanced Computing of Imperial College London

September 2024

Abstract

Training energy-based models with contrastive divergence is a hard problem due to the requirement to generate negative samples from the model distribution during training. This is particularly challenging in discrete domains, since conventional MCMC methods cannot exploit gradient information well and convergence is difficult to assess. Discrete flows are generative models that transform a source distribution to a data distribution through continuous time Markov chains, whose rate matrix or generating velocity is parameterised by a neural network. This work attempts to address the challenges in the training of energy-based models by using discrete flows. We contribute to the literature on discrete flows, specifically that on discrete flow matching, by proposing to directly learn the generating velocity through a regression loss instead of constructing it at sampling time from learned posteriors. We further employ importance sampling techniques to adapt the method to a data-free setting where beyond the target energy model, only a proposal distribution is available. Equipped with new tools that enable discrete flows to act as learned samplers, we propose and assess three different ways of applying them in the training of energy-based models. We demonstrate that all three have the capacity to learn on relatively low-dimensional synthetic benchmark problems, but find that without the use of MCMC-based refinements they are not competitive in higher dimensions where suitable proposal distributions are difficult to construct. In passing, we note the surprising ability of discrete flows to transform between data distributions, which opens up a number of directions for further research.

Acknowledgements

I thank Yingzhen Li for generously offering to supervise this project. Her expertise has been instrumental in guiding initial explorations as well as model refinement. I also thank Tolga Birdal for his helpful feedback on an initial draft. The idea for this project was conceived by Zijing Ou. He developed the initial theory of using importance sampling to learn discrete flows based on energy functions, which I pick up in this work. He further provided first versions of the proofs for Proposition 1 and Proposition 2. He also suggested the idea of deploying the method in interleaved training of EBMs. I also thank him for his continued patience and willingness to introduce me to many techniques at the cutting edge of research in discrete generative modelling, always pointing me to the newest publications in the field. This project would not have been such an enormous learning experience without his efforts.

Contents

Notation

Acronyms

EBM	Energy-based Model
DFS	Discrete Flow Sampler
VM	Velocity Matching
CVM	Conditional Velocity Matching
ECVM	Energy Conditional Velocity Matching
DFM	Discrete Flow Matching
EDFM	Energy Discrete Flow Matching
CFM	Conditional Flow Matching
NIST	National Institute of Standards and Technology
CIFAR	Canadian institute for Advanced Research
MMD	Maximum Mean Discrepancy
NLL	Negative Log Likelihood
CE	Cross-entropy
ED	Energy Discrepancy
CD	Contrastive Divergence
PCD	Persistent Contrastive Divergence
AIS	Annealed Importance Sampling
MCMC	Markov Chain Monte Carlo
MH	Metropolis-Hastings
HMC	Hamiltonian Monte Carlo
(D)ULA	(Discrete) Unadjusted Langevin Algorithm
(D)MALA	(Discrete) Metropolis Adjusted Langevin Algorithm
ACF	Auto-correlation Function
GWG	Gibbs with Gradient
DLP	Discrete Langevin Proposal
DAG	Directed Acyclic Graph
CTMC	Continuous Time Markov Chain
ODE	Ordinary Differential Equation
KL	Kullback-Leibler
NF	Normalising Flow
i.i.d.	Identically and independently distributed
LHS	Left hand side (of an equation)
RHS	Right hand side (of an equation)

Symbols

x	1-dimensional vector
$\boldsymbol{x}$	vector
$\boldsymbol{x}^t$	vector, indexed by t
x_t^i	i-th component of $\boldsymbol{x}_t$
$\boldsymbol{A}$	matrix
$\mathbb{R}^n$	n-dimensional Euclidean space
exp()	exponential function with base e
log()	natural logarithm, i.e. logarithm to the base e
$\lvert \cdot \rvert$	ℓ^1 norm
$\lvert\lvert \cdot \rvert\rvert$	ℓ^2 norm
$Cat(\boldsymbol{p})$	categorical distribution over probability vector $\boldsymbol{p}$
$x \propto y$	x is proportional to y
$\odot$	elementwise product
$o(h^n)$	$\frac{o(h^n)}{h^n} \to 0$ as $h \to 0$, i.e. the $o(h^n)$ term goes to zero strictly faster than h^n
$\mathcal{E}_\phi^k$	energy function parameterised by ϕ, indexed by iteration k
p_θ^k	probability density/mass function parameterised by θ, indexed by iteration k
∇_θ	gradient w.r.t θ
$\mathcal{X}$	discrete or continuous sample space
$\{0,1\}^D$	set of binary vectors of dimension D
$\boldsymbol{x} \sim p$	random vector $\boldsymbol{x}$ distributed with density p
$\text{supp}(p)$	support of probability density p
$\{\boldsymbol{x}_t\} \sim p$	set indexed by t, with each element i.i.d. distributed with density p
$p(\boldsymbol{x}\lvert\boldsymbol{y})$	conditional probability of $\boldsymbol{x}$ given $\boldsymbol{y}$
$\delta_{\boldsymbol{x}}(\cdot)$	Kronecker delta, i.e. probability mass function with all mass at $\boldsymbol{x}$
$\mathbb{E}_p$	expected value w.r.t density p
$\mathbb{V}_p$	variance w.r.t density p
$\tau_{\boldsymbol{x} \to \boldsymbol{x}'}$	trajectory from $\boldsymbol{x}$ to $\boldsymbol{x}'$
$p(\boldsymbol{x} \to \boldsymbol{x}')$	probability of transition from $\boldsymbol{x}$ to $\boldsymbol{x}'$
$R_t(\boldsymbol{x}_t, j)$	in CTMC, entry $(\boldsymbol{x}_t, j)$ of rate matrix R, where $\boldsymbol{x}_t$ is the current state
$u_t(\cdot, \boldsymbol{x}_t)$	in CTMC, generating velocity at $\boldsymbol{x}_t$ at time t
$u_t(\cdot, \boldsymbol{x}_t\lvert\boldsymbol{y})$	in CTMC, generating velocity at $\boldsymbol{x}_t$ at time t, conditional on $\boldsymbol{y}$

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In his investigations 'on the balance of living force between moving material points' [1], 19th-century physicist Ludwig Boltzmann proposed for the first time to express the probability of a system being in a certain state $\boldsymbol{x}$ as a function of the state's energy $\mathcal{E}(\boldsymbol{x})$ and the system's temperature T. Little did he know that his idea, which to this day forms a core pillar of statistical physics and is known as the Boltzmann distribution, would quietly develop a second life in the realm of generative modelling. Most computer scientists probably will have never heard of Boltzmann, partly since Boltzmann distributions are known as energy-based models (EBMs) in the machine learning community, and partly due to the fact that EBMs have been sidelined by the recent dominance of other generative models such as transformers [2] or diffusion models [3]. Before we discuss the characteristics of EBMs that might inhibit their broader recognition, and how the development of new methods could change that, let us first define exactly what an EBM (or equivalently a Boltzmann distribution) looks like:

$$p(\boldsymbol{x}) = \frac{\exp(\frac{-\mathcal{E}(\boldsymbol{x})}{T})}{Z_p} \tag{1.1}$$

As Equation 1.1 shows, a Boltzmann distribution relates the energy of a state $\mathcal{E}(\boldsymbol{x})$ inversely to its probability, $p(\boldsymbol{x})$. That is, low energy states have high probability and high energy states have low probability — in the statistical physics context, this is due to low energy states being more stable. Increasing the temperature term T has a smoothening affect on the probability distribution, shifting probability mass from the peaks to lower probability areas. Z_p is known as the normalising constant and does not correspond to a physical variable in Boltzmann's original framework – its sole purpose is to ensure that the RHS integrates to one over the domain $\mathcal{X}$ within which $\boldsymbol{x}$ can assume states.Thus, in the discrete case $Z_p = \sum_{x \in \mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}))$ and in the continuous case $Z_p = \int_{x \in \mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}))d\boldsymbol{x}$. The normalising constant does not provide much theoretical value, which is why Equation 1.1 is more often specified as $p(\boldsymbol{x}) \propto \tilde{p}(\boldsymbol{x}) = \exp(\frac{-\mathcal{E}(\boldsymbol{x})}{T})$. This *unnormalised* $\tilde{p}(\boldsymbol{x}) \propto \exp(\frac{-\mathcal{E}(\boldsymbol{x})}{T})$ distribution is what makes the framework so interesting in the machine learning context: We can simply parameterise $\mathcal{E}$ by a neural network with parameters ϕ (yielding $\mathcal{E}_\phi$, and implicitly p_ϕ and $\tilde{p}_\phi$) without having to worry about restricting the output of the neural network to integrate to one as would be the case if we were modelling p directly. However, this infinite flexibility in parameterisation comes at the expense of difficult training, sampling and likelihood evaluation. The difficulty of likelihood evaluation in EBMs is simply owed to the intractability of computing Z_p. Circumventing its approximation, sampling from EBMs is traditionally achieved by expensive and parameter dependent MCMC sampling [4], and training is achieved through so-called contrastive divergence (CD) which pushes down

the energy (i.e. the negative unnormalised log probability) of real samples and pushes up that of 'negative' samples from the model [5]. Of course, obtaining these negative samples requires sampling from the model. This challenge is precisely what motivates this project. We ask: How can we amortise the process of sampling from EBMs to improve efficiency in their training? Our investigation will be focused on the case where $\mathcal{X}$ is discrete, since MCMC methods are less effective here and recent progress in the use of discrete flows [6][7] for generative modelling suggest untapped potential in this area.

The most notable previous attempt to amortise the MCMC sampling process in training discrete EBMs, termed EB-GFN, leverages generative flow networks (GFlowNets) as surrogate samplers that are learned simultaneously with the EBM [8]. GFlowNets sample each dimension sequentially and are trained by sampling full trajectories, resembling approaches in reinforcement learning [9]. Therefore, steps required for generation *and* training increase linearly with the dimensionality, making them inefficient on high-dimensional discrete generation tasks such as language modelling. Most other attempts have focused on mixing generative models with MCMC. In the continuous domain for instance, normalising flows have been used to generate non-local proposals that improve mode coverage [10]. But even in the space of 'pure' MCMC samplers progress has all but stalled. Particularly the literature on discrete spaces, where samplers historically have lagged behind their continuous counterparts in performance due to the lack of available local structural information from gradients, has recently seen a series of advancements. The Gibbs with gradients (GWG) algorithm, for instance, augments notoriously slow discrete Gibbs sampling with gradient information to choose the next dimension to alter [11]. Another method, the discrete Langevin proposal (DLP) adapts the continuous Langevin proposal by restricting its domain, thereby enabling any number of dimensions to be altered simultaneously [12]. While both of the latter methods achieve comparable state-of-the-art performance in training of discrete EBMs, they rely on persistent CD (PCD) in which buffers persist the state of chains across iterations of EBM updates. Only the GFlowNet-based alternative, to our knowledge, is able to (almost) match their performance *without* the use of a persistent state [8]. Moreover, the complexity of a single MCMC step with DLP is linear in the dimensionality D of $\mathcal{X}$ [12].

We instead propose the use of discrete flows, which recently have emerged as the method of choice for learning discrete distributions, demonstrating state-of-the-art performance among non-autoregressive models on discrete generative tasks including language modelling [6][7]. Discrete flows rely on transforming a simple source distribution p_{source} to the data distribution p_{data} through a continuous time Markov chain (CTMC). Opposed to GFlowNet, DLP and GWG above, discrete flows have the potential to scale better in higher dimensions, as the discretisation with which the flow is pushed forward is independent of the dimensionality of $\mathcal{X}$, and each dimension can be sampled in parallel at each time step. This renders them a suitable candidate for amortising the sampling process in EBMs. However, existing literature has been focused on learning from *data* (a.k.a. generative modelling) while training EBMs requires sampling from an *unnormalised probability distribution* . Before we tackle the challenging task of training discrete EBMs, we therefore have to adapt discrete flows to this sampling task. After developing our toolkit, which we losely term discrete flow samplers (DFS), and having elicited the components with the most promising attributes, we shall then apply them in training EBMs. To this end, we propose a series of algorithms that interleave the training of DFS and EBM, providing theory and intuition for each, before evaluating them critically. We find that on high-dimensional problems, achieving performance comparable with existing paradigms without the use of MCMC-based refinements requires further work. However, we can present promising indicative results in the EBM setting, as well as on the use of discrete flows in modelling transformations between data distributions.

1.1 Contributions

The aim of this project is twofold: First, we a propose a series of novel velocity matching objectives to train discrete flows on energy instead of data. Second, we explore the use of the thus obtained learned DFS in training discrete EBMs. We believe that the work completed contributes to the literature in the following ways:

1. **We expand the set of viable objectives to train discrete flows on data.** We combine regression style objectives from conditional flow matching in the continuous domain [13][14] with the discrete flow matching (DFM) framework of [6] and [7] to propose conditional velocity matching (CVM). We show that this objective on many benchmark tasks achieves comparable performance with the original cross-entropy-based loss, and thereby extend the toolkit available in downstream tasks using discrete flows.

2. **We augment both existing objectives and our novel objective to learn from energy functions.** We provide a theoretical derivation of how both our regression-based loss and the original cross-entropy-based loss can be augmented to a data-free scenario where the flow is trained on an energy function via importance sampling (yielding objectives ECVM and EDFM). We prove that these objectives have the same minimisers as their data-based counterparts, as well as proportional gradients. We demonstrate their viability empirically where a suitable proposal distribution is available, and their limited effectiveness when proposals are poor.

3. **We demonstrate the flexibility of the resulting expanded discrete flow toolkit by considering different source distributions and probability paths.** Equipped with existing and new objectives for learning discrete flows from data or energy, we demonstrate their versatility by assessing performance across benchmark discrete modelling tasks for different source distributions and probability path types that transform the source to the target. We are able to show that discrete flows are suitable for transforming *between* data distributions with no loss of performance on the target generation quality, which we believe unlocks a myriad of downstream applications.

4. **We investigate the suitability of different discrete flow-based EBM training paradigms both theoretically and empirically.** Finally, we propose a series of approaches to exploit discrete flows in training EBMs with contrastive divergence. We demonstrate the capability of ECVM- and EDFM-based training algorithms to learn energy landscapes in 32-dimensional synthetic problems with a static proposal distribution. We further show the viability of combining DFS and EBM into a dynamic proposal for importance sampling in ECVM and EDFM, but caution that alignment of the models is fragile and must be reinforced with MCMC refinement. We finally explore the use of back-and-forth sampling through the discrete flow combined with a heuristic MH acceptance step akin EB-GFN [8].

1.2 Report Structure

The remainder of this report is structured as follows. Chapter 2 reviews the relevant background literature comprehensively. Chapter 3 focuses on our proposed velocity matching objective, as well as its energy-based version – and evaluates both critically. Chapter 4 approaches the challenge of EBM training with the tools developed in the previous chapter by exploring a series of algorithms first theoretically and then empirically. Chapter 5 concludes, and outlines various directions to explore based on the findings in this report.

Chapter 2

Background and Related Work

This chapter is designed to provide a broad overview of the literature on fundamental concepts in Monte Carlo methods (2.1), MCMC sampling (2.2), flow-based models (2.3) and finally EBMs (2.4). After introducing the notion of Monte Carlo sampling and estimation, we discuss MCMC methods which are often treated as de facto baselines for new samplers in both the discrete and the continuous domain. Flow models, including generative flow networks, conditional flow matching and discrete flow matching, represent an active area of research that this project builds on. EBMs are discussed as the aim of this project is to develop new and improved training and inference algorithms. Some of the more fundamental subsections, including 2.1.3, 2.2.1, parts of 2.2.2, 2.2.3 and 2.2.5, the introduction in 2.3, the introduction in 2.4 as well as parts of 2.4.1, 2.4.2 and 2.4.3 in this chapter follow the respective approaches taken in the more comprehensive book by Kevin P. Murphy [4].

2.1 Monte Carlo Methods

Monte Carlo sampling is a fundamental concept in the empirical approximation of expected values of functions of random variables. Monte Carlo sampling is deployed to approximate the quantity $\mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x}) = \sum_{x \in \mathcal{X}} f(\boldsymbol{x}) p(\boldsymbol{x})$ – or $\mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x}) = \int_{x \in \mathcal{X}} f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}$ in the continuous case – where f is called the target function and p is called the **target distribution**. The term Monte Carlo (MC) simply alludes to the use of randomness in this process. While this quantity can be computed — or approximated with numerical integration — in low dimensions in higher dimensions the problem quickly becomes intractable. In this case, it can be approximated by so-called MC integration via $\mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x}) \approx \frac{1}{N} \sum_{n=1}^{N} f(\boldsymbol{x}_n)$, where $\boldsymbol{x}_n \sim p(\boldsymbol{x})$. This section contains an outline of fundamental concepts in MC sampling, including variance of MC estimates, rejection sampling and importance sampling.

2.1.1 Variance of MC Estimates

It should be noted that while some MC techniques yield unbiased estimates, all estimates are naturally associated with a variance around the biased or unbiased mean. Given an unbiased MC approximation $\hat{\mu}$ of $\mu = \mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x})$, using the Central Limit Theorem, it can be shown that

$$(\hat{\mu} - \mu) \to \mathcal{N}(0, \frac{\sigma^2}{N}) \tag{2.1}$$

where $\sigma^2 = \mathbb{V}_{p(\boldsymbol{x})} f(\boldsymbol{x}) = \mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x})^2 - \mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x})^2$. As an approximation, to obtain an MC estimate that falls within ϵ of the true mean with probability of at least 95%, we require a

sample size of $N \geq \frac{4\hat{\sigma}^2}{\epsilon^2}$, where σ^2 is in practice also approximated by MC as $\hat{\sigma}^2$ [4].

2.1.2 Rejection Sampling

Rejection sampling is a flexible sampling method, and crucially is applicable to distributions $p(\boldsymbol{x}) = \frac{\tilde{p}(\boldsymbol{x})}{Z_p}$ with unknown normalising constant Z_p. At the core of rejection sampling lies a **proposal distribution** $q(\boldsymbol{x})$ which satisfies $Cq(\boldsymbol{x}) \geq \tilde{p}(\mathbf{x})$ for some constant C. The rejection sampling procedure for sampling from p is described in Algorithm L.1.

Algorithm L.1 Rejection Sampling

1: **Input:** $q(\mathbf{x})$ s.t. $Cq(\mathbf{x}) \geq \tilde{p}(\mathbf{x})$
2: Sample $\boldsymbol{x}_0 \sim q$
3: Sample $u_0 \sim \mathcal{U}[0, Cq(\boldsymbol{x}_0)]$
4: **if** $u_0 > \tilde{p}(\boldsymbol{x}_0)$ **then**
5: Reject $\boldsymbol{x}_0$
6: **else**
7: Accept $\boldsymbol{x}_0$
8: **end if**

The acceptance probability at each iteration is $1/C$. Clearly, if C is large, the algorithm becomes inefficient. Thus, it is important for q to closely resemble $\tilde{p}$. The interested reader is referred to [4] for more advanced approaches, including adaptive proposals. We include this basic technique here to illustrate the concepts of proposal and rejection, which are recurring themes in the MC sampling literature.

2.1.3 Importance Sampling

In importance sampling, we attempt to approximate $\mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x})$ by drawing samples $\boldsymbol{x}_n \sim q(\boldsymbol{x}_n)$ from the proposal distribution, and performing weighted MC integration, where ω_n are importance weights:

$$\mathbb{E}_{p(\boldsymbol{x})}[f(\boldsymbol{x})] \approx \frac{1}{N} \sum_{n=1}^{N} \omega_n f(\mathbf{x}_n) \tag{2.2}$$

The main types of importance sampling are direct importance sampling (2.1.3), self-normalised importance sampling (2.1.3) and annealed importance sampling (2.1.3) [4].

Direct Importance Sampling

Suppose we can evaluate the target distribution $p(\boldsymbol{x})$. Note that:

$$\mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x}) = \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} = \int f(\boldsymbol{x}) \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} q(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{q(\boldsymbol{x})} f(\boldsymbol{x}) \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \tag{2.3}$$

Thus, we can employ the following MC integration, where $\boldsymbol{x}_n \sim q(\boldsymbol{x})$ and $\omega_n = \frac{p(\boldsymbol{x}_n)}{q(\boldsymbol{x}_n)}$ are the importance weights:

$$\mathbb{E}_{p(\boldsymbol{x})} f(\boldsymbol{x}) \approx \frac{1}{N} \sum_{n=1}^{N} \frac{p(\boldsymbol{x}_n)}{q(\boldsymbol{x}_n)} f(\mathbf{x}_n) = \frac{1}{N} \sum_{n=1}^{N} \omega_n f(\boldsymbol{x}_n) \tag{2.4}$$

This constitutes an unbiased estimate of the true mean $\mathbb{E}_{p(\boldsymbol{x})}f(\boldsymbol{x})$. However, the normalised distribution $p(\boldsymbol{x})$ may not always be easy to evaluate. This issue is addressed in self-normalised importance sampling [4].

Self-Normalised Importance Sampling

In self-normalised importance sampling, we consider the scenario where only the unnormalised distribution $\tilde{p}(\boldsymbol{x}) = p(\boldsymbol{x})Z_p$ is accessible. In this case, the approximation also regards the normalising constant Z_p. We have:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} = \frac{\int f(\boldsymbol{x})\tilde{p}(\boldsymbol{x})d\boldsymbol{x}}{\int \tilde{p}(\boldsymbol{x})d\boldsymbol{x}} = \frac{\int \frac{\tilde{p}(\boldsymbol{x})}{q(\boldsymbol{x})}f(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}}{\int \frac{\tilde{p}(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x}} \tag{2.5}$$

$$\approx \frac{\frac{1}{N}\sum_{n=1}^{N}\tilde{w}_n f(\boldsymbol{x}_n)}{\frac{1}{N}\sum_{n=1}^{N}\tilde{w}_n} \tag{2.6}$$

Indeed, the final term approximates the normalising constant as $Z \approx \frac{1}{N}\sum_{n=1}^{N}\tilde{w}_n$, where now $\omega_n = \frac{\tilde{p}(\boldsymbol{x}_n)}{q(\boldsymbol{x}_n)}$. In the context of EBMs, this approach is particularly crucial, since in a Boltzmann distribution for a given energy function $\mathcal{E}$, $p(\boldsymbol{x}) = \frac{\exp(-\mathcal{E}(\boldsymbol{x}))}{Z}$, the normalising constant is typically intractable. The caveat with this approximation, however, is that it is only asymptotically unbiased [15][4].

Annealed Importance Sampling

In annealed importance sampling (AIS), as before, we assume that we cannot sample from the target $p = p_0$ or its unnormalised equivalent $\tilde{p}_0 \propto p_0$. We assume that we can only sample from a proposal $\tilde{p}_n \propto p_n$. We then construct a sequence of intermediate distributions as follows:

$$\tilde{p}_t(\boldsymbol{x}) = \tilde{p}_0(\boldsymbol{x})^{\beta_t}\tilde{p}_n(\boldsymbol{x})^{1-\beta_t} \tag{2.7}$$

Here, $1 = \beta_0 > \beta_1 > \cdots > \beta_n = 0$, so that $q_0 = \tilde{p}$. It is possible to construct a Markov chain $\boldsymbol{T}_t(\boldsymbol{x}, \boldsymbol{x}') = p_t(\boldsymbol{x}'|\boldsymbol{x})$, which leaves p_0 invariant. We then obtain an initial sample $\boldsymbol{x}_n \sim p_n$, and sequentially sample $\boldsymbol{x}_{t-1} \sim \boldsymbol{T}_{t-1}(\boldsymbol{x}_t, \cdot)$. Finally, we obtain $\boldsymbol{x}_0 \sim \boldsymbol{T}_0(x_1, \cdot)$, to which we assign a weight defined as

$$\omega = \frac{\tilde{p}_{n-1}(\boldsymbol{x}_{n-1})\tilde{p}_{n-2}(\boldsymbol{x}_{n-2})\cdots\tilde{p}_1(\boldsymbol{x}_1)\tilde{p}_0(\boldsymbol{x}_0)}{\tilde{p}_n(\boldsymbol{x}_{n-1})\tilde{p}_{n-1}(\boldsymbol{x}_{n-2})\cdots\tilde{p}_2(\boldsymbol{x}_1)\tilde{p}_1(\boldsymbol{x}_0)} \tag{2.8}$$

As before, the expectation can be approximated as in Equation 2.6. In the original paper [16] the author further suggests to instead take the thus obtained samples $\boldsymbol{x}$ to initialise Markov chains (2.2) and iterate for a predetermined number of times to obtain new samples $\boldsymbol{x}'$. It should be noted that this additional mixing does not invalidate the weights ω [4]. AIS is indispensable for evaluating the likelihood any Boltzmann distribution, and therefore for evaluating trained EBMs.

2.2 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods form a cornerstone of modern MC methodology, as conventional MC sampling methods (2.1) often perform poorly in high-dimensions. In the context of generative modelling, and in EBMs in particular, sampling high-dimensional distributions is a core problem (see Section D.2 for a categorisation of generative models). The idea of MCMC is the construction of a Markov chain on the sample space $\mathcal{X}$ whose stationary distribution converges to the target distribution $p(\boldsymbol{x})$. However, MCMC sampling inherently poses two challenges. First, while MCMC methods can be initialised anywhere in the support of the target distribution, the distribution of samples of the Markov chain only approaches the target distribution after sufficient 'mixing' has occurred. Thus, initial samples need to be discarded. The time required for this mixing process is called mixing time, varies depending on the target distribution and can generally not be determined analytically. Second, samples drawn from a Markov chain, even after sufficient mixing has taken place, are generally correlated. To obtain fully independent samples, in theory a new Markov chain would need to be initialised for each sample which is costly. In this chapter, we introduce the main algorithms in MCMC sampling. In the following, we are generally interested in sampling from a distribution $p(\boldsymbol{x}) = \frac{\tilde{p}(\boldsymbol{x})}{Z_p}$ with $Z_p = \sum_{\boldsymbol{x}\in\mathcal{X}} \tilde{p}(\boldsymbol{x})$ or $Z_p = \int_{\boldsymbol{x}\in\mathcal{X}} \tilde{p}(\boldsymbol{x})d\boldsymbol{x}$ in the continuous case — which can be seen to fit the structure of EBMs by setting $\tilde{p}(\boldsymbol{x}) = \exp(-\mathcal{E}(\boldsymbol{x}))$ for some energy function $\mathcal{E}(\boldsymbol{x})$.

2.2.1 Metropolis-Hastings

This technique applies to discrete and continuous sample spaces alike. Metropolis-Hastings (MH) is a MCMC technique that bears resemblance to rejection sampling (2.1.2), in that it involves a proposal distribution $q(\boldsymbol{x}'|\boldsymbol{x})$ for moving from $\boldsymbol{x}$ to $\boldsymbol{x}'$ at each step of the Markov chain. The proposed move is then accepted or rejected with probability

$$A = \min(1, \alpha), \tag{2.9}$$

$$\text{where } \alpha = \frac{p(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{p(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})} = \frac{\tilde{p}(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{\tilde{p}(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})} \tag{2.10}$$

Crucially, this expression cancels out the normalising constant Z_p. This makes MH, and related methods, suitable candidates for sampling from EBMs without computing the intractable normalising constant. The proposal distribution can be freely chosen as long as $\text{supp}(p) \subseteq \cup_{\boldsymbol{x}}\text{supp}(q(\cdot|\boldsymbol{x}))$. The procedure is summarised in Algorithm L.2 [4].

Algorithm L.2 Metropolis-Hastings algorithm (adapted from Alg. 12.1 in [4])

1: Initilise $\boldsymbol{x}_0$
2: **for** $t \leftarrow 0$ **to** T **do**
3: Sample $\boldsymbol{x}' \sim q(\boldsymbol{x}'|\boldsymbol{x}_t)$
4: Compute $\alpha = \frac{\tilde{p}(\boldsymbol{x}')q(\boldsymbol{x}_t|\boldsymbol{x}')}{\tilde{p}(\boldsymbol{x}_t)q(\boldsymbol{x}'|\boldsymbol{x}_t)}$
5: Compute acceptance probability $A = \min(1, \alpha)$
6: Set new state to $\boldsymbol{x}_{t+1} = \begin{cases} \boldsymbol{x}' & \text{with probability } A \\ \boldsymbol{x}_t & \text{otherwise} \end{cases}$
7: **end for**

The resulting Markov chain has a transition matrix given by

$$p(\boldsymbol{x}'|\boldsymbol{x}) = \begin{cases} q(\boldsymbol{x}'|\boldsymbol{x})A(\boldsymbol{x}'|\boldsymbol{x}) & \text{if } \boldsymbol{x}' \neq \boldsymbol{x} \\ q(\boldsymbol{x}|\boldsymbol{x}') + \sum\limits_{\boldsymbol{x}' \neq \boldsymbol{x}} q(\boldsymbol{x}'|\boldsymbol{x})(1 - A(\boldsymbol{x}'|\boldsymbol{x})) & \text{otherwise} \end{cases} \tag{2.11}$$

It can be shown that under mild conditions, this Markov chain satisfies the detailed balance condition given by $p(\boldsymbol{x}'|\boldsymbol{x})p(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{x}')p(\boldsymbol{x}')$, which expresses the fact that in equilibrium the probability mass of each state remains constant. This in turn can be shown to guarantee that p is the limiting distribution of the Markov chain. This is summarised in Theorem L.1, which is proved in [4].

Theorem L.1 (Theorem 12.2.1 in [4]) *If the transition matrix defined by the MH algorithm is ergodic and irreducible, then p is its limiting distribution.*

Finally, we briefly discuss the choice of proposal distribution q. The most simple proposal distributions are of the form $q(\boldsymbol{x}'|\boldsymbol{x}) = q(\boldsymbol{x}')$, known as independence samplers. For instance, q can be chosen to be Gaussian. Another common choice of proposal distribution is $q(\boldsymbol{x}'|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}'|\boldsymbol{x}, \tau^2\boldsymbol{I})$, which yields the so called random walk Metropolis algorithm. Note the use of a hyper parameter τ which impacts the variance of the proposal and thereby the mixing behaviour of the Markov chain. It is possible to adjust such parameters as the chain progresses in what is called **adaptive MCMC**, however it is necessary to ensure that the Markov property is not violated. Lastly, it is worth mentioning data-driven MCMC, where the proposal is a function of the current state $\boldsymbol{x}$ as well as the observed data. Thus, the proposal takes the form $q(\boldsymbol{x}'|\boldsymbol{x}, \text{data})$ [4].

2.2.2 Gibbs Sampling

This technique too applies to both discrete and continuous domains. In Gibbs sampling, the next state in the D-dimensional sample space is obtained by sequentially sampling each component conditional on the value of all remaining components. That is, for each i in $1, 2, ..., D$ we sample $x_{t+1}^i \sim p(x^i|\boldsymbol{x}_t^{-i})$, which is known as the full conditional of component i. Note that each iteration of the algorithm, a full sweep across all components is performed. It can be shown that Gibbs sampling technically is special case of MH with an acceptance rate of 100% for each proposal. However, this does not imply that Gibbs converges fast. Each component is sampled in sequence which can slow down sampling significantly in high-dimensional spaces. If groups of components are independent of one another in the sense that $p(\boldsymbol{x}^i|\boldsymbol{x}^{A\setminus\{i\}}, \boldsymbol{x}^B) = p(\boldsymbol{x}^i|\boldsymbol{x}^{A\setminus\{i\}})\ \forall i \in A$ and $p(\boldsymbol{x}^j|x^{B\setminus\{j\}}, \boldsymbol{x}^A) = p(\boldsymbol{x}^i|x^{B\setminus\{j\}})\ \forall j \in B$, then components in A and variables in B can be sampled in parallel, speeding up sampling. This is called **Blocked Gibbs Sampling** [4].

Another complication arises when a closed form is not given for $p(x^i|\boldsymbol{x}_t^{-i})$, so that it cannot be sampled trivially. In this case, we can resort to **Metropolis within Gibbs** a procedure by which we define a distribution $q(\hat{x}^i|x^i)$ from which to sample a proposal, which is then accepted with probability

$$\alpha^i = \frac{p(\boldsymbol{x}_{t+1}^{1:i-1}, \hat{x}^i, \boldsymbol{x}_t^{i+1:D}|\hat{x}^i)q(x_t^i|\hat{x}^i)}{p(\boldsymbol{x}_{t+1}^{1:i-1}, x_t^i, \boldsymbol{x}_t^{i+1:D}|x_t^i)q(\hat{x}^i|x_t^i)} \tag{2.12}$$

according to the procedure in 2.2.1 [4].

Gibbs with Gradient

Gibbs sampling is a particularly common method in sampling from discrete distributions. This is partly due to the fact that gradient-based algorithms such as 2.2.3 cannot easily be applied to discrete distributions since a gradient may not be available or, even if it is, the proposal may step outside the state-space. Gibbs with gradient (GWG), as proposed in [11], amends the Gibbs algorithm intuitively by choosing the next index to update depending on a gradient-based estimate of how likely the update would alter that component (instead of leaving it unchanged). The key insight is that for most discrete distributions, including discrete EBMs, the density is computed by restricting a continuous function to a discrete sample space. We consider here the log likelihood of a Boltzmann distribution given by $\log p(\boldsymbol{x}) = -\mathcal{E}(\boldsymbol{x}) - \log(Z)$. Where we write the unnormalised distribution as $\tilde{p}(\boldsymbol{x}) = \exp(-\mathcal{E}(\boldsymbol{x}))$ to align with the EBM case. The authors of [11] show that for D-dimensional binary samples the likelihood ratio of flipping each bit is approximated by

$$\tilde{d}(\boldsymbol{x}) = -(2\boldsymbol{x} - 1) \odot \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2.13}$$

and, the D-dimensional categorical case has a similar closed form. Note that effectively this can be used to obtain a proposal distribution over all 'neighbours' of the current state, that is all states $\boldsymbol{x}' \in H(\boldsymbol{x})$ where $H(\boldsymbol{x})$ is the Hamming ball of size one around $\boldsymbol{x}$. This can then simply be plugged into Algorithm L.2. A valid distribution in the binary case is obtained using softmax, as

$$q(i|\boldsymbol{x}) = \text{Categorical}\left(\text{Softmax}\left(0.5\tilde{d}(\boldsymbol{x})\right)\right) \tag{2.14}$$

After sampling i — the bit to flip — the proposal is obtained as $\boldsymbol{x}' = \text{bitflip}(\boldsymbol{x}, i)$. Note that Algorithm L.2 also requires $q(i|\boldsymbol{x}')$ which can be computed according to Equation 2.14. The categorical case is analogous. It is noteworthy that [11] report state-of-the art results on training discrete EBMs with persistent contrastive divergence (2.4.1), only outperformed by the more recent discrete Langevin proposal (2.2.4). They also establish theoretical results indicating near optimality in the class of samplers with *local* proposals.

2.2.3 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is designed to facilitate sampling in higher dimensions by leveraging gradient information. Note that this inherently restricts the application of this method to continuous state spaces. However, a recent adaptation to the discrete domain exists, and is covered in Subsection 2.2.4. The method is inspired by Hamiltonian mechanics.

Hamiltonian Mechanics

Hamiltonian mechanics are concerned with the motion of particles in an energy landscape. The state of a particle is determined by its position $\boldsymbol{x}$ and its momentum $\boldsymbol{v}$. The resulting state space is called the phase space. The energy of a state is characterised by the Hamiltonian function $\mathcal{H}(\boldsymbol{x}, \boldsymbol{v}) := \mathcal{E}(\boldsymbol{x}) + \mathcal{K}(\boldsymbol{v})$, where $\mathcal{E}(\boldsymbol{x})$ is the potential energy and $\mathcal{K}(\boldsymbol{v})$ is the kinetic energy. In statistical settings it is common practice to set $\mathcal{E}(\boldsymbol{x}) = -log\ \tilde{p}(x)$ and $\mathcal{K}(\boldsymbol{v}) = \frac{1}{2}\boldsymbol{v}^T\Sigma^{-1}\boldsymbol{v}$ where Σ is a positive definite matrix. Trajectories with constant energy, i.e. stable orbits, are characterised by the Hamiltonian equations:

$$\frac{d\boldsymbol{x}}{dt} = \frac{\partial H}{\partial \boldsymbol{v}} = \frac{\partial K}{\partial \boldsymbol{v}} \tag{2.15}$$

$$\frac{d\boldsymbol{v}}{dt} = -\frac{\partial H}{\partial \boldsymbol{x}} = -\frac{\partial \mathcal{E}}{\partial \boldsymbol{x}} \tag{2.16}$$

There a various ways of numerically integrating these equations in time. Only the more advanced **leapfrog** integrator is presented here. Essentially, the leapfrog integrator performs a partial momentum update first, then performs a full position update, and finally performs another partial momentum update [4].

$$\boldsymbol{v}_{t+\frac{1}{2}} = \boldsymbol{v}_t - \frac{\eta}{2}\frac{\partial \mathcal{E}(\boldsymbol{x}_t)}{\partial \boldsymbol{x}} \tag{2.17}$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta \frac{\partial \mathcal{K}(\boldsymbol{v}_{t+\frac{1}{2}})}{\partial \boldsymbol{v}} \tag{2.18}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_{t+\frac{1}{2}} - \frac{\eta}{2}\frac{\partial \mathcal{E}(\boldsymbol{x}_{t+1})}{\partial \boldsymbol{x}} \tag{2.19}$$

The parameter η is a step size. Next, we will see how this process has informed the HMC algorithm.

HMC Algorithm

To draw the connection to Hamiltonian mechanics, we can treat the Hamiltonian function as an energy function in a Boltzmann distribution:

$$p(\boldsymbol{x}, \boldsymbol{v}) = \frac{1}{Z}\exp\left[-H(x, \boldsymbol{v})\right] = \frac{1}{Z}\exp\left[-\mathcal{E}(\boldsymbol{x}) - \frac{1}{2}\boldsymbol{v}^T\boldsymbol{\Sigma}\boldsymbol{v}\right] \tag{2.20}$$

Furthermore, note that we are interested in the positional component of the phase space, whose distribution can be marginalised out as follows:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{v})d\boldsymbol{v} = \frac{1}{Z_p}e^{-\mathcal{E}(\boldsymbol{x})}\int \frac{1}{Z_q}e^{-\frac{1}{2}\boldsymbol{v}^T\Sigma\boldsymbol{v}}d\boldsymbol{v} = \frac{1}{Z_p}e^{-\mathcal{E}(\boldsymbol{x})} \tag{2.21}$$

where we have split up the normalising constant into one for the potential energy, Z_p, and one for the kinetic energy, Z_q. Thus, we recover our usual setting. Having transformed energies to probability distributions, we can now use the Leapfrog integrator to iteratively sample the sample space. It should be noted that the momentum, $\boldsymbol{v}$, is solely an auxiliary variable, which is sampled from a Gaussian at each step with no persistence. The HMC algorithm is given in Algorithm L.3. Essentially, given the previous state $(\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})$, we set $\boldsymbol{x}'_0 = \boldsymbol{x}_{t-1}$ and sample $\boldsymbol{v}'_0 \sim \mathcal{N}(\boldsymbol{0}, \Sigma)$. Subsequently, L leapfrog steps are performed to obtain a proposal $(\boldsymbol{x}'_L, \boldsymbol{v}'_L)$ which is accepted or rejected according to MH [4].

Metropolis Adjusted Langevin Algorithm and Unadjusted Langevin Algorithm

The Metropolis adjusted Langevin algorithm (MALA) is obtained from HMC by setting the number of leapfrog steps performed at each iteration to $L = 1$ [17]. This can be further simplified by removing the acceptance step. In practice, the gradient is often approximated by a stochastic gradient, which yields **stochastic gradient Langevin descent**. A further simplification — setting $\Sigma = \boldsymbol{I}$ and $\eta = \sqrt{2}$ — yields a stochastic differential equation known as Langevin diffusion:

$$\boldsymbol{x}_t = -\nabla\mathcal{E}(\boldsymbol{x}_t)dt + \sqrt{2}d\mathbf{B}_t \tag{2.22}$$

Here, $\boldsymbol{B}_t$ is D-dimensional Brownian motion. The resulting algorithm for sampling based on Langevin diffusion is known as **unadjusted Langevin algorithm** (ULA) [18][17][4].

Algorithm L.3 Hamiltonian Monte Carlo (adapted from Alg. 12.3 in [4])

for $t = 1 : T$ **do**
  Sample $\boldsymbol{v}_t \sim \mathcal{N}(0, \Sigma)$
  Set $(\boldsymbol{x}'_0, \boldsymbol{v}'_0) = (\boldsymbol{x}_t, \boldsymbol{v}_t)$
  Half step for momentum: $\boldsymbol{v}'_{1/2} = \boldsymbol{v}'_0 - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{x}'_0)$
  for $l = 1 : L$ **do**
    $\boldsymbol{x}'_l = \boldsymbol{x}'_{l-1} + \eta\Sigma^{-1}\boldsymbol{v}'_{l-1/2}$
    $\boldsymbol{v}'_{l+1/2} = \boldsymbol{v}'_{l-1/2} - \eta\nabla\mathcal{E}(\boldsymbol{x}'_l)$
  end for
  Full step for location: $\boldsymbol{x}'_L = \boldsymbol{x}'_{L-1} + \eta\Sigma^{-1}\boldsymbol{v}'_{L-1/2}$
  Half step for momentum: $\boldsymbol{v}'_L = \boldsymbol{v}'_{L-1/2} - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{x}'_L)$
  Compute acceptance probability $A = \min\left(1, \exp[-\mathcal{H}(\boldsymbol{x}'_L, \boldsymbol{v}'_L) + \mathcal{H}(\boldsymbol{x}_t, \boldsymbol{v}_t)]\right)$
  Set new sample to $\boldsymbol{x}_{t+1} = \begin{cases} \boldsymbol{x}' & \text{with probability } A \\ \boldsymbol{x}_t & \text{otherwise} \end{cases}$
end for

2.2.4 Discrete Langevin Proposal

Consider the update rule of the Langevin algorithm (prior to the acceptance step) with variance $\boldsymbol{\Sigma} = \eta\boldsymbol{I}$, and note that it can be rewritten as follows, since $\boldsymbol{\Sigma}^{-1} = \frac{1}{\eta}\boldsymbol{I}$:

$$\boldsymbol{x}' = \boldsymbol{x} - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{x}) + \sqrt{\eta}\boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} = \mathcal{N}(0, \boldsymbol{I}) \tag{2.23}$$

In continuous domains, the gradient step lets the process move closer to high probability (low energy) regions, and the normally distributed momentum introduces noise into the process. In [12], the authors note that the proposed steps are normally distributed with mean $\boldsymbol{x} - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{x})$ and variance $\eta\boldsymbol{I}$, i.e.

$$q(\boldsymbol{x}'|\boldsymbol{x}) = \frac{\exp(-\frac{1}{2\eta}||\boldsymbol{x}' - \boldsymbol{x} + \frac{\eta}{2}\nabla\mathcal{E}||_2^2)}{Z_{\mathcal{X}}(\theta)}, \tag{2.24}$$

Where $\mathcal{X}$ is the sample space, and $Z_{\mathcal{X}}(\theta)$ is the normalising constant over $\mathcal{X}$. The core idea of the proposed discrete Langevin proposal (DLP) is to extend this to *any* discrete domain, $\mathcal{X}$, without the need to compute $Z_{\mathcal{X}}(\theta)$, which typically in discrete domains has complexity $\mathcal{O}(S^D)$, where S is the number of states each of D dimensions can assume [12]. This is done by factorising (without further assumptions), the proposal distribution as $q(\boldsymbol{x}'|\boldsymbol{x}) = \prod_{i=1}^{d} q_i(\boldsymbol{x}'_i|\boldsymbol{x})$ with

$$q_i(\boldsymbol{x}'_i|\boldsymbol{x}) = \text{Categorical}\left(\text{Softmax}\left(-\frac{1}{2}\nabla\mathcal{E}(\boldsymbol{x})_i(\boldsymbol{x}'_i - \boldsymbol{x}_i) - \frac{(\boldsymbol{x}'_i - \boldsymbol{x}_i)^2}{2\eta}\right)\right) \tag{2.25}$$

Thus, after computing the gradient $\nabla\mathcal{E}$ in linear time, we can update all coordinates in parallel, making the resulting proposal computation $\mathcal{O}(D)$ overall. The resulting algorithm, called discrete unadjusted Langevin algorithm (DULA), can then optionally be coupled with an MH acceptance step (Algorithm L.2) to yield the discrete Metropolis-adjusted Langevin algorithm (DMALA). While DULA is asymptotically biased, DMALA is guaranteed to converge to the target distribution at the expense of an additional gradient computation (for $q_i(\boldsymbol{x}|\boldsymbol{x}')$). Together with Gibbs with Gradient (Section 2.2.2), DLP is currently considered state-of-the-art for sampling high-dimensional discrete sample spaces [11][12].

2.2.5 Mixing Time

Mixing time refers to the minimum number of transitions required from *any* initialisation before the stationary distribution of an MCMC, which as discussed previously corresponds to the target distribution, is reached. Formally, given a transition matrix $\boldsymbol{T}$ which is a function of both target and proposal distributions, and any $\epsilon > 0$, we define the mixing time from state $\boldsymbol{x}_0$ as

$$\tau_\epsilon(\boldsymbol{x}_0) := \min\{t : ||\delta_{\boldsymbol{x}_o}(\boldsymbol{x})\boldsymbol{T}^t - p||_1 \leq \epsilon\} \tag{2.26}$$

The mixing time of the Markov chain from *any* initialisation is given by

$$\tau_\varepsilon := \max_{x_0} \tau_\varepsilon(\boldsymbol{x}_0) \tag{2.27}$$

The mixing time is a function of the eigengap $\gamma = \lambda_1 - \lambda_2$ where λ_i is the i-th eigenvalue of the transition matrix. In particular, for a finite state space of *size* n, it can be shown that $\tau_\epsilon = \mathcal{O}(\frac{1}{\gamma} log \frac{n}{\epsilon})$. Generally, chains with low conductance (i.e. subsets of states between which transition is unlikely) have higher mixing times. In practice, since the transition matrix is generally hard to compute, convergence has to be assessed heuristically, for instance through trace plots and estimated potential scale reduction. Methods to speed up convergence can be found in [4]. A second convergence related issue is the chain's correlation, even after it has converged. Consecutive samples are not independent, and therefore a substantial number of samples may be necessary to obtain a empirical distribution that resembles the target. To assess this, one can compute the **effective sample size** that intuitively measures the corresponding sample size had each sample been drawn independently. Formally, the effective sample size is defined as follows:

$$N_{eff} := \frac{N}{1 + 2\sum_{l=1}^{\infty} \rho(l)} \tag{2.28}$$

where $\rho(l) := corr[\boldsymbol{x}_0, \boldsymbol{x}_l]$ is the auto-correlation function (ACF). It is evident that the effective sample size shrinks as the correlation between samples increases. More samples are needed to control the variance of the quantity of interest. In practice, the lag for which we compute the sum of the ACF values in the denominator is truncated at L, the last integer for which $\rho(L) > 0$ [4].

Simulated Annealing

As described above, mixing can be especially challenging in multi-modal distributions where Markov chains have low conductance. Here we describe intuitively the method of annealing. Recall that probability distributions can be expressed as Boltzmann distributions corresponding to an energy function $\mathcal{E}(\boldsymbol{x})$. In fact, the original formulation of a Boltzmann distribution is given by

$$p(\boldsymbol{x}) = \frac{\exp(\frac{-\mathcal{E}(\boldsymbol{x})}{T})}{Z_p} \tag{2.29}$$

where we (re)introduced the temperature term T. Effectively, this term determines how 'bumpy' the distribution is, by scaling down modes relative to other areas of the support at high temperatures. At high temperatures the acceptance probability, given by $\min(1, \alpha)$ with $\alpha = \exp(-(\mathcal{E}(\boldsymbol{x}') - \mathcal{E}(\boldsymbol{x}))/T)$ (assuming $Z_p = 1$, and a symmetric proposal) is therefore higher for proposed moves to higher-energy (lower-probability) states. This enables more exploration of the sample space. The temperature is usually adapted according to an annealing schedule, where the temperature is gradually reduced as the chain progresses [19][4].

2.3 Flow Models

The key idea of flow models is to obtain a complex target probability distribution $p_1(\boldsymbol{x}_1)$=$p(\boldsymbol{x})$ as a transformation of a simpler source distribution $p_0(\boldsymbol{x}_0)$. Flow models differ substantially between continuous or discrete sample spaces, and the applied transformation may be modelled as a finite sequence of transformations, or as continuous dynamics. Sections 2.3.1, 2.3.2 and 2.3.3 describe some key concepts in the more extensive literature on continuous flows, before we discuss two recent advances in the discrete domain in more detail.

2.3.1 Normalising Flows

Normalising flows (NF) generate $\boldsymbol{x}_1$ by passing a random variable $\boldsymbol{x}_0 \in \mathbb{R}^D$ drawn from the distribution $p_0(\boldsymbol{x}_0)$ through a nonlinear, invertible transformation $\boldsymbol{f} : \mathbb{R}^D \to \mathbb{R}^D$, so that $\boldsymbol{x}_1 = \boldsymbol{f}(\boldsymbol{x}_0)$. The target density is then obtained through the change of variable formula

$$p_1(\boldsymbol{x}_1) = p_0(\boldsymbol{g}(\boldsymbol{x}_1))|det\boldsymbol{J}(\boldsymbol{g})(\boldsymbol{x}_1)| = p_0(\boldsymbol{x}_0)|det\boldsymbol{J}(\boldsymbol{f})(\boldsymbol{x}_0)|^{-1} \tag{2.30}$$

where $\boldsymbol{g}(\boldsymbol{x}_1) = \boldsymbol{f}^{-1}(\boldsymbol{x}_1) = \boldsymbol{x}_0$ and $\boldsymbol{J}(\boldsymbol{f})(\boldsymbol{x}_0) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_0}|_{\boldsymbol{x}_0}$ is the Jacobian of $\boldsymbol{f}$ evaluated at $\boldsymbol{x}_0$. The invertible transformation $\boldsymbol{f}$ is known as the flow and p_1 is the pushforward of p_0 along $\boldsymbol{f}$. Typically, $\boldsymbol{f}$ consists of a sequence of bijections $\boldsymbol{f} = \boldsymbol{f}_N \circ \cdots \circ \boldsymbol{f}_1$ [4]. It is crucial to note that the bijective nature of the transformation allows sampling and density evaluation.

2.3.2 Adaptive MCMC

Adaptive MCMC basically exploits the fact that NFs can be used for both sampling and likelihood evaluation, to construct a *non-local* proposal distribution that can be used in Metropolis-Hastings style algorithms (Algorithm L.2) [20]. Note that this is a learned sampler that fits $\hat{p}$ to a target distribution p^*, unlike classical MCMC approaches which typically use gradients in local proposals. This is achieved by minimising an approximation of the Kullback-Leibler (KL) divergence, $D_{KL}(p^*||\hat{p})$. In [20], the authors discuss specifically the scenario where p^*, or equivalently $\mathcal{E}^* \propto p^*$ is available, but samples are sparse. The adaptive MCMC algorithm therefore combines the NF-based kernel with a second, 'classical' gradient step as in MALA or DULA (Section 2.2.3). The full algorithm reads as in Algorithm L.4. Importantly, as this procedure effectively allows sampling from an energy function, it can be applied in training an EBM via contrastive divergence (Subsection 2.4.1) as [10] show.

2.3.3 Continuous Normalising Flows

Here, we are interested in continuous flows, which intuitively can be thought of as a sequence of bijections $\boldsymbol{f} = \boldsymbol{f}_N \circ \cdots \circ \boldsymbol{f}_1$ in the limit $N \to \infty$. Note that this makes them suitable only for continuous state spaces. Mathematically, continuous flows are modelled as smooth time-varying vector fields $u : [0, 1] \times \mathbb{R}^D \to \mathbb{R}^D$ defining the ODE

$$d\boldsymbol{x} = u_t(\boldsymbol{x})dt \tag{2.31}$$

In this scenario, we are interested in the marginal probability path p_t between initial distribution p_0 and final distribution p_1 on $t \in [0, 1]$. If $\boldsymbol{x} \sim p_0(\boldsymbol{x}_0)$ is an initial condition for which u_t solves Equation 2.31, p_t is given by the pushforward for p_0 along u_t

$$p_t := [u_t]_{\#}(p_0) \tag{2.32}$$

Algorithm L.4 Adaptive MCMC (adapted from Alg. 1 in [20])

Require: target $\mathcal{E}^*$, parameterised NF $\mathbf{f}_\theta$, initial persistent state $\{\boldsymbol{x}_i\}_{i=1}^n$, step size $\tau > 0$, total steps k_{max}, number of Langevin steps per NF step k_{lang}

for $k \leftarrow 0$ to k_{max} **do**:
  if $k \bmod k_{lang} = 0$ **then**
    $\{\boldsymbol{x}'_{0,i}\} \sim p_0 \; \forall i$
    $\boldsymbol{x}'_i = \boldsymbol{f}_\theta(\boldsymbol{x}) \; \forall i$
    Compute $\alpha_i = \frac{\mathcal{E}^*(\boldsymbol{x}'_i)p_{1,\theta}(\boldsymbol{x}_i)}{\mathcal{E}^*(\boldsymbol{x}_i)p_{1,\theta}(\boldsymbol{x}'_i)} \; \forall i$
    Compute acceptance probability $A = \min(1, \alpha_i)$
    Set new sample $\forall i \; \boldsymbol{x}_i(k+1) = \begin{cases} \boldsymbol{x}'_i \text{ w.p. } A \\ \boldsymbol{x}_i \text{ otherwise} \end{cases}$
  else
    Perform HMC step (DULA/MALA) according to Algorithm L.2 (obtain $\boldsymbol{x}_i(k+1)$)
  end if
  Update NF with gradient of $\mathcal{L}_{NF,\theta} = -\frac{1}{n}\sum_{i=1}^n \log p_{1,\theta}(\boldsymbol{x}_{i,k+1})$ (i.e. $D_{KL}(p^t||p_{1,\theta})$)
end for

which satisfies the **continuity equation** $\frac{\partial p}{\partial t} = -\nabla \cdot (p_t u_t)$ [14]. Interestingly, from this we can recover the view taken in neural ODE [21], where the continuous flow is derived as the continuous limit of a ResNet [22] with infinitely many layers, and u is parameterised by a neural network with parameters θ. To see this, simply consider the Euler discretisation to integrate equation Equation 2.31:

$$\boldsymbol{x}(t+\epsilon) = \boldsymbol{x}(t) + \epsilon u_\theta(\boldsymbol{x}(t), t) \tag{2.33}$$

Training this flows with standard log likelihood objectives has proven difficult, in part due to the need for integrating the ODE in Equation 2.33 to generate samples, and for integrating two auxiliary ODEs to obtain the Jacobian and gradients for the likelihood computation [4].

2.3.4 Conditional Flow Matching

Flow matching essentially aims to transfer the simplicity of the regression objective used in diffusion models to training CNFs, to overcome the aforementioned difficulties. Assuming that we can sample efficiently from p_t and u_t, the basic flow matching objective is given by

$$\mathcal{L}_{FM} := \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x})}||v_\theta(t,\boldsymbol{x}) - u_t(\boldsymbol{x})||^2 \tag{2.34}$$

where $v_\theta(\cdot,\cdot) : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is a time-dependent vector field parameterised as a neural network [13]. Under the realisation that this is intractable for non-Gaussian source distributions, [14] extended this objective to **conditional flow matching (CFM)**. In CFM, we assume that $p_t(\boldsymbol{x}) = \int p_t(\boldsymbol{x}|\boldsymbol{z})q(\boldsymbol{z})d\boldsymbol{z}$, i.e. that $p_t(\boldsymbol{x})$ is a mixture of probability paths. The objective becomes

$$\mathcal{L}_{CFM}(\theta) := \mathbb{E}_{\mathcal{U}[t;0,1],q(\boldsymbol{z}),p_t(\boldsymbol{x}|\boldsymbol{z})}||v_\theta(t,\boldsymbol{x}) - u_t(\boldsymbol{x}|\boldsymbol{z})||^2 \tag{2.35}$$

where we only require tractable sampling from $q(\boldsymbol{z})$ and $p_t(\boldsymbol{x}|\boldsymbol{z})$. This is helpful when $u_t(\boldsymbol{x})$ is intractable, since, as shown by [14], $\nabla_\theta \mathcal{L}_{FM}(\theta) = \nabla_\theta \mathcal{L}_{CFM}(\theta)$. In the simplest case (termed I-CFM in [14]), where $q(\boldsymbol{z}) = q(\boldsymbol{x}_0)q(\boldsymbol{x}_1)$ is an independent coupling, the conditionals can be specified as Gaussians

$$p_t(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_0, \sigma^2) \tag{2.36}$$

$$u_t(\boldsymbol{x}|\boldsymbol{z}) = \boldsymbol{x}_1 - \boldsymbol{x}_0 \tag{2.37}$$

As $p_t(\boldsymbol{x}|\boldsymbol{z})$ is tractably samplable, in this case $\mathcal{L}_{CFM}$ is efficiently trainable. This can be generalised further to the case where $q(\boldsymbol{z}) := \pi(\boldsymbol{x}_0, \boldsymbol{x}_1)$ is the optimal transport (OT) map with marginals $q(\boldsymbol{x}_0)$ and $q(\boldsymbol{x}_1)$, which can improve training and speed up inference since OT flows are 'straighter' and therefore require fewer integration steps [14].

2.3.5 Discrete Flow Matching

Discrete Flow Matching (DFM)[1] emerges as the discrete analogue of conditional flow matching as presented in [13] and [14]. While the account in this section is focused on the nomenclature and concepts used in [6], the vast majority of results — and the fundamental idea of modelling the flow as a continuous time Markov chain – were first presented in [7].[2] For a direct comparison between concepts of [6] (DFM) and [7] (Discrete Flow Models) see Section D.1. We start with a brief explanation of continuous time Markov chains, based on the account in [7], before moving back to the DFM perspective.

Continuous Time Markov Chains To understand the concept of continuous time Markov chains (CTMC), consider a discrete sample space $\{1, ..., S\}^D$, where S is the number of possible values each of the D dimensions can assume. For simplicity, suppose here $D = 1$. The concept easily translates to higher dimensions. Starting at $t = 0$, a CTMC generates a discrete-space trajectory, and is defined by its transition probability between times steps t and $t + dt$, conditional on the current state of the trajectory x_t:

$$p_{t+dt|t}(j|x_t) = \delta_{x_t}(j) + R_t(x_t, j)dt \tag{2.38}$$

where $R_t(x_t, x_t) := -\sum_{k \neq x_t} R_t(x_t, k)$ to ensure $p_{t+dt}(\cdot|i)$ sums to 1. $R_t(x_t, j)$ is a time-dependent rate matrix, which governs the probability of moving from x_t to j at the infinitesimal time step dt (which is $R_t(x_t, j)dt$ for $j \neq x_t$). The above formulation yields a categorical distribution over the S possible transitions. As with continuous flows, the marginal distribution, called the probability path, at time t is denoted by $p_t(x_t)$. The rate of change of the marginal distribution is governed by the Kolmogorov equation:

$$\partial_t p_t(x_t) = \underbrace{\sum_{j \neq x_t} R_t(j, x_t) p_t(j)}_{\text{Inflow}} - \underbrace{\sum_{j \neq x_t} R_t(x_t, j) p_t(x_t)}_{\text{Outflow}} \tag{2.39}$$

where $p_t \in [0, 1]^S$ is a probability mass vector. Therefore Equation 2.39 defines an ODE in a vector space. It is important to note that the Kolmogorov equation represents the discrete analogue of the continuity equation mentioned in Subsection 2.3.3. Both equations basically require the rate of change in the probability of a state to equal the sum total of in-flowing and out-flowing probability mass via the time-varying vector field and rate matrix respectively. Since CTMC is a continuous process, in practice it has to be simulated with Euler discretisation, yielding $x_{t+\Delta t} \sim Cat(\delta_{x_t}(x_{t+\Delta t}) + R_t(x_t, x_{t+\Delta t})\Delta t)$, in the case of $D = 1$. Care has to be taken when generalising the Euler discretisation to higher dimensions, not

[1] In this report, we consistently use the notation DFM for Discrete Flow Matching. The original paper [6] uses the acronym Discrete FM to avoid confusion with Discrete Flow Models due to [7]. We will not refer to Discrete Flow Models as DFM.

[2] In fact, the idea of this project was based on [7], and [6] was published while the project was ongoing. We still choose the latter as our guiding framework, since it is formulated in a more general, though largely equivalent way.

to be restrictive about the number of dimensions that can change in a single discretisation step, even if the probability of more than one dimension changing at any given time in the underlying CTMC may be zero.

Let us now return to the DFM framework. As in Subsection 2.3.3, we are typically interested in transforming a source distribution p_0 to a target distribution p_1, so that t is restricted to the interval $[0, 1]$. Beyond the uniform $\mathcal{U}(\boldsymbol{x}_0; [0, 1])$ and mask $\delta_M(\boldsymbol{x}_0)$ distributions considered for p_0 in [7], the DFM framework is amenable to *any* source distribution. Another idea that is carried over from the continuous case in [13] is that of couplings $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1)$, which describe the training data and their distribution. Couplings can be unconditional (in line with the scenarios considered in [7]) given by $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1) = p_0(\boldsymbol{x}_0)_1(\boldsymbol{x}_1)$ or conditional $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1) = p_0(\boldsymbol{x}_0|\boldsymbol{x}_1)_1(\boldsymbol{x}_1)$, thereby extending the framework considered in [7]. An example for a conditional coupling is given by $(X_0, X_1) = (I \odot X_1 + (1 - I) \odot (M, \dots, M), X_1)$, where $I \in \{0, 1\}^N$ is some random variable and M is an artificial mask state.

The very general form of conditional probability paths considered in DFM is given by the following formula,

$$p_t(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1) = \sum_{j=1}^{m} \kappa_t^{i,j} w^j(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1) \tag{2.40}$$

where κ describes a general 'scheduler' which must satisfy $\sum_j \kappa_t^{i,j} = 1$ and $\kappa_t^{i,j} \geq 0$. We obtain a marginal probability path p_t which satisfies the source and target distributions for $t \in \{0, 1\}$ as

$$p_t(\boldsymbol{x}) = \sum_{\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathcal{X}} p_t(\boldsymbol{x}|\boldsymbol{x}_0, \boldsymbol{x}_1)\pi(\boldsymbol{x}_0, \boldsymbol{x}_1) \tag{2.41}$$

with $p_t(\boldsymbol{x}|\boldsymbol{x}_0, \boldsymbol{x}_1) = \prod_{i=1}^{N} p_t^i(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1)$ such that $p_t(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1) = \delta_{\boldsymbol{x}_t}(x^i)$ for $t \in \{0, 1\}$.

While the probability path is to be modeled as a CTMC as in [7], instead of explicitly defining a rate matrix, the authors of [6] define a discrete analogue of the continuous generating velocity in [13], called the generating probability velocity $u_t^i(z^i, \boldsymbol{x}_t)$ to describe the 'rate of probability change' at time $t \in [0, 1]$ in the current state x_t. The generation process is then simulated through Euler steps, where each token is updated *independently* according to

$$x_{t+h}^i \sim \delta_{x_t^i}(\cdot) + h u_t^i(\cdot, \boldsymbol{x}_t) \tag{2.42}$$

which yields a valid probability mass function for sufficiently small $h > 0$ if and only if $\sum_{z^i} u_t^i(z^i, \boldsymbol{x}_t) = 0$ and $u_t^i(z^i, \boldsymbol{x}_t) \geq 0$ $\forall i$ *and* $z^i \neq x_t^i$. This is equivalent to Equation 2.38. Before stating how such $u_t^i(\cdot, \boldsymbol{x}_t)$ can be constructed, it is important to define precisely what it means for a probability velocity to generate a probability path.

Definition L.1 (Definition 1 in [6]) *Probability velocity u_t generates the probability path p_t if, for all $t \in [0, 1)$ and given a sample $\boldsymbol{x}_t \sim p_t$, the sample $\boldsymbol{x}_{t+h}$ defined in Equation 2.42 satisfies $\boldsymbol{x}_{t+h} \sim p_{t+h} + o(h)$.*

It is further useful to verify that if we know that a conditional velocity generates a conditional path, then the corresponding marginal path is generated by the marginalised velocity. This is captured in Theorem L.1, proved in [6].

Theorem L.1 (Theorem 2 in [6]) *Given $u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)$ generating $p_t(\boldsymbol{x}|\boldsymbol{x}_0, \boldsymbol{x}_1)$, the marginal velocity field defined by $u_t^i(z^i, \boldsymbol{x}_t) = \sum_{\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathcal{X}} u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1) p_t(\boldsymbol{x}_0, \boldsymbol{x}_1|z)$ generates the marginal probability path $p_t(\boldsymbol{x})$, where $p_t(\boldsymbol{x}_0, \boldsymbol{x}_1|\boldsymbol{x}_t) = \frac{p_t(\boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)\pi(\boldsymbol{x}_0, \boldsymbol{x}_1)}{p_t(\boldsymbol{x})}$.*

This result is established by verifying that $p_t u_t$ satisfy the Kolmogorov equation (Equation 2.39), which is the discrete equivalent to the continuity equation mentioned in Subsection 2.3.3. Given Theorem L.1, if we can find a valid generating velocity for the conditional path specified in Equation 2.40, we know that its marginal generates the desired probability path. One solution is given in the following theorem.

Theorem L.2 (Theorem 3 in [6]) *A generating velocity for the general conditional paths* $p_t(\boldsymbol{x}|\boldsymbol{x}_0, \boldsymbol{x}_1)$ *defined in Equation 2.40 and Equation 2.41 is given by* $u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1) = \sum_{j=1}^m a_t^{i,j} w^j(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1) + b_t^i \delta_{\boldsymbol{x}_t}(z^i)$ *where* $a_t^{i,j} = \dot{\kappa}_t^{i,j} - \frac{\kappa_t^{i,j}}{\kappa_t^{i,\ell}} \dot{\kappa}_t^{i,\ell}$, $b_t^i = \frac{\dot{\kappa}_t^{i,\ell}}{\kappa_t^{i,\ell}}$ *and* $\ell = argmin_{j\in[m]} \left[\frac{\dot{\kappa}^{i,j}}{\kappa_t^{i,j}}\right]$.

This results in the following marginals

$$u_t^i(z^i, \boldsymbol{x}_t) = \sum_{j=1}^m a_t^{i,j} \hat{w}_t^i(z^i, \boldsymbol{x}_t) + b_t^{i,j} \delta_{\boldsymbol{x}_t}(z^i) \tag{2.43}$$

where the posteriors are given by

$$\hat{w}_t^i(z^i, \boldsymbol{x}_t) = \sum_{\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathcal{X}} w^j(z^i|\boldsymbol{x}_0, \boldsymbol{x}_1) p_t(\boldsymbol{x}_0, \boldsymbol{x}_1|\boldsymbol{x}_t) \tag{2.44}$$

It turns out that these posteriors are learnable. The general training loss is given by a summation over the cross-entropy loss of the m posteriors of the constituent probabilities in Equation 2.40:

$$\mathcal{L}_{DFM}(\theta) = -\sum_{j\in[m], i\in[S]} \mathbb{E}_{\mathcal{U}(t;[0,1]), \pi(\boldsymbol{x}_0,\boldsymbol{x}_1), p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1), w^j(y_j^i|\boldsymbol{x}_0,\boldsymbol{x}_1)} \log \hat{w}_t^i(y_j^i|\boldsymbol{x}_t; \theta) \tag{2.45}$$

Sampling Backward in Time

The authors of [6] further provide a simple tweak to Equation 2.43 that makes the marginal generating probability velocity suitable for sampling the probability path in reverse time – that is, according to $x_{t+h}^i \sim \delta_{x_t^i}(\cdot) - h u_t^i(\cdot, \boldsymbol{x}_t)$ – by simply assigning $\ell = \text{argmax}_{j\in[m]} \left[\frac{\dot{\kappa}^{i,j}}{\kappa_t^{i,j}}\right]$ (see Theorem 7 in [6]).

Example

To give a simple example, consider $p_t(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1) = (1-\kappa_t)\delta_{\boldsymbol{x}_0}(x^i) + \kappa_t \delta_{\boldsymbol{x}_1}(x^i)$ with κ_t monotonously increasing and $\kappa_0 = 0$, $\kappa_1 = 1$. The generating velocity is given by $u_t^i(z^i, \boldsymbol{x}_t) = \frac{\dot{\kappa}_t}{1-\kappa_t}\left[p_{1|t}(z^i|\boldsymbol{x}_t) - \delta_{\boldsymbol{x}_t}(z^i)\right]$, where the learnable 'probability denoiser' is given by $p_{1|t}(z^i|\boldsymbol{x}_t) = \sum_{\boldsymbol{x}_0,\boldsymbol{x}_1\in\mathcal{X}} \delta_{\boldsymbol{x}_1}(z^i) p_t(\boldsymbol{x}_0, \boldsymbol{x}_1|\boldsymbol{x}_t)$. The loss also simplifies to a single cross-entropy loss that closely resembles that in [7]:

$$\mathcal{L}(\theta) = -\sum_{i\in[S]} \mathbb{E}_{u(t,[0,1]), \pi(\boldsymbol{x}_0,\boldsymbol{x}_1), p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \log p_{1|t}(x_1^i|\boldsymbol{x}_t) \tag{2.46}$$

Interestingly, it can be shown that the probability denoiser is time-independent (compare Proposition 6 in [6]), and that for two schedulers κ_t and κ_t' and corresponding posteriors $\hat{w}_t(z^i|\boldsymbol{x}_t)$ and $\hat{w}_t'(z^i|\boldsymbol{x}_t)$, we have $\hat{w}_{t'}(z^i|\boldsymbol{x}_t) = \hat{w}_t'(z^i|\boldsymbol{x}_t)$, where $t' = \kappa_{\kappa_t'}^{-1}$ and κ^{-1} is the inverse of κ (Proposition 8 in [6]). Sampling backward in time in this case realises as $u_t^i(z^i, \boldsymbol{x}_t) = \frac{\dot{\kappa}_t}{\kappa_t}\left[\delta_{\boldsymbol{x}_t}(z^i) - p_{0|t}(z^i|\boldsymbol{x}_t)\right]$, where $p_{0|t}(z^i|\boldsymbol{x}_t) = \sum_{x_0,x_1\in\mathcal{X}} \delta_{x_0}(x^i) p_t(x_0, x_1|x_t)$ is the 'probability noise-prediction'. Under the assumption that $\pi(x_0, x_1) = p(x_0)p(x_1)$ and that the source is i.i.d. with $p(x_0) = \prod_{i=1}^N p(x_0^i)$, we even obtain an estimation-free velocity given by $u_t^i(z^i, \boldsymbol{x}_t) = \frac{\dot{\kappa}_t}{\kappa_t}\left[\delta_{\boldsymbol{x}_t}(z^i) - p_0(z^i)\right]$ [6].

2.3.6 Generative Flow Networks

While this method bears the term 'flow' in its name, it is more closely related to reinforcement learning (RL) then to the aforementioned paradigms. A generative flow network (GFlowNet) is a method by which an object, in a discrete or continuous state space, is generated by sequentially sampling each dimension of the D-dimensional state space conditional on the dimensions already generated [23]. Thus, sampling a GFlowNet involves exactly D sampling steps.[3] The similarity to RL stems from the training objective, which seeks to train the sequential generator to sample high-return solutions, as defined by $R(\boldsymbol{x})$. However, it differs from RL in that the aim is to generate diverse trajectories, making the trained model suitable for sampling. The method is regarded a 'network', as the intermediate steps in the generation can be regarded as nodes of a directed acyclic graph (DAG) in which crucially multiple trajectories can lead from the root node $\boldsymbol{s}_0$ to a given terminal node $\boldsymbol{x}$, where all dimensions have been sampled, via intermediate (incomplete) states $\boldsymbol{s}$. Note that the collection of terminal nodes $\boldsymbol{x}$ constitute the sample space $\mathcal{X}$. Given the reward at the terminal nodes, GFlowNets are design to construct a 'flow' of the total sum of terminal rewards starting from the root node, which satisfies the flow consistency Equation 2.47.

Proposition L.1 (Proposition 2 in [23]) *Let $\mathcal{A}(\boldsymbol{s})$ denote the action set available at $\boldsymbol{s}$ and let $T(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{s}'$ denote the set of state-action pairs that yield $\boldsymbol{s}'$. Define policy π generating trajectories starting at state $\boldsymbol{s}_0$ by sampling actions $a \in \mathcal{A}(\boldsymbol{s})$ according to $\pi(a|\boldsymbol{s}) = \frac{F(\boldsymbol{s},a)}{F(\boldsymbol{s})}$ where $F(\boldsymbol{s}, a) > 0$ is the flow through edge $(\boldsymbol{s}, a)$, and $F(\boldsymbol{s}) = R(\boldsymbol{s}) + \sum_{a\in\mathcal{A}(\boldsymbol{s})} F(\boldsymbol{s}, a)$ where $R(\boldsymbol{s}) = 0$ for non-terminal nodes $\boldsymbol{s}$ and $F(\boldsymbol{x}) = R(\boldsymbol{x}) > 0$ for terminal nodes $\boldsymbol{x}$. The flow consistency equation*

$$\sum_{\boldsymbol{s},a:T(\boldsymbol{s},a)=\boldsymbol{s}'} F(\boldsymbol{s}, a) = R(\boldsymbol{s}') + \sum_{a'\in\mathcal{A}(\boldsymbol{s}')} F(\boldsymbol{s}', a') \tag{2.47}$$

is satisfied. Let $\pi(\boldsymbol{s})$ denote the probability of visiting state $\boldsymbol{s}$ when starting at $\boldsymbol{s}_0$ and following $\pi(\cdot|\cdot)$. Then:

1. $\pi(\boldsymbol{s}) = \frac{F(\boldsymbol{s})}{F(\boldsymbol{s}_0)}$
2. $F(\boldsymbol{s}_0) = \sum_{\boldsymbol{x}\in\mathcal{X}} R(\boldsymbol{x})$
3. $\pi(\boldsymbol{s}) = \frac{R(\boldsymbol{x})}{\sum_{\boldsymbol{x}'\in\mathcal{X}} R(\boldsymbol{x}')}$

This results in the following loss function, to train the log flow parameterised by a neural network, where trajectories τ may be sampled from *any* policy. The flows are parameterised on a log-space for numerical stability.

$$\mathcal{L}_{\theta,\epsilon}(\tau) = \sum_{\boldsymbol{s}'\in\tau\neq\boldsymbol{s}_0} \left(\log\left[\epsilon + \sum_{\boldsymbol{s},a:T(\boldsymbol{s},a)=\boldsymbol{s}'} \exp F_\theta^{\log}(\boldsymbol{s}, a)\right] - \log\left[\epsilon + R(\boldsymbol{s}') + \sum_{a'\in\mathcal{A}(\boldsymbol{s}')} \exp F_\theta^{\log}(\boldsymbol{s}', a')\right] \right)^2 \tag{2.48}$$

In later work [9], an improved trajectory balance objective is proposed effectively training two flows: One representing a forward flow (or policy) $P_F(\cdot|\boldsymbol{s}; \theta)$ representing a distribution over states obtained by sampling one more dimension in $\boldsymbol{s}$, and one backward flow (or policy)

[3] Note that thus the number of sampling steps in GFlowNet is linear in the dimensionality D, while it is constant in the case of DFM

$P_B(\cdot|\boldsymbol{s};\theta)$ representing a distribution over all incomplete states obtained by deleting one coordinate from $\boldsymbol{s}$. The resulting objective is given in Equation 2.49.

$$\mathcal{L}_\theta(\tau) = \left[\log\left(\frac{Z_\theta \prod_{t=0}^{n-1} P_F(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t;\theta)}{R(\boldsymbol{s}_n)\prod_{t=0}^{n-1} P_B(\boldsymbol{s}_t|\boldsymbol{s}_{t+1};\theta)}\right)\right]^2 \tag{2.49}$$

Note the presence of a learned normalising constant Z_θ in this case, which we do not discuss here. While [9] initially propose to train this objective on trajectories sampled 'on-policy' from the current $P(\cdot|\cdot,\theta)$, [8] show that when data is available effective training can be achieved by using a mixture of trajectories sampled from the forward policy and the backward policy initialised from data. How this can be used in the context of EBMs is discussed in Section 2.4.1.

2.4 Energy Based Models

This section is dedicated to the precise definition of EBM training and sampling techniques drawing on fundamentals established in 2.1, 2.2 and 2.3. Recall that in an EBM, the joint probability is modelled as a Boltzmann distribution (abstracting away the temperature term):[4]

$$p(\boldsymbol{x}) = \frac{\exp(-\mathcal{E}(\boldsymbol{x}))}{Z_p} \tag{2.50}$$

where $\mathcal{E}(\boldsymbol{x}) \geq 0$ is the energy function and the normalising constant is $Z_p = \sum_{\boldsymbol{x}\in\mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}))$ or $Z_p = \int_{\boldsymbol{x}\in\mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}))d\boldsymbol{x}$ in the continuous case. Recall further that in the context of generative modelling, the energy function is learned as a neural network parameterised by ϕ and we write $\mathcal{E}_\phi(\boldsymbol{x})$, Z_ϕ and p_ϕ. Note that Z_ϕ is constant with respect to $\boldsymbol{x}$ but a function of ϕ. As we will see, Z_ϕ is not typically required in EBM training. However, it can be estimated using methods such as AIS (2.1.3) for approximate density evaluation purposes [4][26].

2.4.1 Contrastive Divergence

As with other common generative models (see Appendix D) EBMs can be trained by maximising the likelihood of the observed data under the model through gradient descent. Given the data distribution p_{data} and the model distribution p_ϕ, the standard MLE parameter estimate is formalised as

$$\phi^*_{MLE} = \text{argmax}_\phi(\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log p_\phi(\boldsymbol{x})]) \tag{2.51}$$

Where the log likelihood can be evaluated, this objective can be optimised by using gradient-based optimisers such as ADAM [27], for which the gradient with respect to the parameters ϕ can be obtained through automatic differentiation. However, the likelihood cannot be easily obtained for EBMs, due to the intractability of the normalising constant. Instead, EBM training relies on approximating this gradient by MC methods. Given Equation 2.50, the gradient of the log likelihood of an EBM is given by

$$\nabla_\phi \log p_\phi(\boldsymbol{x}) = -\nabla_\phi \mathcal{E}_\phi(\boldsymbol{x}) - \nabla_\phi \log Z_\phi \tag{2.52}$$

Since $\mathcal{E}_\phi$ is the quantity represented by a neural network, it is easy to compute its gradient by automatic differentiation. An MC estimate is only needed for the second term on the

[4] In fact, the precursors of EBMs are known as Boltzmann machines [24] [25].

right hand side of 2.52. To this end, note that $\nabla_\phi \log Z_\phi = \mathbb{E}_{p_\phi(\boldsymbol{x})}\left[-\nabla_\phi \mathcal{E}_\phi(\boldsymbol{x})\right]$. A valid MC approximation, as shown in [4], is given by

$$\nabla_\phi \log Z_\phi = \mathbb{E}_{p_\phi(\boldsymbol{x})}\left[-\nabla_\phi \mathcal{E}_\phi(\boldsymbol{x}_n)\right] \approx -\frac{1}{N}\sum_{n=1}^{N} \nabla_\phi \mathcal{E}_\phi(\boldsymbol{x}_n) \tag{2.53}$$

Thus, the difficulty of training EBM boils down to sampling $\boldsymbol{x}_n \sim p_\phi(\boldsymbol{x})$. The established method for doing so is to apply MCMC methods such as HMC (2.2.3) or MALA (2.2.3) in the continuous case, or Gibbs (2.2.2), Gibbs with Gradient (2.2.2) or DLP (2.2.4) in the discrete case. Note that all of the aforementioned samplers are solely based on the energy function $\mathcal{E}_\phi$ or its gradient by construction. To circumvent the computational cost of achieving convergence from randomly initialised chains, in **contrastive divergence** (CD) [5] chains are initialised from data samples, which yields good results after few iterations of MCMC. This process can further be amended by maintaining the state of the Markov chain across gradient steps. This so called **persistent contrastive divergence** (PCD) may improve performance further [28][29]. A more recent adaptation instead applies a replay buffer for the initialisation of the MCMC which is used for 95% of the iterations, as well as an additional L2 regularisation loss [30]. This approach achieves state of the art generation quality on high-dimensional datasets ImageNet32x32, ImageNet128x128 and CIFAR-10. The DLP-based discrete version of PCD similarly is currently the state of the art for discrete image modelling [12].

Various attempts have been made to overcome the difficulty of sampling $\boldsymbol{x} \sim p_\phi(\boldsymbol{x})$ in training EBMs by interleaving the process with the training of a second generative model to provide approximate samples. Below, we outline an approach in the continuous and one in the discrete domain.

CD with Adaptive Flow Sampling As discussed in Section 2.2, MCMC methods may struggle in high-dimensional multimodal distributions with disconnected modes by not transitioning between high-probability areas at the appropriate rates leading to long mixing times, and in the context of EBM training, to violating the assumption that negative examples be sampled from the EBM. To illustrate how flow-based modals may alleviate this issue, we consider for the continuous case the approach taken by [10]. In essence, they exploit the adaptive MCMC algorithm outlined in Subsection 2.3.2 to generate uncorrelated proposals to update a persistent state which is then used as negative samples in training the EBM. The algorithm is given in Algorithm L.5.

EB-GFN The difficulty in sampling from EBMs in discrete domains lies beyond the issue of mode coverage and in the computational complexity of many traditional MCMC methods, such as Gibbs sampling (2.2.2). While Gibbs with Gradients (2.2.2) and Discrete Langevin Proposals (2.2.4) amortise the computation for each step, sampling accurately from high-dimensional domains without access to a persistent state as in PCD may still require hundreds of steps. In [8] the authors deploy a GFlowNet (2.3.6) surrogate model that learns with reward $R(\boldsymbol{x}) = \exp(-\mathcal{E}_\phi(\boldsymbol{x}))$ in between iterations of CD. The training process is described in Algorithm L.6.

While CD is the most established training paradigm for EBMs, various other methods have been proposed. We outline three selected methods below. Further methods are described in [4], including adversarial training and the use of a Stein-discrepancy based loss function.

Algorithm L.5 Adaptive MCMC-EBM (adapted from Alg. 1 in [10])

Require: EBM $\mathcal{E}^*_\phi$, parameterised NF $\mathbf{f}_\theta$, persistent state $\{x_i\}_{i=1}^n$, step sizes $\tau_{EBM} > 0$ and $\tau_{NF} > 0$, total steps k_{max}, number of Langevin steps per NF step k_{lang}

for $k \leftarrow 0$ to k_{max} **do**:
  Sample $\{x_{\text{data},i}\} \sim p_{\text{data}}$
  if $k \bmod k_{lang} = 0$ **then**
    $\{\boldsymbol{x}'_{0,i}\} \sim p_0 \ \forall i$
    $\boldsymbol{x}'_i = \boldsymbol{f}_\theta(\boldsymbol{x}) \ \forall i$
    Compute $\alpha_i = \frac{\mathcal{E}^*(\boldsymbol{x}'_i)p_{1,\theta}(\boldsymbol{x}_i)}{\mathcal{E}^*(\boldsymbol{x}_i)p_{1,\theta}(\boldsymbol{x}'_i)} \ \forall i$
    Compute acceptance probability $A = \min(1, \alpha_i)$
    Set new sample $\forall i \ \boldsymbol{x}_i(k+1) = \begin{cases} \boldsymbol{x}'_i \text{ w.p. } A \\ \boldsymbol{x}_i \text{ otherwise} \end{cases}$
  else
    Perform HMC step (DULA/MALA) according to Algorithm L.2 (obtain $\boldsymbol{x}_i(k+1)$)
  end if
  Update EBM with gradient of $\mathcal{L}_{EBM,\phi} = -\frac{1}{n}\sum_{i=1}^n \mathcal{E}_\phi(\boldsymbol{x}_{data,i}) - \mathcal{E}_\phi(\boldsymbol{x}_i(k+1))$
  Update NF with gradient of $\mathcal{L}_{NF,\theta} = -\frac{1}{n}\sum_{i=1}^n \log p_{1,\theta}(\boldsymbol{x}_i(k+1))$ (i.e. $D_{KL}(p^t||p_{1,\theta})$)
end for

Algorithm L.6 EB-GFN Joint Training Framework (adapted from Alg. 1 in [8])

Require: Training dataset $\{\boldsymbol{x}_i\}$, hyperparameter $\alpha \in [0, 1]$, GFlowNet's P_F, P_B, Z with parameters θ and EBM $\mathcal{E}_\phi$

repeat
  $X \sim \text{Bernoulli}(\alpha)$
  if $X = 1$ **then**
    Sample forward trajectory $\tau \sim P_F(\tau)$
  else
    Uniformly sample $\boldsymbol{x}_i$ from dataset
    Sample backward trajectory $\tau \sim P_B(\tau|\boldsymbol{x}_i)$
  end if
  Update the GFlowNet via gradient step on $\mathcal{L}_\theta(\tau)$ with reward $R(x) = e^{-\mathcal{E}(x;\phi)}$ (2.49)
  Update energy function with Algorithm L.7
until some convergence condition

Algorithm L.7 GFlowNet-Guided Energy Function Update (adapted from Alg. 2 in [8])

Require: Training dataset $\{\boldsymbol{x}_i\}$, GFlowNet providing $\{P_F, P_B, Z\}$, energy function $\mathcal{E}_\phi$, horizon K

Sample $\boldsymbol{x} \sim p_{\text{data}}$
Sample K-step bwd. trajectory from $P_B(\cdot|\cdot;\theta)$: $\tau = (\boldsymbol{x} = \boldsymbol{s}_D \rightarrow \boldsymbol{s}_{D-1} \rightarrow \cdots \rightarrow \boldsymbol{s}_{D-K})$
Sample K-step fwd. trajectory from $P_F(\cdot|\cdot;\theta)$: $\tau' = (\boldsymbol{s}_{D-K} \rightarrow \boldsymbol{s}'_{D-K+1} \rightarrow \cdots \rightarrow \boldsymbol{s}'_D = \boldsymbol{x}')$
Compute $\alpha = \frac{\exp(-\mathcal{E}_\phi(\boldsymbol{x}'_1))P_B(\tau'|\boldsymbol{x}')P_F(\tau)}{\exp(-\mathcal{E}_\phi(\boldsymbol{x}_1))P_B(\tau|\boldsymbol{x})P_F(\tau')}$
Compute acceptance probability $A = \min(1, \alpha)$
Set negative sample to $\boldsymbol{x}' = \begin{cases} \boldsymbol{x}' & \text{with probability } A \\ \boldsymbol{x} & \text{otherwise} \end{cases}$
Update EBM with gradient of $\mathcal{E}_\phi(\boldsymbol{x}) - \mathcal{E}_\phi(\boldsymbol{x}')$

2.4.2 Score Matching

Score matching relies on the idea that if $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ are two log probabilities with equal gradient, then $f(\boldsymbol{x}) \equiv g(\boldsymbol{x})$. In the context of an EBM, since the normalising constant is invariant to the input, we have $\nabla_x log\ p_\phi(\boldsymbol{x}) = -\nabla_x \mathcal{E}_\phi(\boldsymbol{x})$. This is known as the Stein score, $\boldsymbol{s}_\phi(\boldsymbol{x})$. The score matching objective is then formulated in terms of the Fisher divergence [31]:

$$D_F(p_{\text{data}}(\boldsymbol{x}) \parallel p_\phi(\boldsymbol{x})) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\left[\frac{1}{2}\left\|\nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log p_\phi(\boldsymbol{x})\right\|^2\right] \tag{2.54}$$

The density function of the data distribution, p_{data} is unknown. Under regularity conditions including that p_{data} be continuously differentiable (i.e. non-discrete), as shown by [31], the objective can be rewritten as

$$D_F(p_{\text{data}}(x) \parallel p_\phi(\boldsymbol{x})) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\left[\frac{1}{2}\left\|\boldsymbol{s}_\phi(\boldsymbol{x})\right\|^2 + \text{tr}(J_{\boldsymbol{x}}\boldsymbol{s}_\phi(\boldsymbol{x}))\right] + C \tag{2.55}$$

where $J_{\boldsymbol{x}}$ is the Jacobian of $\boldsymbol{s}_\phi$. This removes the dependence on $p_{\text{data}}(\boldsymbol{x})$, but has quadratic complexity in the dimensionality D of $\boldsymbol{x}$. Denoising score matching alleviates the computational complexity as well as the restriction to continuous sample spaces by adding noise to the data samples to obtain a noisy data distribution $q(\tilde{\boldsymbol{x}}) = \int q(\tilde{\boldsymbol{x}}|\boldsymbol{x})p_{\text{data}}(\boldsymbol{x})d\boldsymbol{x}$. Score matching on this noisy distribution can be performed according to the following objective, proposed by [32].

$$D_F(q(\tilde{\boldsymbol{x}}) \parallel p_\phi(\tilde{\boldsymbol{x}})) = -\frac{1}{2}\mathbb{E}_{q(\tilde{\boldsymbol{x}},\boldsymbol{x})}\left[\left\|\boldsymbol{s}_\phi(\tilde{\boldsymbol{x}}) - \frac{(\boldsymbol{x}-\tilde{\boldsymbol{x}})}{\sigma^2}\right\|^2\right] + C \tag{2.56}$$

where σ is the variance of the Gaussian noise introduced to obtain $\tilde{\boldsymbol{x}}$. Note however, that this can only approximate the true score matching objective, as the the Fisher divergence is computed with respect to the noisy distribution. Sliced score matching is a consistent alternative, with linear complexity in D [33]. It involves minimising the sliced Fisher divergence given by

$$D_{SF}(p_{\text{data}}(\boldsymbol{x})||p_\phi(\boldsymbol{x})) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x}),p(\boldsymbol{v})}[\frac{1}{2}(\boldsymbol{v}^T\nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x}) - \boldsymbol{v}^T\nabla_{\boldsymbol{x}} \log p_\phi(\boldsymbol{x}))^2] \tag{2.57}$$

$$= \mathbb{E}_{p_{\text{data}}(\boldsymbol{x}),p(\boldsymbol{v})}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial \mathcal{E}_\phi(\boldsymbol{x})}{\partial x_i}v_i\right)^2 + \sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial^2 \mathcal{E}_\phi(\boldsymbol{x})}{\partial x_i \partial x_j}v_i v_j\right] + C \tag{2.58}$$

where $p(\boldsymbol{v})$ denotes a projection distribution s.t. $\mathbb{E}_{p(\boldsymbol{v})}[\boldsymbol{v}\boldsymbol{v}^T]$ is positive definite [4]. While score matching in its original form is only suitable for continuous state spaces, adaptations to the discrete setting exist [34].

2.4.3 Noise Contrastive Estimation

Noise contrastive estimation aims at approximating the EBM indirectly through creating two mixture models: $p_{n,\text{data}}(\boldsymbol{x}) = p(y=0)p_n(\boldsymbol{x}) + p(y=1)p_{\text{data}}(\boldsymbol{x})$ and $p_{n,\phi}(\boldsymbol{x}) = p(y=0)p_n(\boldsymbol{x}) + p(y=1)p_\phi(\boldsymbol{x})$, where y is distributed according to a Bernoulli distribution and p_n is some noise distribution such as $\mathcal{N}(0, \boldsymbol{I})$. Importantly, we parameterise the EBM as

$p_\phi(\boldsymbol{x}) = \exp(-\mathcal{E}_\phi(\boldsymbol{x}))/Z_\phi$ with Z_ϕ being a learnable scalar parameter. From Bayes theorem, we obtain

$$p_{n,\text{data}}(y=0|\boldsymbol{x}) = \frac{p_n(\boldsymbol{x})}{p_n(\boldsymbol{x}) + \nu p_{\text{data}}(\boldsymbol{x})} \tag{2.59}$$

$$p_{n,\phi}(y=0|\boldsymbol{x}) = \frac{p_n(\boldsymbol{x})}{p_n(\boldsymbol{x}) + \nu p_\phi(\boldsymbol{x})} \tag{2.60}$$

where $\nu = p(y=1)/p(y=0)$. The NCD objective then fits $p_\phi(\boldsymbol{x})$ to $p_{\text{data}}(\boldsymbol{x})$ indirectly through conditional MLE with the following objective:

$$\phi^* = \text{argmin}_\phi \mathbb{E}_{p_{n,\text{data}}(\boldsymbol{x})} \left[D_{KL}(p_{n,\text{data}}(y|\boldsymbol{x}) \parallel p_{n,\phi}(y|\boldsymbol{x})) \right] \tag{2.61}$$

$$= \text{argmax}_\phi \mathbb{E}_{p_{n,\text{data}}(\boldsymbol{x},y)} \left[\log p_{n,\phi}(y|\boldsymbol{x}) \right] \tag{2.62}$$

It is easy to establish that at the optimum $p_{\phi^*}(\boldsymbol{x}) \equiv p_{\text{data}}(\boldsymbol{x})$. The thus obtained $\mathcal{E}_{\phi^*}(\boldsymbol{x})$ is the unnormalised energy function matching the data distribution, and Z_{ϕ^*}, the normalisation constant, is learned as a byproduct [35].

2.4.4 Energy Discrepancy

Training continuous or discrete energy based models by energy discrepancy (ED), put forward in [36] and [37], is based on the idea of minimising the energy of true data points (positive samples) and maximising the energy of their perturbations (negative samples). Crucially, this does not require expensive MCMC sampling from the model. ED is based around the idea of contrastive potential induced by a conditional probability density $q(\boldsymbol{y}|\boldsymbol{x})$ for a given data distribution p_{data}. The contrastive potential is defined as

$$\mathcal{E}_q(\boldsymbol{y}) = -\log \sum_{\boldsymbol{x}' \in \mathcal{X}} q(\boldsymbol{y}|\boldsymbol{x}') \exp(-\mathcal{E}(\boldsymbol{x}')) \tag{2.63}$$

and computing its expected difference from the true energy $\mathcal{E} \propto p_{\text{data}}$ in expectation over p_{data} yields the concept of ED, defined as follows:

$$ED_q(p_{\text{data}}, \mathcal{E}) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\mathcal{E}(\boldsymbol{x})] - \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})}[\mathcal{E}_q(\boldsymbol{y})] \tag{2.64}$$

The authors of [37] show that the minimiser of Equation 2.64, say $\mathcal{E}^*$, satisfies $p_{\text{data}} \propto \exp(-\mathcal{E}^*)$, as desired. In practice, the method uses batches of N data samples $\boldsymbol{x}^i_{\text{data}}$, each with a perturbed counterpart $\boldsymbol{y}_i \sim q(\boldsymbol{y_i}|\boldsymbol{x_i})$. For each $\boldsymbol{y}_i$, M 'approximate recoveries' $x^{i,j}_{\sim}$ are computed – for instance by perturbing $\boldsymbol{y}^i$ again with q – that serve to estimate the contrastive potential induced by q. For a given parameterised energy function function $\mathcal{E}_\phi$, the EBM is then trained by minimising an approximation of the ED defined above, according to the following objective

$$\mathcal{L}_{q,M,w}(\mathcal{E}_\phi) := \frac{1}{N} \sum_{i=1}^{N} \log \left(w + \sum_{j=1}^{M} \exp(\mathcal{E}_\phi(\boldsymbol{x}^i_{\text{data}}) - \mathcal{E}_\phi(\boldsymbol{x}^{i,j}_{\sim})) \right) - \log(M) \tag{2.65}$$

where ω is added as a deterministic lower bound to stabilise training, and the logsumexp trick is used. Empirically, training with ED is significantly faster than training with CD, and performs well on synthetic data. However, results on (discrete) image generation have yet to exhibit comparable quality.

Chapter 3

Conditional Velocity Matching and Discrete Flow Samplers

In this chapter we describe the derivation of the proposed data-based **conditional velocity matching** (CVM) regression objective $\mathcal{L}_{CVM}$ (Equation 3.12), as well as its data-free extension termed **energy conditional velocity matching** (ECVM) objective $\mathcal{L}_{ECVM}$ (Equation 3.13). We also provide a cross-entropy based alternative, $\mathcal{L}_{EDFM}$ (Equation 3.15), which adapts Discrete Flow Matching [6] to the data-free case. We provide a thorough evaluation of all techniques at the end of this chapter. After building a solid understanding for our toolkit, we shall then apply it to the training of EBMs in Chapter 4.

3.1 Setting

We will first review the setting, which is derived from that in [6]. We consider the task of transforming a source distribution $p_{\text{source}}(\boldsymbol{x}_0) = p_0(\boldsymbol{x}_0)$ to a target distribution $p_1(\boldsymbol{x}_1)$. Assume that the sample space is discrete and D-dimensional, i.e. $\boldsymbol{x}_t \in \mathcal{X} = \{1, \cdots, S\}^D$, where S indicates the number of states that each dimension can assume, and naturally $t \in [0, 1]$. We model the transformation from p_0 to p_1 via a probability path p_t as a CTMC. We assume that generation from the CTMC is via sampling each dimension independently from a generating probability velocity $u_t^i(z_t^i, \boldsymbol{x}_t)$ starting at $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x}_0)$ according to the following Euler discretisation

$$x_{t+h}^i \sim \delta_{x_t^i}(\cdot) + h u_t^i(\cdot, \boldsymbol{x}_t) \tag{3.1}$$

for some step size h. We follow Definition L.1, given in [6], in saying that a probability path $p_t(\boldsymbol{x}_t)$ is generated by the velocity $u_t^i(\cdot, \boldsymbol{x}_t)$, if sampling according to Equation 3.1 with sufficiently small $h > 0$ yields samples $\boldsymbol{x}_{t+h} \sim p_{t+h} + o(h)$. We generally require that Equation 3.1 define a valid probability mass function, which amounts to $\sum_{z^i} u_t^i(z^i, \boldsymbol{x}_t) = 1$ and $u_t^i(z^i, \boldsymbol{x}_t) \geq 0 \; \forall z^i \neq x^i$. Finally, we may define the general joint distribution of $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ as a conditional or unconditional coupling, $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1) = p_0(\boldsymbol{x}_0|\boldsymbol{x}_1)p_1(\boldsymbol{x}_1)$ or $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1) = p_0(\boldsymbol{x}_0)p_1(\boldsymbol{x}_1)$. This joint distribution is used in sampling data pairs $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ for training. We will generally assume unconditional coupling. The core objective of all methods discussed in this chapter is learning a generating probability velocity $u_t^i(z_t^i, \boldsymbol{x}_t)$ to generate samples from p_1, which importantly may be a *data* distribution, or the distribution implicitly defined by a target *energy* function $\mathcal{E}(\boldsymbol{x}_1)$, by sampling according to Equation 3.1 starting from p_{source}.

3.1.1 Constructing Generating Velocities

We can express the **probability path** p_t in terms of conditional probability paths by taking expectation over the joint distribution of the coupling $(\boldsymbol{x}_0, \boldsymbol{x}_1)$, where we make an additional factorisation (or conditional independence) assumption. We write:

$$p_t(\boldsymbol{x}_t) := \mathbb{E}_{\pi(\boldsymbol{x}_0,\boldsymbol{x}_1)} p_t(\boldsymbol{x}_t|\boldsymbol{x}_1,\boldsymbol{x}_0), \text{ where } p_t(\boldsymbol{x}_t|\boldsymbol{x}_1,\boldsymbol{x}_0) = \prod_i^D p^i_{t|1}(x^i_t|\boldsymbol{x}_0,\boldsymbol{x}_1) \tag{3.2}$$

Note further that we can express the **generating probability velocity** at a given time t for a current state $\boldsymbol{x}_t$ as an expectation over $p(\boldsymbol{x}_0, \boldsymbol{x}_1|\boldsymbol{x}_t) = \frac{\boldsymbol{p}(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)\pi(\boldsymbol{x}_0,\boldsymbol{x}_1)}{p_t(\boldsymbol{x}_t)}$:

$$u^i_t(z^i, \boldsymbol{x}_t) = \mathbb{E}_{p(\boldsymbol{x}_0,\boldsymbol{x}_1|\boldsymbol{x}_t)} u^i_t(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1) \tag{3.3}$$

As we shall see, for our proposed objectives CVM (Equation 3.12), ECVM (Equation 3.13) and EDFM (Equation 3.15) to work in practice, we will require access to a closed form conditional velocity $u^i_t(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)$ that is known to generate $p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)$. If this is the case then Theorem L.1 tells us that $u^i_t(z^i, \boldsymbol{x}_t)$ (3.3) will generate $p_t(\boldsymbol{x}_t)$ (3.2). A flexible framework defining such closed forms is given by Theorem L.2. Specifically, if we have a conditional path of type

$$p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1) = \prod_{i=1}^N p^i_t(x^i|\boldsymbol{x}_0,\boldsymbol{x}_1) \tag{3.4}$$

$$\text{with } p_t(x^i_t|\boldsymbol{x}_0,\boldsymbol{x}_1) = \sum_{j=1}^m \kappa^{i,j}_t w^j(x^i|\boldsymbol{x}_0,\boldsymbol{x}_1), \tag{3.5}$$

$$\text{where } \sum_j \kappa^{i,j}_t = 1,\ \kappa^{i,j}_t \geq 0 \text{ and } p_t(x^i_t|\boldsymbol{x}_0,\boldsymbol{x}_1) = \delta_{\boldsymbol{x}_t}(x^i) \text{ for } t \in \{0,1\} \tag{3.6}$$

then it can be generated by

$$u^i_t(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1) = \sum_{j=1}^m a^{i,j}_t w^j(z^i|\boldsymbol{x}_0,\boldsymbol{x}_1) + b^i_t \delta_{\boldsymbol{x}_t}(z^i) \tag{3.7}$$

$$\text{where } a^{i,j}_t = \dot{\kappa}^{i,j}_t - \frac{\kappa^{i,j}_t \dot{\kappa}^{i,\ell}_t}{\kappa^{i,\ell}_t},\ b^i_t = \frac{\dot{\kappa}^{i,\ell}_t}{\kappa^{i,\ell}_t} \text{ and } \ell = argmin_{j\in[m]} \left[\frac{\dot{\kappa}^{i,j}}{\kappa^{i,j}_t}\right] \tag{3.8}$$

In the above, each $w^j(x^i|\boldsymbol{x}_0,\boldsymbol{x}_1)$ represents some probability density in its own right. The key realisation is that $u^i_t(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)$ can be explicitly constructed under the assumed conditional path, and therefore makes the aforementioned learning objectives tractable. Subsection 3.1.2 illustrates how such conditional paths and there corresponding conditional velocities can be realised. As described in Section 2.3.5, it is easy to obtain the velocity field for sampling backward in time by assigning $\ell = \text{argmax}_{j\in[m]} \left[\frac{\dot{\kappa}^{i,j}}{\kappa^{i,j}_t}\right]$ [6].

3.1.2 Source Distributions and Conditional Paths

Within this framework, the two variable elements — beyond the coupling $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1)$ which we already assume to be unconditional — are the source distribution p_{source} and the conditional path $p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)$. We will now outline the choices that are considered in this project.

Source Distributions We align with the literature on discrete flows, particularly [7] and [6], in considering the 'mask' and the uniform source distributions. The mask distribution is given by $p_{\text{mask}} = \delta_M(\boldsymbol{x}_0)$, where $M \notin \mathcal{X}$ is an artificial state that each dimension can assume which must not appear in the training data. The uniform distribution $p_{\text{uni}} = U[\boldsymbol{x}_0; \mathcal{X}]$, is simply a uniform distribution over all possible states in the sample space. In addition, we consider the less conventional choice of $p_{\text{source}} = p_{\text{data}}$, which to our knowledge has not been applied in the context of discrete flows. Extending this idea, we also consider the case $p_0 = p'_{\text{data}}$ and $p_1 = p_{\text{data}}$, where p'_{data} is a second data distribution, so that the sampling process represents a transformation from one data distribution to another. It is worth noting that for a symmetric choice of path and coupling, the relationship between source and target distribution can be inverted at generation time by sampling backwards from $t = 1$, as long as the discrete was set up to enable bidirectional sampling at training time.

Conditional Paths For the conditional paths, we consider three types of convex interpolants between p_{source} and p_{data}, as well as a third path that additionally introduces uniform noise to the conditional path for $t \in (0, 1)$, see Table 3.1 and Figure 3.1. For illustration purposes, let us first consider the simple example of linear interpolation between source and data in the forward sampling scenario, which is adapted from [6] by fixing a linear scheduler. In this case, we have $m = 2$ with $w^1 = \delta_{\boldsymbol{x}_1}(x^i)$ and $w^2 = \delta_{\boldsymbol{x}_0}(x^i)$, and

$$p_t(x^i|\boldsymbol{x}_0, \boldsymbol{x}_1) = t\delta_{\boldsymbol{x}_1}(x^i) + (1-t)\delta_{\boldsymbol{x}_0}(x^i) \tag{3.9}$$

where we have set $\kappa_t^1 = t$ and $\kappa^2 = 1-t$. We therefore have (in the forward sampling scenario) $\ell = 2$, $\dot{\kappa}_t^1 = 1$ and $\dot{\kappa}_t^2 = -1$. Substituting into Equation 3.7 then yields the corresponding conditional forward velocity which simplifies to the following:

$$u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1) = \frac{1}{1-t}\left[\delta_{\boldsymbol{x}_1}(z^i) - \delta_{\boldsymbol{x}_t}(z^i)\right]. \tag{3.10}$$

	conditional path		**forward velocity**		**backward velocity**	
type	w^j	k^j	a^j	b	a^j	b
linear	$w^1 = \delta_{\boldsymbol{x}_1}$ $w^2 = \delta_{\boldsymbol{x}_0}$	$\kappa^1 = t$	$a^1 = \frac{1}{1-t}$	$-a_1$	$a_2 = \frac{-1}{t}$	$-a_2$
quadratic		$\kappa^1 = t^2$	$a^1 = \frac{2t}{1-t^2}$		$a_2 = \frac{2}{t}$	
cubic		$\kappa^1 = 2t^3 - 3t^2 + 2t$	$a^1 = \frac{1-4t}{2t^2-t+1} + \frac{1}{1-t}$		$a_2 = \frac{3-4t}{2t^2-3t+2} - \frac{1}{t}$	
quadratic with noise	$w^1 = \delta_{\boldsymbol{x}_1}$ $w^2 = \mathcal{U}[\mathcal{X}]$ $w^3 = \delta_{\boldsymbol{x}_0}$	$\kappa^1 = t^2$ $\kappa^2 = t - t^2$	$a^1 = \frac{t(2-t)}{1-t}$ $a^2 = 1 - t$	$\frac{-1}{1-t}$	$a_2 = -1$ $a_3 = \frac{t-2}{t}$	$\frac{2}{t}$

Table 3.1: Table of specifications considered for Equation 3.7. Note that $\kappa^j = 1 - \sum_{k \neq j} \kappa^k$ and $a_i = 0$ unless otherwise stated. The cubic path has proven effective on language modelling tasks in [6]. They also mention *some* noise-based path, but without specification.

3.2 Velocity Matching Objectives

We are now ready to turn to the derivation of the two alternative objective functions for learning the generating velocities of discrete flows from their conditional probability paths, which constitute a core contribution of this project. At the heart of our approach lies the idea to directly parameterise the generating velocity, instead of constructing it at sampling time from learned posteriors of the components w^j. This approach is inspired by CFM [13].

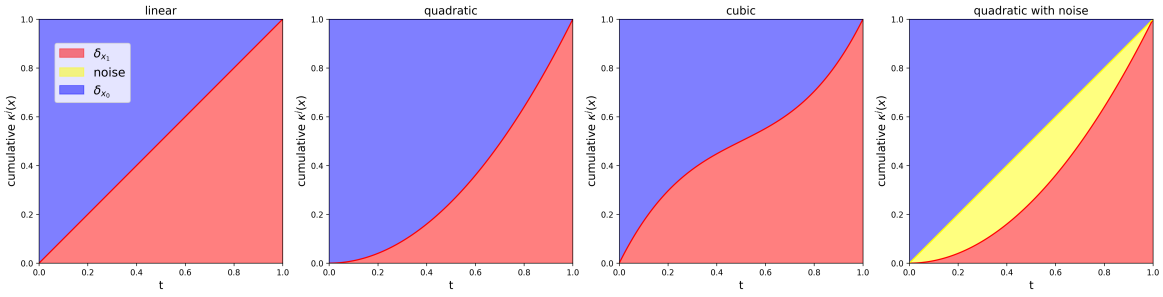


Figure 3.1: Cumulative probability of schedulers by path type. Example: For the linear scheduler, the conditional density at $t = 0.5$ is given by $p_{0.5}(x_t^i|\boldsymbol{x}_0, \boldsymbol{x}_1) = 0.5\delta_{x_1^i} + 0.5\delta_{x_0^i}$.

3.2.1 Velocity Matching

In the hypothetical scenario where $u_t^i(z^i, \boldsymbol{x}_t)$ is known, and we have access to samples from the generated path $p_t(\boldsymbol{x})$, the generating probability velocity can be learned as a neural network $\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta)$ by minimising a simple regression objective that we term velocity matching (VM):

$$\mathcal{L}_{\text{VM}}(\theta) := \mathbb{E}_{\mathcal{U}[t;0,1],p_t(x_t)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t)\|_2^2. \tag{3.11}$$

It is evident that this is minimised for $\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) = u_t^i(z^i, \boldsymbol{x}_t)$, since $\|\cdot\|_2^2$ is non-negative.

3.2.2 Conditional Velocity Matching

However, $u_t^i(z^i, \boldsymbol{x}_t)$ is not generally known and p_t cannot easily be sampled from. Note now that Equation 3.7 enables us to construct the generating velocity conditional on pairs of samples drawn from our coupling distribution $\pi(\boldsymbol{x}_0, \boldsymbol{x}_1)$. We can show that the following Conditional Velocity Matching (CVM) objective is equivalent to Equation 3.11 up to an additive constant, and the two objectives therefore have proportional gradients. This is formally captured in Proposition 1 and a proof is provided in A.1.

$$\mathcal{L}_{\text{CVM}}(\theta) := \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1),\pi(\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2. \tag{3.12}$$

Proposition 1 *The velocity matching and conditional velocity matching objectives have equal gradients, i.e.* $\nabla_\theta \mathcal{L}_{\text{VM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CVM}}(\theta)$.

3.2.3 Energy Conditional Velocity Matching

Energy conditional velocity matching generalises the conditional rate matching objective to the case where no training data is available, and instead we have access to an energy function $\mathcal{E}(\boldsymbol{x}_1) = -\log p_1(\boldsymbol{x}_1) - \log Z$, where $Z = \sum_{\mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}_1))$ is the normalising constant. To this end, we propose the energy conditional velocity matching objective given by

$$\mathcal{L}_{\text{ECVM}}(\theta) :=$$
$$\mathbb{E}_{\mathcal{U}[t;0,1],\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{q_1(\boldsymbol{x}_1)} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2 \tag{3.13}$$

where q_1 is a tractable distribution that is easy to sample from, and $\hat{\pi} := p_0(\boldsymbol{x}_0)q_1(\boldsymbol{x}_1)$ is the corresponding coupling. In A.3 we proof Proposition 2, which states that the gradients of this objective and the original velocity matching objective (3.11) are proportional. It turns out that the proportionality factor c is simply Z^{-1}.

Proposition 2 *The velocity matching and energy conditional velocity matching objectives have equal gradients up to multiplicative constant, i.e.* $\nabla_\theta \mathcal{L}_{\text{VM}}(\theta) = c\nabla_\theta \mathcal{L}_{\text{ECVM}}(\theta)$.

This in principle allows learning the CTMC without access to samples, provided that the proposal distribution q_1 is close enough to p_1 to avoid excessive variance in the importance weights $\frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{q_1(\boldsymbol{x}_1)}$. We term this model a **discrete flow sampler** (DFS), from which samples are obtained by approximating the flow with the Euler method as in Equation 3.1. Note though that in the context of EBM training we will use the term DFS losely for any learned discrete flow that is designed to sample from some distribution, whether trained on energy or data. Despite the apparent solution for the lack of data samples presented by this importance sampling approach, we shall see that especially in high-dimensions this approach is faced with significant challenges. We further note that empirically we found it beneficial to re-weight the loss by a factor of $(1-t)$ for both CVM and ECVM, which we omit in all equations.

Self-normalisation By applying self-normalisation to the importance sampling weights in Equation 3.13 — as outlined in Section 2.1.3 — we can correct for the Z^{-1} factor, since $Z = \mathbb{E}_{q_1(\boldsymbol{x}_1)} \exp(-\mathcal{E}(\boldsymbol{x}_1))/q_1(\boldsymbol{x}_1)$. Recall however, that the corresponding MC approximation of this expectation will only be asymptotically unbiased [4].

3.3 Energy Discrete Flow Matching

Besides proposing the above objective designed to directly learn the generating velocity, we further develop a loss for learning the DFS that is adapted from the loss used in DFM [6]. Recall that contrary to our velocity matching approach, this basically amounts to directly learning the posterior probabilities $\hat{w}_t^j(z^i, \boldsymbol{x}_t)$ via a cross-entropy loss. Given the estimated posteriors $\hat{w}_t^j$, the generating velocity can be constructed at sampling time as follows:

$$u_t^i(z^i, \boldsymbol{x}_t) = \sum_{j=1}^{m} a_t^{i,j} \hat{w}_t^j(z^i, \boldsymbol{x}_t) + b_t^{i,j} \delta_{\boldsymbol{x}_t}(z^i) \tag{3.14}$$

Our objective, given in Equation 3.15, adapts this to the data-free scenario of learning to sample from an energy function. As in Subsection 3.2.3, we couple samples from the source distribution with samples from a proposal distribution q_1, instead of samples from p_1.

$$\mathcal{L}_{EDFM} = \sum_{j\in[m], i\in[N]} \mathbb{E}_{t,\hat{\pi}(x_0,x_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1),w^j(y_j^i|\boldsymbol{x}_0,\boldsymbol{x}_1)} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{q_1(\boldsymbol{x}_1)} \log \hat{w}_t^j(y_j^i, \boldsymbol{x}_t; \theta) \tag{3.15}$$

Analogously to Proposition 2, we present the following proposition, proved in Section A.3. As before, $c = Z^{-1}$. We can apply self-normalisation to $\mathcal{L}_{EDFM}$ in the same way as to $\mathcal{L}_{ECVM}$ and with the same finite-sample limitations.

Proposition 3 *The velocity matching and energy conditional velocity matching objectives have equal gradients up to a multiplicative constant, i.e.* $\nabla_\theta \mathcal{L}_{\text{DFM}}(\theta) = c\nabla_\theta \mathcal{L}_{\text{EDFM}}(\theta)$.

Finally, we summarise the different objectives discussed in this chapter in Table 3.2.

	Regression	**CE**
Data-based	CVM	DFM [6]
Energy-based	ECVM	EDFM

Table 3.2: CVM, ECVM, DFM and EDFM by availability of data and loss type.

3.4 Evaluation I

This section is dedicated to the evaluation of the discrete flow learning toolkit described in this chapter. The insights from this evaluation will guide the choices considered in Chapter 4, which focuses on the use of discrete flows in EBM training. A second evaluation section can be found at the end of that chapter (4.5). We begin with a brief discussion of the data and evaluation metrics used, before systematically considering a variety of set-ups for learning discrete flows.

3.4.1 Training Data

We restrict our evaluation to binary data of different dimensions. This includes discretised synthetic data modified from distributions provided in the `sklearn` library, as well as discretised images from the MNIST dataset of hand-drawn digits, the Omniglot dataset of hand-written characters and the Caltech Silhouettes dataset containing the shapes of a range of objects.

3.4.2 Synthetic Data

The data generation for the seven synthetic datasets follows that of [36] which follows that of [38]. The basis for each dataset is two-dimensional continuous data from the `sklearn` library generated with noise parameter 0.1, which is then transformed to a 32-bit vectors using Gray code [39]. Gray code turns the continuous samples $\hat{x} := [\hat{x}_1, \hat{x}_2] \in \mathbb{R}^2$ into discrete samples $\boldsymbol{x} \in \{0,1\}^{32}$ by quantising $\hat{x}_1$ and $\hat{x}_2$ into 16-bit representations which are then concatenated. Samples of the datasets in the 2D plane are depicted in Figure 3.2.

Figure 3.2: Synthetic datasets: swissroll, circles, moons, 8gaussians, pinwheel, 2spirals, checkerboard.

3.4.3 Discrete Image Datasets

The MNIST dataset is a grayscale image dataset of 60,000 training samples and 10,000 test samples of handwritten digits (0-9). Each image has dimensions 28x28. Pixels are binarised from the original grayscale in two ways. The **static MNIST** version is binarised at the decision boundary 0.5. The stochastic **dynamic MNIST** is obtained by treating each pixel as a bernoulli variable parameterised by its intensity, which is then sampled. The **Omniglot** dataset contains grayscale images of 1,623 hand-written character classes from 50 different alphabets, with each class containing 20 samples. The training set has a size of 19,280 and the test set contains 13,180 images. While the original dataset has dimensions 128x128, we

reduce this to 28x28. The dataset is dynamically binarised in the same way as dynamic MNIST above. The **Caltech Silhouettes** dataset contains binarised silhouettes from 101 object classes, with a training set of size 4,100 and a test set of size 2,307. As with the other datasets, the dimensionality is 28x28. To construct the datasets, we use code from the original paper repository of [12]. Examples from all datasets can be found in Figure 3.3.

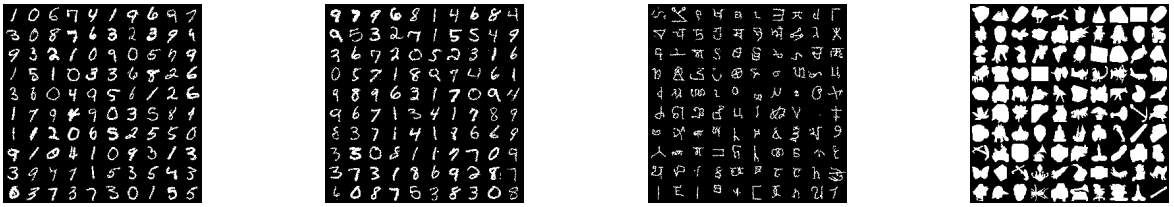


Figure 3.3: Binarised image datasets: Static MNIST, Dynamic MNIST, Omniglot, Caltech Silhouettes.

3.4.4 Metrics

To evaluate the performance of our models on the synthetic and discrete image data, we deploy two core evaluation metrics, the maximum mean discrepancy (MMD) and the negative log likelihood (NLL). Different data and model types require different metrics.

MMD

MMD is a framework for comparing distributions p and q based on their samples first proposed in [40]. It was originally developed jointly with statistical tests of the null hypotheses that $p = q$, but recently has been deployed as a measure of assessing mere closeness of p to q without testing for statistical significance [36][8]. We chose this metric to maintain consistency and comparability with the existing literature. The squared MMD is computed as

$$\mathrm{MMD}^2(X, Y) = \frac{1}{m(m-1)} \sum_{i \neq j} k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{2}{mn} \sum_{i,j} k(\boldsymbol{x}_i, \boldsymbol{y}_j) + \frac{1}{n(n-1)} \sum_{i \neq j} k(\boldsymbol{y}_i, \boldsymbol{y}_j) \quad (3.16)$$

where $X = \boldsymbol{x}_1, ..., \boldsymbol{x}_m$ are samples drawn from p and $Y = \boldsymbol{y}_1, ..., \boldsymbol{y}_n$ are samples drawn from q and k is a kernel function [40]. Aligning with the literature, we choose k to be the exponential hamming kernel, given by $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(\sum_d |x^d - y^d| \cdot \beta)$, where β is a bandwidth parameter [36][8]. We sometimes call this metric H-MMD. We only apply MMD in the case of the synthetic data, due to excessive computational cost for the high-dimensional discrete image data. Since the synthetic data can always be converted back to samples from a continuous distribution in the 2D plane, we supplement H-MMD on the discretised samples with a second metric that uses a radial basis function as kernel. This is given by $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(\|\boldsymbol{x} - \boldsymbol{y}\|^2 \cdot \frac{1}{2\sigma^2})$ where $\boldsymbol{x}$ and $\boldsymbol{y}$ have been converted to their Euclidean representations. We refer to this metric as Euclidean MMD (E-MMD). The reason for departing from the dominant approach in the literature (H-MMD) is that we noted in experiments that in the case of the synthetic data, unless distributions *are* equal, the Hamming MMD is not a reliable indicator of closeness of distributions when viewed in the 2D plane. That is, if two competing distributions p and p' are to be evaluated against a target distribution p^*, unless H-MMD$(p, p^*) \approx$ H-MMD$(p', p^*) \approx 0$, we cannot conclude from H-MMD$(p, p^*) >$ H-MMD(p', p^*) that samples from p' have a distribution that is visually (more) similar to

that of samples from p^* in the 2D plane. We note that this may be due to the highly non-linear nature of the Gray code transformation [8]. We find that E-MMD actually satisfies this requirement, which is unsurprising as it is computed in Euclidean space. We choose $\beta = \sigma = 0.1$, as this is the β chosen in [8], and we find that with this σ a lower E-MMD is correlated with visually closer distributions for the synthetic datasets. In all computations of MMD, we follow [8] and perform 10 repeated estimations with 4000 samples each, which are either generated from the model(s) of interest or drawn from the data distribution.

NLL

NLL is widely used as a metric for evaluating model fit on a held-out test set. Note of course, that this requires the evaluation of likelihoods which in the case of EBMs requires an accurate estimation of the normalising constant, and in the case of discrete flows is not feasible either. When evaluating the NLL of samples from trained EBMs, we estimate the normalising constant Z by importance sampling (Subsection 2.1.3). We can then obtain the likelihood of a sample $\boldsymbol{x}$ as $\frac{\exp(-\mathcal{E}(\boldsymbol{x}))}{Z}$, or equivalently its negative log likelihood as $-\mathcal{E}(\boldsymbol{x})+Z$. However, for the discrete image data we also use NLL to evaluate samples generated from learned samplers. We do so by passing the samples $\boldsymbol{x}$ to a pre-trained energy $\mathcal{E}$ function for which we have estimated Z. This provides a simple comparative measure to assess sample quality from different models, however we note that it can only give a rough guidance since the measure does not consider distributional closeness – i.e. it can be exploited by a model returning only the mode of the model with which we measure NLL. We find that it generally corresponds to the perceived sample quality where visual representations are available. One key assumption is that the model used for the NLL computation is close to the target likelihood, which is trivially the case when the learned sampler is trained on the same energy function. When estimating normalising constants with importance sampling, we follow [36] in the synthetic data setting and draw 1,000,000 samples from a Bernoulli distribution with parameter 0.5. In the discrete image setting we follow [12] and compute the normalising constant with 300,000 iterations of AIS (2.1.3).

3.4.5 Conditional Velocity Matching

Synthetic Data

Let us first consider our conditional velocity matching (CVM) objective (3.12). Before performing a broad evaluation of the method across datasets, we first elicit the optimal combination of source distribution p_0 and conditional path on the 2spirals dataset.[1] We perform this exercise for our CVM loss and the original DFM objective [6]. We consider p_{uni}, p_{mask} and p_{data} for the source distribution, and linear, quadratic (with and without noise) as well as cubic schedulers for the conditional path interpolating between the source distribution and the data (compare Figure 3.1). We parameterise the generating velocity as an MLP with three hidden layers of dimension 1024 and a Swish activation function [41] on all hidden layers. There is one output layer, except in the case of DFM with the quadratic noise path where we add another layer for noise prediction. We train all models for 2×10^5 iterations with batch size 128. Throughout this paper, unless otherwise stated, we will use $h = 0.05$ as the Euler discretisation parameter. Results are shown in Table 3.3. Recall that H-MMD results are not necessarily reflective of perceived closeness in the 2D plane. We

[1] We perform the same exercise on the checkerboard dataset, which confirms all statements made about the 2spirals case. The results can be found in Appendix C.

observe that in terms of H-MMD, the CVM-trained models outperforms the DFM-trained models consistently. When considering the E-MMD based on the 2D projections of the discrete data, better performance aligns with the perceived closeness of the samples to the target distribution visually. On the basis of this metric, we find that the DFM ouperforms our CVM marginally, and that both CVM- and DFM-trained models achieve the best results for the p_{mask} source paired with the linear conditional path. Generations can be found in Figure 3.4.

Table 3.3: CVM/DFM trained on data; Hamming and Euclidean MMD for combinations of loss, source distribution and conditional path specification; Data: 2spirals; MMD in units of 1×10^{-4}.

	Source	**Hamming MMD ↓**				**Euclidean MMD ↓**			
		linear	**quad.**	**cubic**	**noise**	**linear**	**quad.**	**cubic**	**noise**
CVM	mask	0.524	0.769	**0.234**	0.800	**1.965**	2.730	2.718	3.584
	uni	0.683	0.525	0.751	1.163	3.297	5.723	10.299	5.806
	data	0.706	0.498	0.528	0.691	3.493	6.181	10.598	5.806
DFM	mask	8.143	7.085	8.082	5.892	**1.946**	3.006	2.144	5.764
	uni	**5.554**	8.855	5.908	9.753	3.011	5.553	5.222	6.876
	data	6.691	9.637	5.495	11.173	3.505	5.989	5.750	6.540

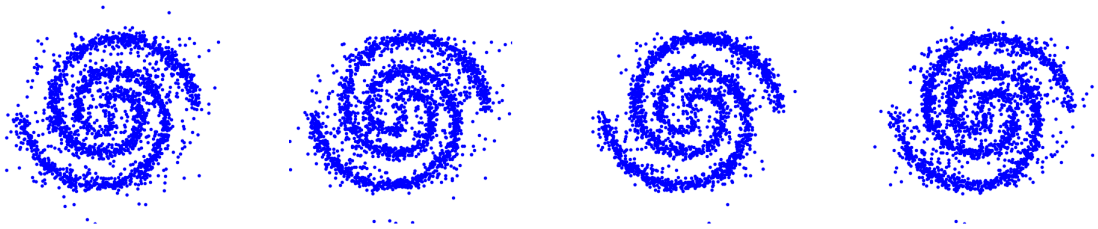


Figure 3.4: Generations from best CVM and DFM configurations according to Hamming and Euclidean MMD on 2spirals; left to right: CVM-mask-lin, CVM-mask-cubic, DFM-mask-lin, DFM-uni-linear.

After determining that CVM and DFM trained flows perform best with the mask source distribution and a linear path on two key synthetic datasets, we now expand our evaluation to the remaining five synthetic datasets (Table 3.4). As a baseline, we trained a GFlowNet [9] where we used the original release code from [8] with a few adaptations. The GFlowNet cannot be trained on data directly, so we trained it on a pre-trained energy model which was learned using ED according to [36]. GFlowNet is a relevant competitor especially regarding the context of EBM training. Note that the GFlowNet component is specified as three-layer MLP with hidden dimensionality at only 256. We see that the GFlowNet does outperform both dicrete flow models in terms of our preferred E-MMD, and also in terms of H-MMD, except for the cirles and moons datasets. To put this result into perspective, we point out that if the GFlowNet were learned directly from data through an intermediate EBM as in EB-GFN, then it would be considerable more expensive to train compared to discrete flows as it requires training an EBM, as well as sampling back-and-forth through the GFlowNet to compute the GFlowNet loss at each iteration. The number of steps for sampling through the

GFlowNet increases linearly with the number of dimensions making it infeasible in truly high-dimensional scenarios – whereas the Euler discretisation for discrete flows is independent of dimensionality. Between the DFM and CVM objectives, we observe that E-MMD values are comparable, with DFM dominating on some datasets such as pinwheel and CVM dominating on others such as checkerboard. We conclude that CVM is a competitive objective in the synthetic data settings compared to DFM.

Table 3.4: Hamming MMD and Euclidean MMD for mask source and linear path across all synthetic datasets. The Hamming MMD values should be interpreted with caution since the differences are not reflected visually – we include them to align with the literature e.g. [36]; MMD in units of 1×10^{-4}.

Metric	Method	2spirals	checkerboard	circles	moons	pinwheel	swissroll	8gaussians
H-MMD↓	CVM	0.524	0.330	**0.488**	**0.385**	1.272	0.704	0.759
	DFM	8.143	9.371	15.723	4.802	6.032	5.815	10.197
	GFN	**0.355**	**0.097**	11.276	17.596	**0.008**	**0.021**	**0.338**
E-MMD↓	CVM	1.965	0.156	**0.390**	1.667	3.019	1.361	0.963
	DFM	1.946	0.827	1.184	**1.499**	1.676	1.526	1.440
	GFN	**1.037**	**0.074**	0.453	3.608	**0.748**	**0.707**	**0.251**

Discrete Image Data

We now repeat the exercise of determining the best combination of source and conditional path for the discrete image data case. We do this on the static MNIST dataset. The generating velocity is now parameterised as a ResNet [42] with six residual blocks with 64 feature maps and two initial projection layers with hidden dimension 1024, one for the sample and one for the time step. We additionally apply sinusoidal time step embeddings [2]. In the case of CVM there is one linear output layer for the forward velocity and optionally a second one for the backward velocity. In the case of DFM, there is one denoising output layer and one additional noise prediction layer in the case of the quadratic noise path. There is optionally a third output layer for source prediction for generating the backward velocity. We evaluate samples with the NLL under a state-of-the-art pre-trained EBM trained with PCD and DLP according to [12]. Note that this is an evaluation method not found in the related literature, but we find that this is suitable and that lower NLL does generally correspond to higher quality generations. All models are trained for 5×10^4 epochs with batch size 100, following [36]. The results are shown in Table 3.5. We observe that under the chosen metric the DFM-trained models outperform CVM consistently. Interestingly, while the linear path with mask source performs well under both objectives, we note that the optimal performance specifications for CVM and DFM differ, being mask source with the cubic path for CVM and uniform source with linear path for DFM. Note that both are symmetric paths. We show example generations for mask source with linear path for both model types, as well as generations from the best specification in each case in Figure 3.5. It is evident that the mask source with linear path are still strong alternatives to the respective best models. The strong performance of the cubic path under CVM further corresponds to the finding in [6], where a cubic scheduler is found to perform best on language modelling tasks, albeit under the DFM objective.

We now expand our analysis to include the remaining discrete image datasets. The

Table 3.5: NLL for DFM, CVM with all source/path specifications on static MNIST. Computed using same pre-trained EBM for all models.

	source	**linear**	**quadratic**	**cubic**	**noise**
CVM	mask	205.681	213.526	**204.906**	254.900
	uni	212.889	219.617	279.179	241.053
	data	208.132	219.617	265.767	238.631
DFM	mask	119.344	154.269	139.069	159.863
	uni	**102.658**	126.125	128.721	142.339
	data	106.244	125.488	122.131	136.182

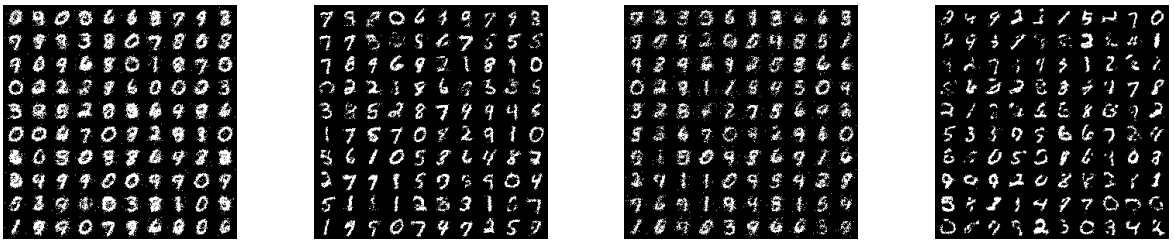


Figure 3.5: Left to right: CVM with mask source and linear path; DFM with mask source and linear path; CVM with mask source and cubic path; DFM with unifrom source and linear path.

results are given in Table 3.6. We used the mask with linear specification as a benchmark for both CVM and DFM, and consider separately the specification that performed best for each on static MNIST. As a baseline, we consider a GFlowNet trained on a pre-trained EBM learned with state-of-the-art PCD using DLP [12]. As before, this is relevant as we will consider EB-GFN as our main baseline for training discrete EBMs with learned samplers. The GFlowNet is parameterised in the same way as in Section 3.4.5, following [8]. We observe that on this high-dimensional problem DFM outperforms GFlowNet, and improves on CVM by a margin. Generally, CVM generations appear slighly 'noisier'. We suspect that longer training might close the gap between the two approaches. Nevertheless, for the purpose of training EBMs fast training is essential to prevent the surrogate sampler from diverging from the EBM which it is designed to sample. We conclude that given the competitive performance of the DFM trained model compared to GFlowNet, discrete flows are sufficiently powerful to be deployed in EBM training on discrete image data. Yet, so far we have assumed the availability of training data. The energy-based case is covered in Subsection 3.4.6.

Table 3.6: NLL for GflowNet and DFM, CVM with three selected specifications across all discrete image datasets. Computed using same pre-trained EBM for all models.

Set-up	**Method**	**stat. MNIST**	**dyn. MNIST**	**Omniglot**	**Caltech**
mask-linear	CVM	205.681	206.691	255.597	279.715
	DFM	119.344	132.061	224.783	174.386
mask-cubic	CVM	204.906	217.538	253.792	270.968
uni-linear	DFM	**102.658**	**103.157**	**198.559**	**132.870**
	GFN	167.921	176.456	211.213	270.278

Transforming Data to Data Lastly, we consider the novel scenario of transforming one data distribution $p_0 = p'_{\text{data}}$ to another $p_1 = p_{\text{data}}$. We consider the case where p_0 is the Omniglot dataset and p_1 is the static MNIST dataset. We test this for our two symmetric paths, linear and cubic. Beyond the forwards generation quality, it is of interest to analyse the impact on performance from training the model to learn the reverse flow additionally to modelling the forward flow. Recall that this requires an additional output layer in both CVM and DFM. The results are depicted in Table 3.7. Note that, perhaps surprisingly, under the linear path, performance slightly improves on the results in Table 3.6. This confirms the theoretic claim made in [6] that the discrete flow framework is suitable for any source distribution. However, we do observe a slight performance drop when the backward velocity is trained simultaneously. This can be explained by some of the model's expressive power being diverted from the forward modelling task to the backward modelling task. Images of source and generation in the forward-only scenario, as well as images of forward and backward generations are shown in Figure 3.6.

Table 3.7: NLL of Omniglot to static MNIST for CVM, DFM and linear and cubic paths. Figures for forward-only model and for forward-and-backward model.

	Forward Only		**Forward and Backward**	
	linear	**cubic**	**linear**	**cubic**
CVM	**203.193**	237.728	226.278	246.267
DFM	**102.442**	114.035	118.569	122.744

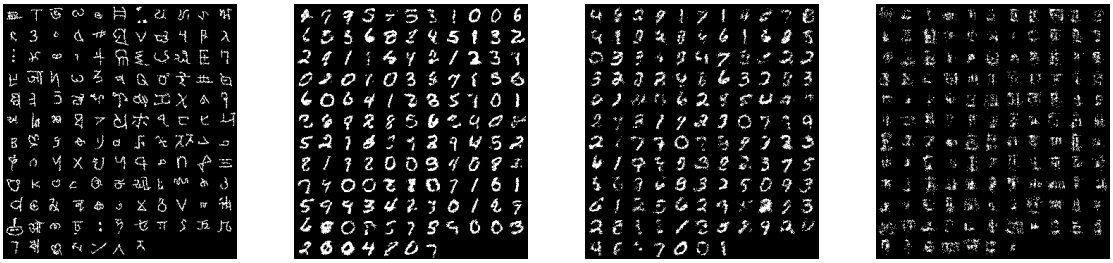


Figure 3.6: Different artefacts from Omniglot to static MNIST modelling; left to right: source data samples, generations from DFM-linear-forward-only model, generations from DFM-linear-forward-and-backward model, backward generations from same model.

3.4.6 Energy Conditional Velocity Matching

Synthetic Data

We will now analyse the performance of our energy-based losses ECVM and EDFM. We parameterise the flow in the same way as in Section 3.4.5 and train for 2×10^5 iterations. The proposal distribution is chosen to be a factorised Bernoulli distribution, where for each dimension the parameter is set to the data mean observed in 10.000 samples. That is, $q_1(\boldsymbol{x}_1) = \prod_{i=0}^{D} Bern(x^i; \bar{x}^i)$. We train all models on a pre-trained EBM learned with ED [36]. We again use GFlowNet as the baseline. We find that the generations from the DFS give a reasonable approximation to the target distribution, see Figure 3.7. Quantitatively, we find that the performance on our preferred E-MMD drops below that in the data-based

case (compare Table 3.4). Increases vary between a factor of roughly $\times 1.5$ (for 8gaussians, EDFM vs. DFM) to roughly $\times 5$ (for moons, EDFM vs. DFM). In terms of H-MMD the picture is mixed, with many increases in the order of $\times 2$, but also one improvement (for circles, EDFM vs. DFM). While GFlowNet dominates our models on E-MMD, on H-MMD our ECVM trained model is competitive on the circles and moons datasets. Performance between EDFM and ECVM is similar, with the velocity matching objective performing much better again on H-MMD. Overall, this deterioration in performance is expected and can be attributed to the limited 'overlap' between the proposal and the target distribution. This is evident from the normalised importance weights, which are heavily skewed towards zero and typically spread in the range $[0, 10]$, see Figure 3.9. Recall that optimal correspondence between target and proposal would imply weights of one. Yet, in the limit $i \to \infty$ we would expect the performance of the model to converge to that trained on data, as we have shown in Subsection 3.2.3.

Table 3.8: Hamming MMD and Euclidean MMD for mask source and linear conditional path specification across all discrete image datasets. All models were trained on the same pre-trained EBM; MMD in units of 1×10^{-4}.

Metric	Method	2spirals	checkerboard	circles	moons	pinwheel	swissroll	8gaussians
H-MMD↓	ECVM	1.631	0.866	**2.145**	**3.894**	2.582	1.534	1.861
	EDFM	18.692	16.206	13.493	16.036	14.699	11.997	17.341
	GFN	**0.355**	**0.097**	11.276	17.596	**0.008**	**0.021**	**0.338**
E-MMD↓	ECVM	5.008	1.017	1.206	7.451	6.083	2.901	1.648
	EDFM	6.391	1.708	2.045	7.026	5.007	2.327	2.026
	GFN	**1.037**	**0.073**	**0.453**	**3.608**	**0.748**	**0.707**	**0.251**

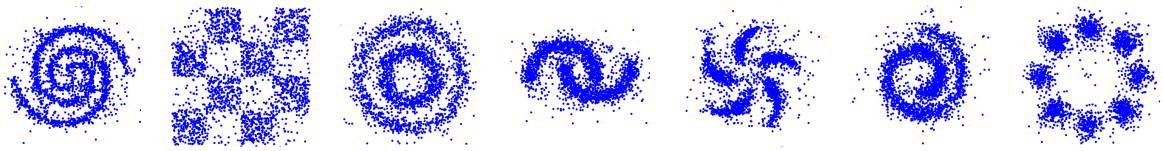


Figure 3.7: ECVM generations for synthetic datasets; left to right: 2spirals, checkerboard, moons, circles, pinwheel, swissroll, 8gaussians.

Discrete Image Data

On the discrete image datasets, we use the intelligence gathered in Section 3.4.5 to choose the cubic path from the mask source for training the ECVM model and the linear path from uniform source for training the EDFM model. Parameters are otherwise the same. The target EBM used is trained with PCD using DLP [12]. As in the synthetic data case, we choose $q_1(\boldsymbol{x}_1) = \prod_{i=0}^{D} Bern(x^i; \bar{x}^i)$. We observe a stark drop in performance when evaluating the generated samples under the target model, as Table 3.9 shows. This is most apparent for the EDFM model, where the data-based version even outperformed GFlowNet (Table 3.6). For both ECVM and EDFM the measured NLL deteriorates most for the Caltech Silhouettes data, which is typically the most challenging to learn. To continue our discussion of importance weights, we note that the weight distribution in Figure 3.9 immediately reveals why the discrete flow cannot learn a good approximation of the target distributions in a reasonable

time frame. Over 95% of weights fall into a the first percentile of the range $[0, 100]$. Thus for each batch of size 100 used in training, the loss is dominated by the particle with the maximum weight and a handful of other samples from the proposal. We want to emphasise that when sampling from this proposal in the synthetic data scenario, samples that have a high likelihood under the data distribution (e.g. points on the spiral shape) come up with reasonable frequency. However, the same proposal producing an image resembling say, a seven, in the case of static MNIST is an extreme tail event. While the theoretical results established in Subsection 3.2.3 hold no less, a finite sample approximation is infeasible in practice. The poor generation quality under this proposal distribution is demonstrated in Figure 3.8.[2]

Table 3.9: NLL for GflowNet and EDFM, ECVM with respective best specifications from the data case across all discrete image datasets trained with Bernoulli proposal with parameter set at the data mean. All models were trained on the same pre-trained EBM.

Set-up	Method	stat. MNIST	dyn. MNIST	Omniglot	Caltech
mask-cubic	ECVM	237.153	230.263	233.864	447.169
uni-linear	EDFM	288.833	279.018	318.588	533.848
	GFN	**167.921**	**176.456**	**211.213**	**270.278**

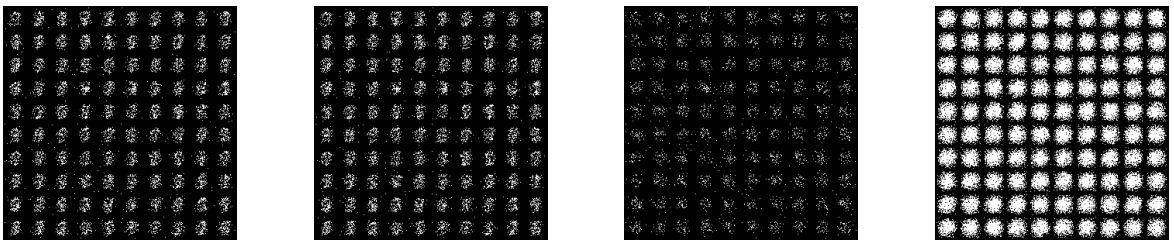


Figure 3.8: Generations of ECVM-trained model (mask source, cubic path); left to right: static MNIST, dynamic MNIST, Omniglot, Caltech Silhouettes.

3.4.7 Summary

We have established that our proposed conditional velocity matching loss exhibits competitive performance compared to the original DFM loss proposed in [6] on synthetic data, and good generation quality on discrete image data, comparable with GFlowNet [9][8]. We have further demonstrated that data-trained DFM beats energy-trained GFlowNet on discrete image generation measured by NLL under the target EBM. We further show that the energy-based ECVM and EDFM perform as expected: well if a reasonable proposal distribution is available (e.g. synthetic data) and rather poorly if the proposal diverges too much from the target distribution (e.g. discrete image data). We conclude that in principle we would expect discrete flows to be suitable generators for negative samples in EBM training under the right parameter settings and with suitable proposals.

[2]Finding a good proposal distribution – with a tractable likelihood – is hard, which is why MCMC methods are dominant in higher-dimensional settings [4]. We propose a (not entirely MCMC-free) framework for exploiting the energy function itself for *unnormalised* likelihood evaluation in the proposal in Chapter 4

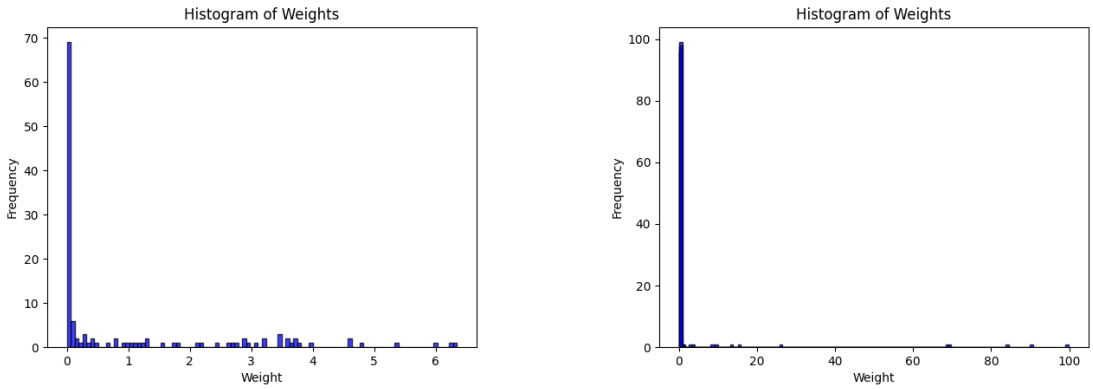


Figure 3.9: Normalised importance weights in ECVM loss; left: 2spirals at iteration 10^5, right: static MNIST at iteration 25×10^3.

Chapter 4

Training Energy-Based Models with Discrete Flows

As described in Section 2.4, the challenge in training EBMs with contrastive divergence is efficient sampling of negative samples from the model distribution. As shown in [8], it is not necessary to run costly MCMC using the the parameterised energy function $\mathcal{E}_\phi$ if a surrogate model can be trained efficiently in an interleaved manner to sample approximately from the EBM at training time. This project originated in the idea to implement this surrogate sampler with a discrete flow based model, such as one trained with DFM [6] or with our energy-based objectives (3.2.3, 3.3) to obtain an MCMC-free training paradigm. In this chapter we explore various avenues founded in theory through which this goal can be approached. We caution here that the approaches face different challenges, and that making our approaches competitive beyond a series of 'toy' problems has so far only been possible with the use of MCMC refinement steps. Note that every method written in terms of ECVM or CVM can be equivalently implemented in terms of EDFM and DFM respectively. A thorough evaluation of the individual methods can be found at the end of this chapter (3.4).

4.1 Contrastive Divergence

Recall from Section 2.4, that the established framework for training EBMs is contrastive divergence (CD). In contrastive divergence, updates to the parameterised energy function, $\mathcal{E}_\phi \propto p_\phi$ are performed proportionally to

$$\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} \nabla_\phi \mathcal{E}_\phi(\boldsymbol{x}) - \mathbb{E}_{p_\phi(\boldsymbol{x}')} \nabla_\phi \mathcal{E}_\phi(\boldsymbol{x}') \tag{4.1}$$

which in practice is approximated by

$$-\frac{1}{N}\left(\sum_{n=1}^{N} \nabla_\theta \mathcal{E}_\theta(\boldsymbol{x}_n) - \sum_{n=1}^{N} \nabla_\theta \mathcal{E}_\theta(\boldsymbol{x}'_n)\right) \tag{4.2}$$

where we sample $\boldsymbol{x}_n \sim p_{\text{data}}$ and $\boldsymbol{x}'_n \sim p_\phi$. What the methods proposed in this chapter are aiming to accomplish, is the amortisation of the latter sampling procedure through sampling from a discrete flow parameterised by θ with implicit distribution p_θ, which is trained simultaneously with the EBM to achieve $p_\theta \approx p_\phi$ throughout the training process. To maintain clearer notation, in what follows we will abstract away from within-batch indices like n above, and the corresponding normalisation by $1/N$.

4.2 EBM-DFS

A first naïve approach, that we label EBM-DFS, is presented in Algorithm 1. The algorithm relies on the hypothesis that the ECVM (3.2.3) trained DFS converges under the utilised proposal distribution q_1, and does so at reasonable speeds, since during training p_ϕ represents a moving target. To accommodate for the potential need for the DFS to 'catch-up', we introduce a time horizon M_{DFS} that determines the number of DFS training steps per training step of the EBM. In theory, Proposition 2 tells us that the parameterised flow will converge to $u_t^i(z^i, \boldsymbol{x}_t)$, and therefore near the optimum $p_\theta \approx p_\phi$, for a fixed p_ϕ. We will see that this is the core assumption underlying all approaches discussed in this chapter in some way.

Algorithm 1 EBM-DFS

Require: Training dataset $\{\boldsymbol{x}_{data}\}$, DFS $u_t^i(z^i, \boldsymbol{x}_t; \theta)$, EBM $\mathcal{E}_\phi$, iterations of DFS per iteration of EBM M_{DFS}

Initialise DFS's θ and EBM's ϕ s.t. $p_\theta \approx p_\phi$
repeat
  for $m \leftarrow 1$ **to** M_{DFS} **do**
    Sample $\boldsymbol{x}_1 \sim q_1(\boldsymbol{x}_0)$, $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x}_1)$, $t \sim \mathcal{U}[0,1]$, $\boldsymbol{x}_t \sim p_t(x_t|\boldsymbol{x}_0, \boldsymbol{x}_1)$
    Compute $\mathcal{L}_{\text{ECVM}}$ ($\mathcal{L}_{\text{EDFM}}$)
    Update DFS via gradient step on $\mathcal{L}_{\text{ECVM}}$ ($\mathcal{L}_{\text{EDFM}}$)
  end for
  Sample batch $\boldsymbol{x}_{\text{data}} \sim p_{\text{data}}$
  Generate batch $\boldsymbol{x}_1$ with Algorithm 2
  Update $\mathcal{E}_\phi$ using with gradient of $\mathcal{E}_\phi(\boldsymbol{x}_{\text{data}}) - \mathcal{E}_\phi(\boldsymbol{x}_1)$
until convergence

Algorithm 2 DFS Generation

Require: DFS $u_t^i(z^i, \boldsymbol{x}_t; \theta)$, EBM $\mathcal{E}_\phi$, time step size h

$t \leftarrow 0$
Sample batch $\boldsymbol{x}_0 \sim p_0$
while $t < 1$ **do**
  Sample $x_t^i \sim Cat\left(\delta_{x_t^i}(\cdot) + h u_t^i(\cdot, \boldsymbol{x}_t; \theta)\right)$
  $t \leftarrow t + h$
end while
return $\boldsymbol{x}_1$

We label this approach as naïve, since in high dimensions the specification of a static proposal distribution q_1 whose density overlaps p_θ substantially throughout training is challenging. The remainder of this chapter is dedicated to adaptations of this algorithm aimed at alleviating the shortcomings of a fixed proposal q_1. One approach, termed 'bootstrapped proposal' exploits the assumption that $p_\theta^k \approx p_\phi^k$ (at every iteration k) to infer that if $\boldsymbol{x} \sim p_\theta^k$, then we may evaluate the unnormalised likelihood — an unnormalised q_1^k so to speak — of $\boldsymbol{x}$ as $\mathcal{E}_\phi^k(\boldsymbol{x})$. Motivated by the success of using GFlowNet to perturb data with 'back-and-forth' sampling and a 'heuristic' MH step in [8], we further test a similar 'back-and-forth' style algorithm with discrete flows. We defer a final approach that swaps the role of 'leader' and 'follower' to Appendix B, due to a lack of interesting results.

Initialising $p_\theta \approx p_\phi$ All algorithms discussed in this section require that initially we have $p_\theta \approx p_\phi$. This requirement could be understood as a base case of an inductive argument, and its violation is likely to propagate through the training process, potentially leading the EBM and the DFS to diverge entirely. In practice, the difficulty of satisfying this requirement depends on the choice of source distribution for the discrete flow, and on whether the EBM is randomly initialised, or modelled as the difference to some base distribution, such as the negative log of a factorized Bernoulli distribution in the binary case. Even for non-uniform initialisations of the EBM, we can achieve this approximate equality within a reasonable number of warm-up iterations in which we fix the EBM and train the discrete flow with a uniform proposal distribution, or a Bernoulli proposal parameterised with the data mean, and $\mathcal{L}_{ECVM}$ until convergence. In higher dimensions more sophisticated techniques may be necessary to achieve this.

4.3 EBM-DFS with Bootstrapped Proposal

The dynamic 'bootstrapped proposal' is our core proposition to overcome the lack of a tractable q distribution. The results in Section 3.4 have clearly shown that static proposals are infeasible in higher-dimensional settings. The core realisation of the bootstrapped proposal is that under the assumption that $p_\theta^k \approx p_\phi^k$ at each iteration k, we can use $\mathcal{E}_\phi^k(\boldsymbol{x}_1)$ as an unnormalised likelihood of a sample drawn from p_θ^k using the DFS, when computing ECVM with respect to $\mathcal{E}_\phi^{k+1}$. That is, we can use the DFS's samples together with the previous steps' energy, to perform importance sampling in ECVM. While the idea of using the DFS's distribution as q may seem trivial at first, recall that one of the core shortcomings of discrete flows modelled as an Euler discretisation of a CTMC is the lack of a tractable likelihood: The evaluation of likelihoods, i.e. computing $p_\theta(\boldsymbol{x}_1)$ for a given sample $\boldsymbol{x}_1$ typically requires integrating *all* possible paths by which $\boldsymbol{x}_1$ could have been obtained from *any* sample $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x})$. That is $p_\theta(\boldsymbol{x}_1) = \sum_{\boldsymbol{x}_0} \sum_j p(\tau^j_{\boldsymbol{x}_0 \to \boldsymbol{x}_1}) p_0(\boldsymbol{x}_0)$, where $\{\tau^j_{\boldsymbol{x}_0 \to \boldsymbol{x}_1}\}$ is the set of all possible paths between x_0 and x_1 indexed by j. The cardinality of this set increases exponentially with the number of steps in the chosen Euler discretisation $\frac{1}{h}$ and in the number of dimensions D. It also grows polynomially with the number of states S that each dimension can assume. But how does the assumption that $p_\theta^k \approx p_\phi^k$ help us if we do not have access to p_ϕ^k either? To realise how the approach might still be viable, consider an *unnormalised* adaptation of ECVM, which we indicate by an asterisk:

$$\mathcal{L}_{\text{ECVM}^*}(\theta) :=$$
$$\mathbb{E}_{\mathcal{U}[t;0,1],\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{\exp(-\mathcal{E}_{q_1}(\boldsymbol{x}_1))} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2 \quad (4.3)$$

where $\mathcal{E}_{q_1} = Z_{q_1} q_1$, is the energy function corresponding to some proposal q_1 and Z_{q_1} is the corresponding normalising constant. It is easy to see that theoretically, the resulting gradient is proportional to that of the original $\mathcal{L}_{ECVM}$, and therefore that of $\mathcal{L}_{VM}$. We capture this formally in Proposition 4, and provide the proof in Section A.4. To avoid redundancy, we only state informally that of course the analogous statement can be made for an unnormalised $\mathcal{L}_{EDFM^*}$.

Proposition 4 *The velocity matching and unnormalised energy conditional velocity matching objectives have equal gradients up to a multiplicative constant c, i.e. $\nabla_\theta \mathcal{L}_{\text{VM}}(\theta) = c \nabla_\theta \mathcal{L}_{\text{ECVM}^*}(\theta)$, where $c = \frac{Z_{q_1}}{Z}$ is the ratio of normalising constants of $\mathcal{E}_{q_1}$ and $\mathcal{E}$.*

We further point out that $\frac{Z_{q_1}}{Z} = (\mathbb{E}_{q_1(\boldsymbol{x}_1)} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{\exp(-\mathcal{E}_{q_1}(\boldsymbol{x}_1))})^{-1}$, so that by dividing the loss by the average importance weight, we can optionally adjust for this factor. As discussed previously, when approximating this quantity with minibatches the approximation is only asymptotically unbiased [4]. Based on this, now consider Algorithm 3, where essentially we dynamically set $q_1 = \mathcal{E}_\phi^k$ at each iteration. Note that this requires computing $\mathcal{E}_\phi^k(\boldsymbol{x}_1)$ prior to performing the gradient step on $\mathcal{E}_\phi$. Note further that when we sample M_{DFS} batches from the discrete flow, we may use all the generated batches $\boldsymbol{x}_{m,1}$ as negative samples for the EBM step. However, instead of updating the EBM with each batch in sequence we must accumulate the gradient before performing a gradient step, since otherwise we would break the lockstep assumption $p_\theta^k \approx p_\phi^k$ rendering the negative samples invalid. Alternatively, one can implement the algorithm so that the negative samples used to train the EBM are distinct from those used as samples in DFS. Intuitively, this may prevent a potential misalignment between the EBM and the DFS, because the DFS is otherwise trained on those samples (in step 13) for which the energy has just been pushed upward (in step 10). Theoretically of course, the validity of the importance sampling approach is not violated by this potential misalignment.

Algorithm 3 EBM-DFS with Bootstrapped Proposal

Require: Training dataset $\{\boldsymbol{x}_{data}\}$, DFS $u_t^i(z^i, \boldsymbol{x}_t; \theta)$, EBM $\mathcal{E}_\phi$, M_{DFS} iterations of DFS per iteration of EBM, number of optional DMALA steps M_{DMALA}

1: Initialise DFS's θ and EBM's ϕ s.t. $p_\theta \approx p_\phi$
2: $k \leftarrow 0$
3: **repeat**
4: **for** $m \leftarrow 0$ **to** M_{DFS} **do**
5: Sample batch $\boldsymbol{x}_{\text{data},m} \sim p_{data}$
6: Generate batch $\boldsymbol{x}_{1,m}$ with Algorithm 2
7: **optionally** Refine batch $\boldsymbol{x}_{1,m}$ with M_{DMALA} steps of DMALA (2.2.4)
8: Store $\mathcal{E}_\phi^k(\boldsymbol{x}_{1,m})$
9: **end for**
10: Update $\mathcal{E}_\phi$ via gradient of $\frac{1}{M}\sum_m \mathcal{E}_\phi^k(\boldsymbol{x}_{\text{data,m}}) - \mathcal{E}_\phi^k(\boldsymbol{x}_{1,m})$ (obtain $\mathcal{E}_\phi^{k+1}$)
11: **for** $m \leftarrow 0$ **to** M_{DFS} **do**
12: Sample $t \sim \mathcal{U}[0,1]$, $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x}_0)$, $\boldsymbol{x}_t \sim p_t(x_t|\boldsymbol{x}_0, \boldsymbol{x}_{1,m})$
13: Step DFS via gradient of $\mathcal{L}_{ECVM*}$ ($\mathcal{L}_{EDFM*}$) (with $\mathcal{E}_{q_1}(\boldsymbol{x}_{1,m}) = \mathcal{E}_\phi^k(\boldsymbol{x}_{1,m})$)
14: **end for**
15: $k \leftarrow k + 1$
16: **until** convergence

MCMC Refinement In practice, we observe that the DFS model may collapse when being trained on its own samples. We therefore propose to optionally refine the samples $\boldsymbol{x}_{1,m}$ with M_{DMALA} steps of DMALA (2.2.4) using $\mathcal{E}_\phi^k$. It is worth considering this step in some depth. We can view this step to have three separate beneficial effects that all lead to a reinforcement of the lockstep assumption $p_\theta^k \approx p_\phi^k$. First, the negative sample quality is improved. Here, the DMALA steps can be seen as a short MCMC chain initialised with our DFS, so that the Metropolis adjustment in the limit will guarantee samples distribute according to p_ϕ^k. Second, by the same token, the likelihood of the samples used in ECVM will be more accurate, ensuring the validity of the importance sampling weights. Third, as a second round effect, DFS samples in the next iteration will be a more accurate initialisation to the DMALA steps. Of course, one threat to the validity of our algorithm is that its

hypothetical success is only attributed to the effectiveness of DMALA. We shall therefore carefully monitor the effect of DMALA through an ablation study in the evaluation.

4.4 EBM-DFS with Heuristic MH Acceptance

In this section, we explore a version of EBM-DFS that mimics the interleaved training algorithm in [8] in that it exploit the ability to sample backward and forward from our trained discrete flow to perturb data samples. We also deploy a 'heuristic' MH acceptance step on the thus proposed perturbation with the intention to generate samples that more closely resemble the energy function's distribution. We call this step 'heuristic', as in the same way as the MH acceptance step proposed in [7], it does not strictly satisfy the Metropolis-Hastings framework. While the MH acceptance step in EB-GFN should not be considered exact, it still is reported to be a crucial part of their algorithm [7]. Motivated by the empirical success, we therefore attempt to achieve similar results in an adaptation of said algorithm that replaces the GFlowNet with a data-trained discrete flow. We caution that this approach only showed limited empirical success towards the end of this project, which is why we only consider it briefly in the evaluation and defer further experiments to future work. We present the method in Algorithm 4.

Algorithm 4 EBM-DFS with Heuristic MH Acceptance

Require: Training dataset $\{\boldsymbol{x}_{\text{data}}\}$, DFS $u_t^i(z^i, \boldsymbol{x}_t; \theta)$, EBM $\mathcal{E}_\phi$, M_{DFS} iterations of DFS per iteration of EBM, turning time for back-and-forth sampling $t_{\text{turn},k}$

Initialise DFS's θ and EBM's ϕ s.t. $p_\theta \approx p_\phi$
repeat
  Generate batch $\boldsymbol{x}_1$ with Algorithm 5
  Sample $\boldsymbol{x}_{\text{data}} \sim p_{\text{data}}$
  Update $\mathcal{E}_\phi$ via gradient of $\mathcal{E}_\phi(\boldsymbol{x}_{\text{data}}) - \mathcal{E}_\phi(\boldsymbol{x}_1)$
  for $m \leftarrow 1$ **to** M_{DFS} **do**
    Generate batch $\boldsymbol{x}_1$ with Algorithm 5
    Sample $t \sim \mathcal{U}[0,1]$, $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x}_0)$, $\boldsymbol{x}_t \sim p_t(x_t|\boldsymbol{x}_0, \boldsymbol{x}_1)$
    Step DFS via gradient of $\mathcal{L}_{CVM}$ ($\mathcal{L}_{DFM}$)
  end for
until convergence

Note that in Algorithm 5, which is responsible for generating samples in a 'back-and-forth' perturbation of the data, we make use of both forward sampling and backward sampling. This requires that we are able to predict both the forward velocity field $u^i_{\text{forth},t}(\cdot, \boldsymbol{x}_t; \theta)$ and the backward velocity $u^i_{\text{back},t}(\cdot, \boldsymbol{x}_t; \theta)$, which is not needed in the other EBM training paradigms where generation is exclusively forwards in time. While this does not imply additional training steps, it should be noted that this requires an additional output layer in the network parameterising the discrete flow, as well as a composite loss for the forward and backward loss velocities. As we have seen in Section 3.4.5, this can reduce the expressive power of the forward generative model in high dimensions. As with the other methods, this too can be implemented with the CE-based EDFM loss in Section 3.3, in which case we similarly require the prediction of logits for the source distribution.

Validity of the MH Acceptance Step We briefly discuss here to what extend the MH step performed in Algorithm 5 (proposed in [8]) diverges from the MH rule in Sub-

Algorithm 5 DFS Generation with Heuristic MH Acceptance

Require: Training dataset $\{\boldsymbol{x}_{\text{data}}\}$, DFS $u_t^i(z^i, \boldsymbol{x}_t; \theta)$, EBM $\mathcal{E}_\phi$, turning time for back-and-forth sampling $t_{\text{turn},k}$

$t \leftarrow 1$
Sample $\boldsymbol{x}_1 \sim p_{\text{data}}$
Initialise trajectory probabilities $p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}) = 1$, $p(\tau_{\boldsymbol{x}'_t \to \boldsymbol{x}_t}) = 1$
while $t_{\text{turn},k} < t$ **do**
    Sample $\boldsymbol{x}^i_{t-h} \sim Cat\left(\delta_{x_t^i}(\cdot) - h u^i_{\text{back},t}(\cdot, \boldsymbol{x}_t; \theta)\right)$
    $p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}) \leftarrow p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}) p_\theta(\boldsymbol{x}_t \to \boldsymbol{x}_{t-h})$
    $p(\tau_{\boldsymbol{x}'_t \to \boldsymbol{x}_t}) \leftarrow p(\tau_{\boldsymbol{x}'_t \to \boldsymbol{x}_t}) p_\theta(\boldsymbol{x}_{t-h} \to \boldsymbol{x}_t)$
    $t \leftarrow t - h$
end while
while $t < 1$ **do**
    Sample $\boldsymbol{x}'^i_{t+h} \sim Cat\left(\delta_{x_t^i}(\cdot) + h u^i_{\text{forth},t}(\cdot, \boldsymbol{x}_t; \theta)\right)$
    $p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}) \leftarrow p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}) p_\theta(\boldsymbol{x}_t \to \boldsymbol{x}_{t+h})$
    $p(\tau_{\boldsymbol{x}'_t \to \boldsymbol{x}_t}) \leftarrow p(\tau_{\boldsymbol{x}'_t \to \boldsymbol{x}_t}) p_\theta(\boldsymbol{x}_{t+h} \to \boldsymbol{x}_t)$
    $t \leftarrow t + h$
end while
Compute $\alpha = \frac{\exp(-\mathcal{E}_\phi(\boldsymbol{x}'_1)) p(\tau_{\boldsymbol{x}'_t \to \boldsymbol{x}_t})}{\exp(-\mathcal{E}_\phi(\boldsymbol{x}_1)) p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t})}$
Compute acceptance probability $A = \min(1, \alpha)$
Set negative sample to $\boldsymbol{x}'_1 = \begin{cases} \boldsymbol{x}'_1 & \text{with probability } A \\ \boldsymbol{x}_1 & \text{otherwise} \end{cases}$
return $\boldsymbol{x}'_1$

section 2.2.1. Recall that the MH rule, for some proposal distribution $q(\boldsymbol{x}'|\boldsymbol{x})$ and unnormalised target distribution $\tilde{p}$, defines the acceptance probability of a proposal $\boldsymbol{x}'$ from q as $A(\boldsymbol{x}'|\boldsymbol{x}) = \min(1, \alpha)$ where $\alpha = \frac{\tilde{p}(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{\tilde{p}(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})}$. In our case, the proposal $\boldsymbol{x}'$ is obtained by sampling *some* trajectory $\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}$. However, $p(\tau_{\boldsymbol{x}_t \to \boldsymbol{x}'_t})$ does not correspond to the probability of proposing $\boldsymbol{x}'$ from $\boldsymbol{x}$ since $\boldsymbol{x}'$ can be obtained via *many* trajectories. That is, $q(\boldsymbol{x}'|\boldsymbol{x}) = \sum_j p(\tau^j_{\boldsymbol{x}_t \to \boldsymbol{x}'_t})$, where the set $\{\tau^j_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}\}$ is the set of all possible back-and-forth trajectories between $\boldsymbol{x}_1$ and $\boldsymbol{x}'_1$. Thus, $p(\tau^j_{\boldsymbol{x}_t \to \boldsymbol{x}'_t}) << q(\boldsymbol{x}'|\boldsymbol{x})$ for any j. Nevertheless, in experiments of EB-GFN on synthetic data, we observed an acceptance rate of between 0.4 and 0.6, suggesting that this difference in scale may be offset by making the same 'approximation' in both the numerator and the denominator.

4.5 Evaluation II

This is the second of two evaluation sections in this report. For a separate evaluation of the CVM, ECVM, DFM and EDFM objectives, see Section 3.4. This section is dedicated to the analysis of the performance of the latter objectives in the context of the EBM training paradigms proposed in this chapter. We assume familiarity with the datasets and the NLL and MMD metrics described in Subsection 3.4.4. We further built on Section 3.4 in that unless stated otherwise, we use the optimal source-path combination determined for CVM and DFM on the respective dataset.

4.5.1 EBM-DFS

We first consider the naïve EBM-DFS algorithm described in Algorithm 1. We have seen in Section 3.4 that the proposal $q_1(\boldsymbol{x}_1) = \prod_{i=0}^{D} Bern(x^i; \bar{x}^i)$ is sufficient for learning to sample from a pre-trained EBM in the case of the synthetic data, but it is not aligned well enough with the target distribution to learn an EBM pre-trained on the discrete image data (3.4). Thus, we cannot expect our ECVM (EDFM) trained sampler to provide meaningful negative samples, especially towards the final stages of the training on discrete image data. We therefore restrict our analysis to the synthetic data. The hypothesis to test here is that EBM-DFS works *given* a suitable proposal distribution. We parameterise the EBM as a four layer MLP with hidden states of dimensionality 256 and the Swish activation function [41], following the setup in [36]. We train the model for 2×10^5 iterations, where for each EBM iteration we perform 10 update steps on the DFS to account for the loss in efficiency due to importance sampling. This ought to ensure that the DFS 'keeps up' with the EBM at all times. We consider the value of this in a separate ablation. We further find that our model performs best with ℓ^2 regularisation of the energy values weighted at 0.1 added to the CD loss. As a baseline we consider EB-GFN which to our knowledge delivers state-of-the-art performance on EBM training with the synthetic data [8]. Note that GFlowNet is also a learned sampler, rendering it a more suitable baseline than MCMC based CD training. We consider the NLL of a test set under the models, as well as H-MMD and E-MMD between the EBM and the data, where we draw samples from the EBM with Gibbs sampling and estimate normalising constants as described in Subsection 3.4.4.

We find that our EBM-DFS approach learns a valid energy function, as can be seen from the heat maps shown in Figure 4.1. Still, we find that with one exception, EB-GFN outperforms our model in terms of NLL and both MMD metrics Table 4.1. On the checkerboard dataset, the EBM-DFS trained with the EDFM objective can outperform EB-GFN in terms of E-MMD. We conclude that while our hypothesis is validated, the naïve approach to training the EBM with a fixed proposal distribution does not deliver state-of-the-art performance, beyond its inherent limitation in higher dimensions.

Noisy Samplers It is worth noting that the EBM learned despite the DFS generating samples that are more noisy than samples drawn from the the EBM with Gibbs sampling, as shown in Figure 4.1. We hypothesise that more noise generally reduces accuracy of the learned EBM, but does not undermine the idea of contrastive divergence — pushing down the energy of negative samples — since the noisy samples are unlikely to fall in high-probability regions of the target distribution, which typically lies on a specific lower dimensional manifold. We observed several cases — for instance using an empirical proposal distribution $q(\boldsymbol{x}_1) = \sum_i^N \delta_{\boldsymbol{x}_{\text{data},i}}(\boldsymbol{x}_1)$ — where the DFS 'gets ahead' of the EBM. This then causes negative samples to lie *between* the current EBM distribution and the target, so that contrastive divergence leads the EBM to diverge away from the DFS.

4.5.2 Ablations

We consider the impact of reducing the number of iterations that we update the DFS for each update of the EBM from ten to one. We compare the discrepancy between the DFS and the EBM during training by measuring the Euclidean MMD of the DFS samples against samples drawn from the EBM via Gibbs sampling. Recall that this is relevant for the validity of our core assumption for the algorithm ($p_\theta \approx p_\phi$). We also monitor the the Euclidean MMD of the EBM against data. To test whether the effect of increased number of iterations can

Table 4.1: NLL, H-MMD and E-MMD for the EBM-DFS trained with EDVM, EDFM and the baseline model EB-GFN; EBM-DFS was trained for 2×10^5 iterations, EB-GFN for 10^5; Figures for NLL, H-MMD of EB-GFN taken from [8]; MMD in units of 1×10^{-4}.

Metric	Method	2spirals	checkerboard	circles	moons	pinwheel	swissroll	8gaussians
NLL↓	ECVM	21.112	21.284	20.966	20.580	20.566	20.641	20.617
	EDFM	20.715	21.375	20.920	20.283	20.420	20.505	20.398
	EB-GFN	**20.050**	**20.696**	**20.546**	**19.732**	**19.554**	**20.146**	**19.982**
H-MMD↓	ECVM	1.147	1.673	3.252	2.800	7.353	3.764	2.805
	EDFM	1.389	1.757	2.420	2.981	10.102	3.237	2.156
	EB-GFN	**0.583**	**1.206**	**0.305**	**0.121**	**0.492**	**0.274**	**0.531**
E-MMD↓	ECVM	1.038	0.229	0.770	1.889	5.997	2.213	2.306
	EDFM	1.117	**0.157**	7.318	1.878	8.561	2.545	2.139
	EB-GFN	**0.829**	0.241	**-0.147**	**0.299**	**1.124**	**0.808**	**0.307**

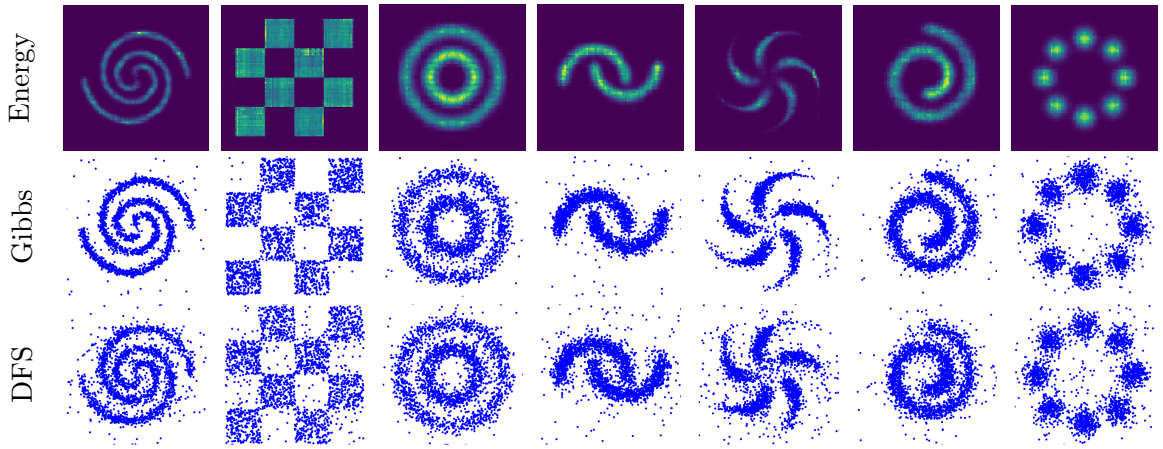


Figure 4.1: Energy heat map, Gibbs samples and final DFS samples for EBM-DFS trained with ECVM objective.

equally be achieved by increasing the learning rate, we further provide data from another experiment where the DFS was updated once per iteration with an increased learning rate of 0.001. The results are graphed in Figure 4.2. We find that indeed the additional iterations are crucial for the DFS to remain close to the EBM during training. For the DFS trained for only one iteration per iteration of the EBM, we find that it diverges strongly from iteration 4×10^4 onwards. While the spread between DFS and EBM also increases in the case of ten DFS iterations per EBM iteration, it remains much closer to the target throughout training. Nevertheless, we note that as training progresses, the spread plateaus and slightly shrinks. This is likely due to the EBM converging, enabling the DFS to catch up 'across' EBM iterations. Interestingly, we find that the impact on the EBM training only shows up significantly later, around iteration 6×10^4, after which the EBM trained with the single step DFS exhibits strong volatility across sampling steps and performs consistently worse in terms of E-MMD. We attribute this mild effect on EBM training to the relatively low dimensionality of the problem, allowing for some noise in the negative samples, as discussed in Section 4.5.1. The experiment with increased learning rate achieves similar results as the one with the orgininal learning rate, confirming that the lower spread can only be achieved by increased iterations. Note that this is intuitive, due to importance sampling requiring a larger

sample size for accurate estimation. Overall, we find that the additional iterations introduce stability into the EBM training process by reinforcing the core lockstep assumption.

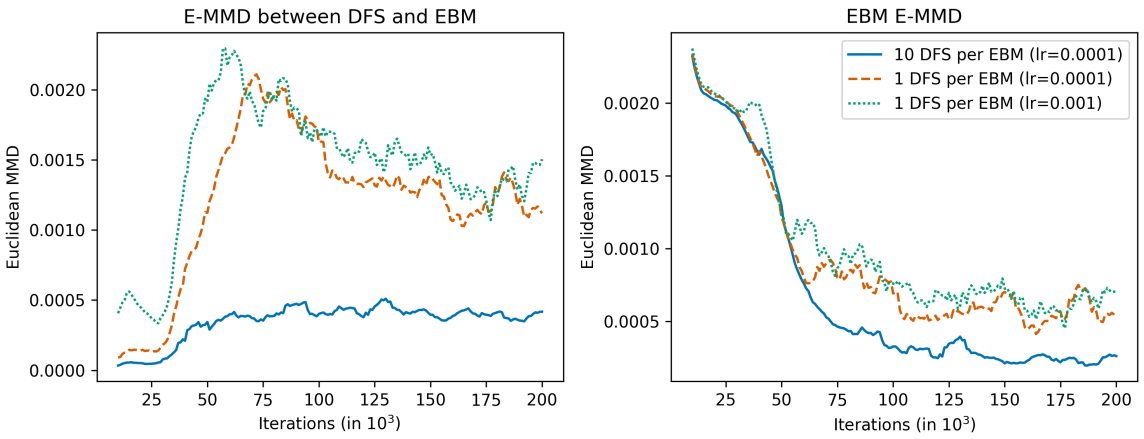


Figure 4.2: Left: Euclidean MMD between DFS and EBM during training; right: Euclidean MMD between EBM and data during training.

4.5.3 EBM-DFS with Bootstrapped Proposal

We now turn to the evaluation of our second proposed algorithm. The hypothesis here is essentially that we can exploit the alignment between DFS and EBM (i.e. $p_\theta \approx p_\phi$) during training in a proposal that samples by generation from the DFS and performs likelihood evaluation using the EBM. As we have stated already in Section 4.3, it turns out that the DFS collapses if it is solely trained on its own samples, which is why we resort to MCMC refinement with DMALA to diversify the samples while maintaining (or even increasing) said alignment. In the expectation that this algorithm may perform better on high-dimensional data due to its non-static proposal, we consider both the synthetic data and the discrete image data this time. For the refinement sampler, we use the DMALA code from the original paper [12], and unless otherwise stated initialise the sampler with a step size of 1 for the synthetic data and 0.2 for the discrete image data and increase (decrease) the step size by a factor of 1.1 (0.9) every 50 iterations if the acceptance rate rises above 0.6 (falls below 0.4).

4.5.4 Synthetic Data

In Figure 4.3, we show the learned energy landscape by training on the synthetic data with Algorithm 3, together with samples from a Gibbs sampler and the final DFS that was used during training. These are generated with $M_{\text{DFS}} = 1$ (i.e. one DFS step per EBM step) and $M_{\text{DMALA}} = 10$ (i.e. ten DMALA steps after DFS generation). We use ECVM*, and train for 2×10^5 iterations. We first emphasise that opposed to the EBM-DFS with static proposal, it appears to be sufficient to train the DFS for only one iteration for each EBM update. This is intuitive, as using the current model state as a proposal reduces the variation in importance weights drastically, as can be seen in Figure 4.4. Recall that this was the primary idea behind this algorithm in the first place. From the energy landscapes, it is evident that despite MCMC refinement (or potentially as a result thereof) the energy landscapes appear slightly less even as was the case with the static proposal in EBM-DFS.

This is visible especially in the 2spirals and checkerboard energy landscapes and their Gibbs samples. We explain this with the fact that if an area is under-sampled by the DFS, then it will be assigned higher energy in training with CD. While the MCMC refinement should in theory counter such trends by moving towards such high energy, there is no guarantee that proposals are placed in those area especially when the energy landscapes are complicated with disconnected modes, as is the case for 2Spirals or checkerboard, and when their are only few refinement steps. Besides these slight signs of mode collapse, we find that the algorithm is able to train both the energy function and the DFS reasonably well. This is confirmed by the evaluation results in Table 4.2. We observe comparable performance to EBM-DFS in most cases, and surprisingly find that despite seemingly uneven energy landscapes we outperform our EBM-DFS on 2spirals and checkerboard. It is reasonable to suspect that much of the success of this method is due to the MCMC refinement steps. This shall be considered in detail in the following section.

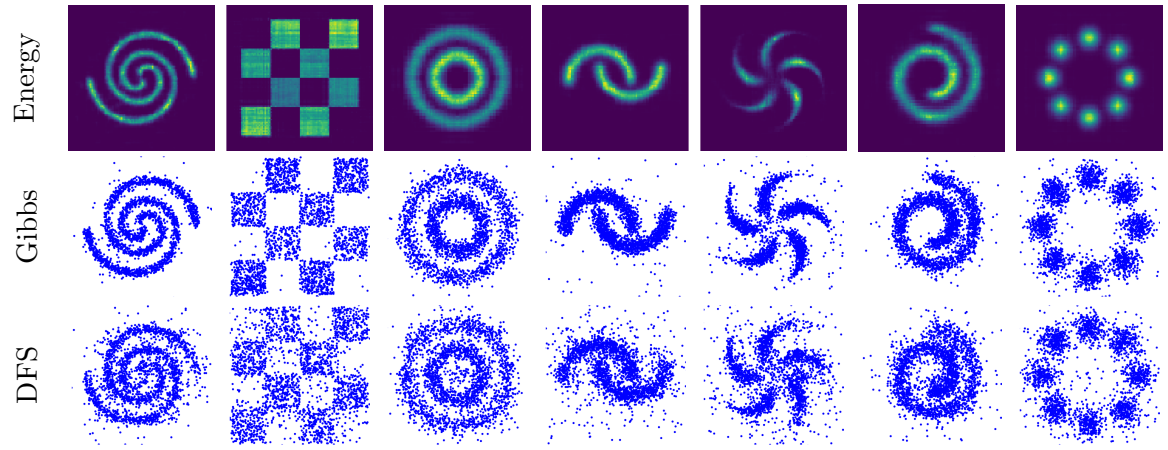


Figure 4.3: Energy heat map, Gibbs samples and final DFS samples for EBM-DFS with bootstrapped proposal trained with ECVM* objective.

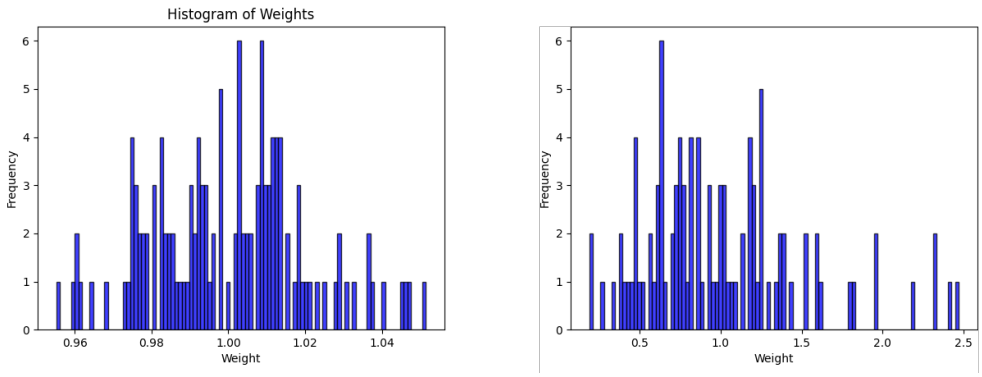


Figure 4.4: Normalised importance weights during training of EBM-DFS with bootstrapped proposal; left: 2spirals at iteration 10^5 (ECVM), right: static MNIST at iteration 4.5×10^4 (EDFM)

Table 4.2: NLL, H-MMD and E-MMD for the EBM-DFS **with bootstrapped proposal** trained with EDVM, EDFM and the baseline model EB-GFN; We add the best EBM-DFS model based on Table 4.1 as EBM-DFS*; Our models were trained for 2×10^5 iterations, EB-GFN for 10^5; Figures for NLL, H-MMD of EB-GFN taken from [8]; MMD in units of 1×10^{-4}.

Metric	Method	2spirals	checkerboard	circles	moons	pinwheel	swissroll	8gaussians
NLL↓	ECVM	20.450	20.938	20.731	20.557	20.404	20.666	20.382
	EDFM	20.431	20.914	20.861	20.473	20.329	20.623	20.183
	EBM-DFS*	20.715	21.375	20.920	20.283	20.420	20.505	20.398
	EB-GFN	**20.050**	**20.696**	**20.546**	**19.732**	**19.554**	**20.146**	**19.982**
H-MMD↓	ECVM	1.691	5.380	1.772	2.447	7.297	2.998	1.953
	EDFM	12.122	**0.483**	1.131	2.538	6.315	1.421	2.309
	EBM-DFS*	1.147	1.673	2.420	2.800	7.352	3.237	2.156
	EB-GFN	**0.583**	1.206	**0.305**	**0.121**	**0.492**	**0.274**	**0.531**
E-MMD↓	ECVM	1.758	0.862	0.893	2.288	5.964	2.395	2.280
	EDFM	5.389	**0.151**	0.994	2.694	5.236	2.145	3.280
	EBM-DFS*	1.038	0.157	0.770	1.878	5.997	2.213	2.139
	EB-GFN	**0.829**	0.241	**-0.147**	**0.299**	**1.124**	**0.808**	**0.307**

4.5.5 Ablations

We now consider to what extend the success of our algorithm depends on the number of MCMC steps performed, and how performance changes if we a) do not reuse the refined training samples and instead use a new sample from the DFS as negative sample or b) split the ten refinement steps equally between refinement of the DFS training sample, and refinement of a new DFS sample to be used as negative sample. Recall that the reason for introducing MCMC (with DMALA) was the apparent collapse of DFS when trained on its own samples directly. We can therefore assume that *to some extend* our algorithm will always be dependent on MCMC. As the case $M_{\text{DMALA}} = 0$ is therefore uninteresting, we consider instead $M_{\text{DMALA}} \in \{2, 5\}$ beyond our original choice of $M_{\text{DMALA}} = 10$. As in Subsection 4.5.5, we monitor the Euclidean MMD between DFS and EBM as well as between EBM and data throughout the training process. The results are shown in Figure 4.5. First, we can confirm the stabilising effect that increased closeness between the sampler and the EBM has on EBM training which already emerged in Subsection 4.5.2. It is clear that all alternatives perform worse than our parameter choice in terms of both closeness of p_θ and p_ϕ, as well as closeness of p_ϕ to the data. With regards to the number of refinement steps, this is a sign that the performance of our method rests heavily on the adjustment of samples with MCMC. With regards to the choice of reusing the refined samples from DFS training on the other hand, the results suggest that learning in the DFS is not harmed by training on the same samples for which the energy has just been pushed up (compare Algorithm 3). Nevertheless, from the discrepancy in performance between our main model and the model where negative samples are generated from the DFS without refinement, it is evident that not only the DFS training, but also the EBM training relies on the refinement. We must therefore conclude that the competitive results that our model achieves compared to EB-GFN are largely attributable to the use of MCMC. One anomaly to note here is that the refinement with two steps actually performs better than the refinement with five steps in terms of E-MMD for large parts of the training. This is due to the low dimensionality of the problem at hand, where counterintuitively we find that to some degree the EBM can be

trained on random negative samples – which in this case are caused by a DFS that around iteration 5×10^4 collapses to random noise. Of course, this cannot be the case in higher dimensional problems (as we shall see), so we ignore this anomaly here. Finally, we point out that for $M_{\text{DMALA}} = 5$ we observed increased instability over the course of training where mode collapse appears and disappears periodically (see Figure C.2).

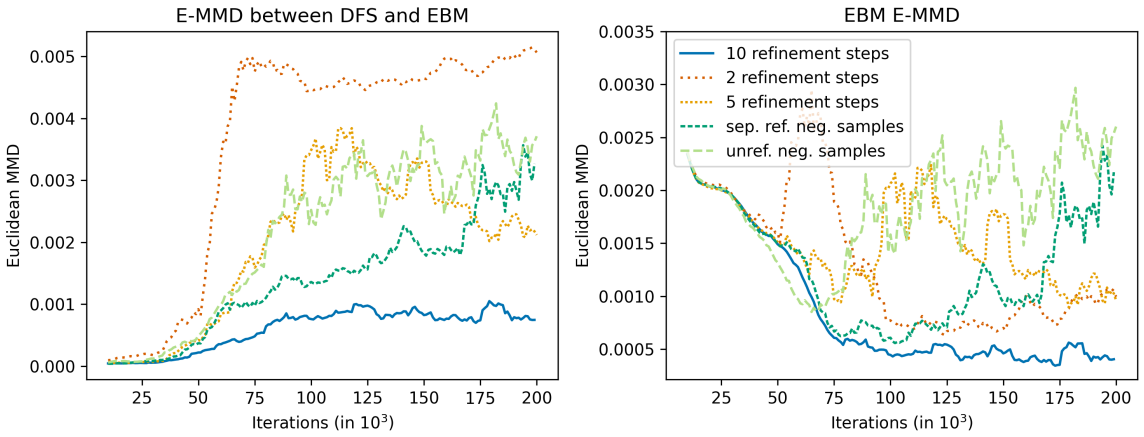


Figure 4.5: Different ablations on EBM-DFS with bootstrapped proposal on 2spirals; Left: Euclidean MMD between the DFS and the EBM during training; Right: Euclidean MMD between the EBM and the data during training.

4.5.6 Discrete Image Data

We now turn to the evaluation of our algorithm on discrete image data. Recall that we did not apply the original EBM-DFS on this data, since the results from training on a pre-trained energy were not showing sufficient learning in the DFS due to the static proposal distribution (compare Section 3.4.6). We follow [8] in specifying the EBM as a simple MLP with three hidden layers and 256 hidden dimensions, instead of a larger ResNet architecture, as done for instance in [12], since this significantly speeds up training. However, we also note that in the initial explorations done with the ResNet architecture, both our algorithms, and surprisingly the EB-GFN original code as well, lacked stability in training. Our main model is specified with $M_{\text{DMALA}} = 40$, which is the result of learning in the synthetic data case that the algorithm does rely on the refinement. We will consider lower values of M_{DMALA} in the ablations. Furthermore, we restrict this evaluation to the energy-based DFM objective, since the evaluation in Section 3.4.5 suggested the superiority of the DFM objective for learning with discrete image data. We further find that some level of ℓ^2 regularisation is beneficial, as was the case for the synthetic data, and we set the weight on the ℓ^2 loss to 1×10^{-4}. We run the experiments for 5×10^4 iterations, with NLL evaluation on a validation set at every 5000th iteration, to ensure comparability primarily with our main baseline EB-GFN [8], but also with [12]. The baseline from [12] amounts to PCD trained with 40 steps of DMALA at each iteration, using a buffer for persistent state – this therefore delivers useful comparison between buffer and DFM. We report the best performance of an exponential moving average EBM with decay parameter 0.999 on the test set. Generations are shown in Figure 4.6, and quantitative results are shown in Table 4.3. The EBM generations are obtained with 10^3 steps of DMALA. Beyond the DFS samples, we show the refined samples after M_{DMALA} steps. On first glance, it is clear that our DFS based method cannot compete with the DMALA PCD method, while it comes closer to the performance of EB-GFN. We treat the exceptionally

high NLL on static MNIST as an outlier, which will be put into perspective further in the ablations — note that the generations differ only barely from those for dynamic MNIST where the model appears relatively competitive. It is worth noting that PCD has lower complexity than our method, since the number of DMALA steps performed per iteration is the same, but no separate computation is required for training the DFS — of course the buffer requires more space. We emphasise that the bulk of processing time per iteration is due to DMALA, and therefore our algorithm cannot deliver on the hope of higher scalability than existing methods. Nevertheless, the algorithm trained the DFS to produce samples reasonably close to the target, which barely change under the final refinement. In the case of the MNIST data and the Caltech Silhouettes data however, it appears that the EBM is experiencing a mode collapse not seen in the DFS, suggesting that the generative distribution of the DFS is not tracing the EBM sufficiently close. Note though that the MCMC samples (top) vary with parameter settings, and are generally prone to mode collapse potentially explaining some of the discrepancies. Finally, we confirm that where the interleaved training works, importance weights shown in Figure 4.4 have low variation, in a stark contrast to the static proposal case in Figure 3.9.

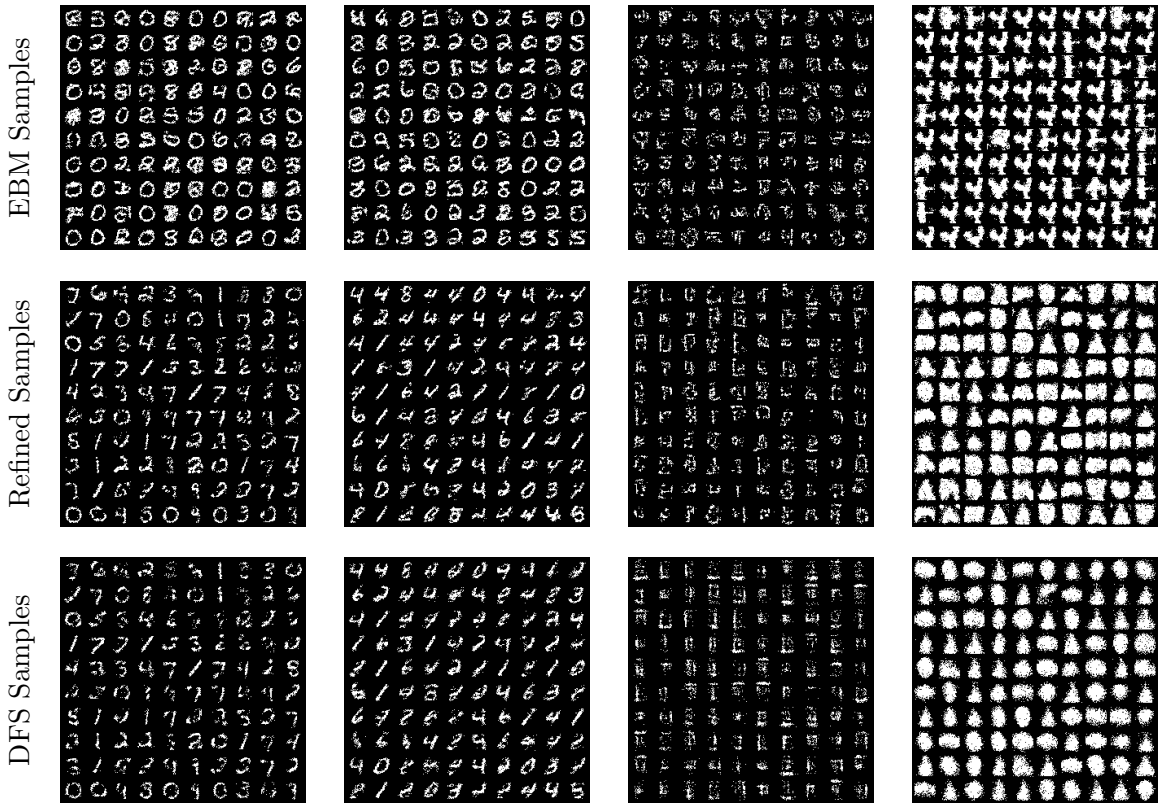


Figure 4.6: EDFM-trained EBM-DFS with bootstrapped proposal (uniform source, linear path, with EDFM); left to right: static MNIST, dynamic MNIST, Omniglot, Caltech Silhouettes; top to bottom: EBM samples obtained with 10×10^3 DMALA steps, samples refined from DFS samples with $M_{\text{DMALA}} = 40$ steps, DFS samples.

4.5.7 Ablations

We briefly compare the negative likelihood of the trained EBM for $M_{\text{DMALA}} \in \{10, 20, 30\}$ to test the reliance of our method on the MCMC refinement steps. The results are shown

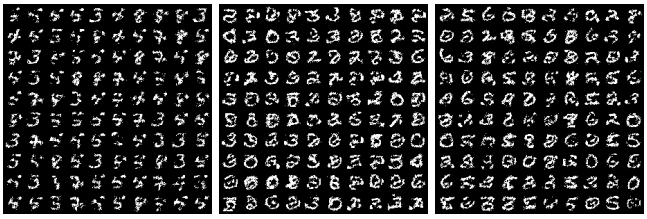
Table 4.3: NLL for GflowNet and our EDFM* trained bootstrapped EBM-DFS on discrete image datasets (with EDFM). Data for GFlowNet taken from [8] and data for DMALA taken from [12].

Method	**stat. MNIST**	**dyn. MNIST**	**Omniglot**	**Caltech**
EBM-DFS-B	447.327	146.14	154.25	232.48
GFN	102.43	105.75	112.59	185.57
DMALA	**79.46**	**79.54**	**91.11**	**87.82**

in table Table 4.4. Unfortunately, the results are little conclusive, but we can safely say that it is feasible to reduce the number of M_{DFS} steps significantly below 40 without a large increase in NLL or an obvious loss in generation quality. This is evidence in favour of our method, since it means that it is dependent on the number of refinement steps *only to a limited extend*. Recall however, that $M_{\text{DFS}} = 0$ is infeasible and leads to model collapse in the DFS.

Table 4.4: EBM-DFS with bootstrapped proposal ablations on static MNIST (with EDFM); left to right: NLL on discrete image datasets for different values of M_{DMALA}, samples from EBM-DFS with bootstrapped proposal trained with $M_{\text{DMALA}} = 10$, $M_{\text{DMALA}} = 20$, $M_{\text{DMALA}} = 30$ respectively obtained with 10×10^3 iterations of DMALA.

M_{DMALA}	**NLL**
10	156.465
20	311.767
30	151.980
40	447.327



4.6 EBM-DFS with Heuristic MH Acceptance

As alluded in Section 4.4, we will only briefly describe the behaviour of our proposed EBM-DFS algorithm with heuristic MH acceptance step as tuning this model proved difficult and only limited insights are available at the time of writing. To be precise, we have only been able to observe limited learning in the EBM trained on synthetic data, and all experiments on discrete image data have so far resulted in a divergence between EBM and DFS. We hypothesise that with sufficient parameter tuning, further positive results can be achieved. We outline some ideas to this end in Chapter 5. Recall that the hypothesis behind this algorithm is that instead of learning the energy directly, we can obtain negative samples from the EBM (to train the EBM with CD, and the DFS with CVM or DFM) through a heuristic MH in which perturbations are proposed by sampling back-and-forth through the DFS, starting from data.

4.6.1 Synthetic Data

On the synthetic data, we found out of those that were tried, the following parameter settings enabled the EBM to learn. The EBM and DFS are specified in the same way as in previous evaluations on the synthetic data. We fix $t_{\text{turn}} = 0.5$ throughout training, and set $M_{\text{DFS}} = 1$.

We use the DFM loss from [6] to train the DFS. As before, we perform 2×10^5 iterations. We monitor the DFS loss and acceptance rate of the MH step, together with DFS generations and the EBM heat during training (see Figure 4.7). Due to time and resource constraints, results are at present only available for the 2spirals dataset. Interestingly, the EBM first collapses and assigns low energy to points away from the DFS and the data. However, from iteration 6×10^4 onwards, the sprials shape starts to emerge and appears relatively clear by iteration 10×10^4. Nevertheless, some probability mass remains in the low probability areas away from the data, due to the proposal rarely returning negative samples there. This is cause by the DFS samples relatively soon shaping up in the center of domain when viewed in Euclidean space, away from the edges. As the EBM starts 'shaping up', we see the acceptance rate which is initially around 90%, drop and plateau between 70% and 80%. Note that unlike in classical MCMC, a high acceptance rate is desirable, since if $\boldsymbol{x}'_1 = \boldsymbol{x}_1$, the CD loss from this pairing of negative sample and data sample will be zero. The final EBM achieved a test **NLL of 20.813**, falling short of the 20.050 benchmark for EB-GFN after 10^5 iterations.

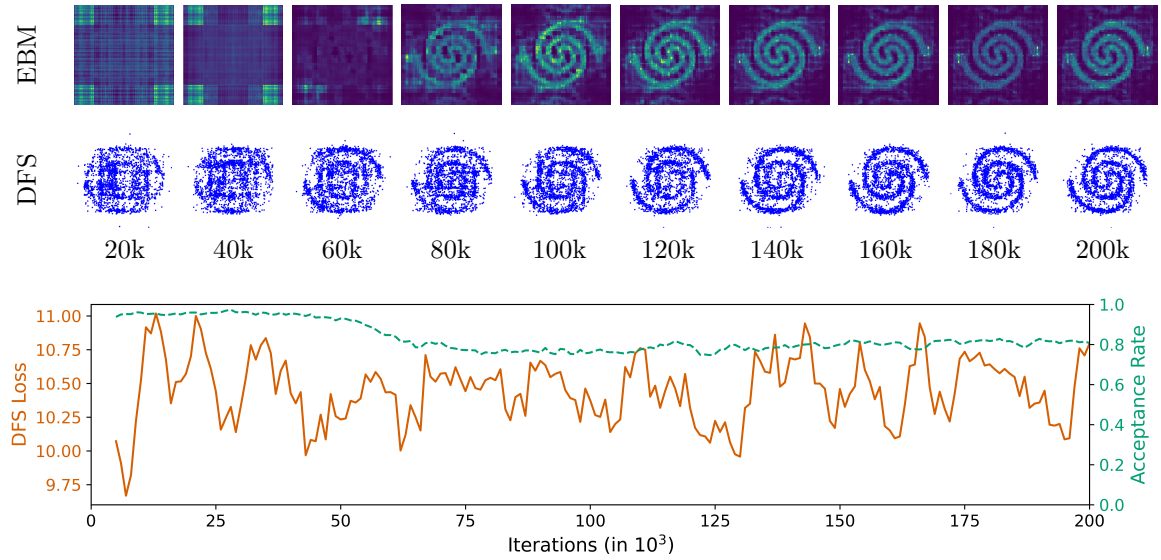


Figure 4.7: Training of EBM-DFS with heuristic MH step on 2spirals; top to bottom: EBM heatmap, DFS samples, plot of acceptance rate of MH step and DFS loss.

4.6.2 Discrete Image Data

For the lack of a model that performed well on the discrete image task, we use the case of static MNIST to illustrate how the MH step operates. We consider the same parameter setting as for the synthetic case discussed above, except that we adapt the flow and energy models to this setting, and we use our optimal source-path combination (i.e. uniform-linear) established in Section 3.4. Figure 4.8 shows instances of batches of size 100 for $\boldsymbol{x}_1$ (sampled from data) $\boldsymbol{x}'_{0.5}$ (obtained after backwards sampling from $\boldsymbol{x}_1$), $\boldsymbol{x}'_1$ (obtained after forward sampling from $\boldsymbol{x}'_{0.5}$) and $\boldsymbol{x}'_{1,\text{MH}}$ (obtained after applying the MH acceptance rule to $\boldsymbol{x}'_1$). We also show DFS and EBM generations. All shown instances are for iteration 5000. As intended, we find that $\boldsymbol{x}'_{0.5}$ retains some of the structural information of $\boldsymbol{x}_1$, and $\boldsymbol{x}'_1$ does appear to be a noisy reconstruction of the latter. $\boldsymbol{x}'_{1,\text{MH}}$ is then a mixture of both, where the share of samples carried over from $\boldsymbol{x}'_1$ is determined by the acceptance rate. We believe that one main symptom of why the EBM still collapses despite the MH step working as

expected is a low acceptance rate – this initially lies around 50%, but soon falls to around 5%, as evident from the $\boldsymbol{x}'_1$ shown here. We have not been able to achieve a consistently high acceptance rate above 10%.

Figure 4.8: Understanding the MH step; left to right: $\boldsymbol{x}_1$, $\boldsymbol{x}'_{0.5}$, $\boldsymbol{x}'_1$, $\boldsymbol{x}'_{1,\mathrm{MH}}$, DFS samples, EBM samples obtained with 10×10^3 steps of DMALA.

4.7 Summary

In this chapters we considered three approaches to deploy discrete flows in the training of EBMs with contrastive divergence. We find that each approach yields promising results on synthetic data benchmarks. Nevertheless, only by refining DFS samples with MCMC steps to get closer to the EBM's distribution have we been able to demonstrate effective training on higher-dimensional discrete image datasets. We believe that while discrete flows are powerful generative models when trained on data, further work is necessary to achieve similar performance when used as learned samplers of an energy function. Alternative objectives that go beyond the importance sampling approach proposed here should be explored to this end.

Chapter 5

Conclusion

We close this report by considering legal, social, ethical as well as professional considerations relevant to the work conducted (5.1), recapitulating the main achievements of the project (5.2), and discussing new avenues to be explored in future research (5.3).

5.1 Legal, Social, Ethical and Professional Considerations

As described in the introduction, EBMs are extremely versatile generative models that can be applied across discrete and continuous domains. Moreover, discrete flows are presently developing into a leading method for learning discrete data distributions in various scenarios from language modelling to sequence prediction in proteins [7][6]. We emphasise that by advancing the theory and empirical understanding for training these methods, we intend to *exclusively* foster their use for civilian purposes. Yet, we cannot ignore the possibility of these general purpose techniques being used in illegal, military or otherwise harmful downstream applications. It is worth remembering that MCMC methods were originally developed at the Los Alamos National Laboratory during World War II, where they were deployed by von Neumann on nuclear fusion and fission problems during the development of the first atomic bomb [43]. They have since become a prevalent method used in many civilian applications such as drug discovery [44]. To manage the risk inherent to such 'dual use' technologies, we support efforts to develop and apply mitigating frameworks. Relevant proposals that have been broad forward include for instance the adaptation of the Dual Use Research of Concern framework from the life science context to generative modelling [45].

The synthetic data used in this project is created with the open source `scikit-learn` library, which is licensed under a BSD3 License. The Omniglot Silhouettes dataset is licensed under a Creative Commons Attribution 4.0 License. The original MNIST dataset is available openly at Yann LeCun's website, and was derived from NIST's Special Database 3 and Special Database 1.[1] The original Caltech Silhouettes dataset is available openly at Benjamin M. Marlin's website.[2] We note that the code for [6] is not publicly available and therefore all DFS related code is entirely our own. All other code, particularly for data loading, evaluation and the training of our pre-trained EBMs was derived from [12], [36] and [8]. Some initial code adapted from [7] was provided at the start of this project by Zijing Ou, however most of this was abandoned when switching our core methodology from Discrete Flow Models [7] to Discrete Flow Matching [6]. To the best of our knowledge there is no copyright or licensing

[1] https://yann.lecun.com/exdb/mnist/

[2] https://people.cs.umass.edu/ marlin/data.shtml

infringement. Lastly, we state that the project report by former MSc student Cornelius Braun has been a helpful resource for structuring this report.[3]

5.2 Achievements

We now provide a brief overview of the achievements of the project.

1. **We have successfully adapt the regression style objective from continuous conditional flow matching to discrete flows.** This expands the toolkit available for learning discrete flows from data. We have shown that our new conditional velocity matching performs competitively on synthetic discrete data, and successfully learns in higher dimensional discrete image data settings.

2. **We successfully transform between complex data distributions.** In a departure from [6] and [7], we achieve this by substituting a generic source distribution with a second data distribution. Surprisingly, in our experiment this did not reduce generation quality.

3. **We make a first leap towards adapting discrete flows to data-free scenarios with our energy based objectives.** To our knowledge, discrete flow matching has hitherto been limited to data-based training. By applying concepts from importance sampling, we show that we can successfully learn flows to sample from a given distribution defined by an energy function, given the availability of a good proposal.

4. **We propose EBM-DFS which enables the use of DFSs in the training of EBMs.** We have demonstrated that as long as a suitable proposal distribution is available, we can train EBMs with a DFS without the need for MCMC or the use of a buffer.

5. **We propose and evaluate EBM-DFS with bootstrap proposal.** This novel idea merges the generative capabilities of DFS with unnormalised likelihood evaluation of the EBM providing a 'bootstrapped' dynamic proposal during interleaved training. We provide a neat theoretical derivation, but find that in practice this training method still requires MCMC sampling to reinforce model alignment between the energy model.

6. **We provide evidence that heuristic MH steps can work in discrete flows.** Despite arguing that the MH step in [8] is only a heuristic approximation of true MH acceptance steps, we demonstrate that beyond GFlowNets, this notion yields empirically solid results with two-way discrete flows. We show indicative results suggesting that this can be used in the interleaved training of EBMs.

7. **We show empirically the validity of two alternative evaluation methods for discrete flows.** We demonstrate how using Euclidean MMD in evaluating discrete generative models and samplers on the widely used discretised synthetic datasets from `scikit-learn` can improve the convergence monitoring of models by aligning more closely with the perceived closeness of distributions projected into Euclidean space. Moreover, we find that evaluating the NLL of generated samples under a ground truth pre-trained models proves an effective and fast way for monitoring sample quality in higher dimensional problems such as discrete MNIST.

[3] https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/2122-pg-projects/Black-Box-Constrained-Bayesian-Optimisation-With-Tree-Ensembles.pdf

5.3 Future Work

It appears that the project has opened at least as many questions as it answered, which is why believe there is ample scope for further work in this area. Here, we outline a selection of research directions that we expect to improve our understanding of discrete flows in different areas:

- **Expanded evaluation of EBM-DFS with Heursitic MH Step.** Due to time and resource constraints, we were only able to show basic results indicating the potential of this model. Further experiments should include the use of multiple consecutive back-and-forth steps for better refinement, akin to conventional MCMC. We further believe that an adaptive algorithm for the 'turning time' parameter could help control the acceptance rate within a desired range.
- **Persistent state.** Given that PCD [12] approaches still dominate both our approach, as well as EB-GFN [8], it may be worth exploring empirically if we can exploit the DFS's ability to model data-to-data transformations to update samples in a buffer. The discrete flow in that case could be seen as a fixed length Markov chain.
- **Exploration of applicability of RL type objectives.** Inspired by the effectiveness of GFlowNets [9][8], we believe that the importance sampling based approach for data-free learning could be improved on by treating the negative energy as a reward, of sampling through the model. Note however that this would likely compromise the superior training efficiency that discrete flows enjoy under data-based paradigms.
- **Adapting Liouville Flow Samplers to the discrete case.** In a recent publication [46] propose an importance sampling based technique for learning velocity fields in the continuous case, by learning the solution to an equation known as the Generalised Liouville Equation due to [47]. While the relevant theory is presently beyond our knowledge, we suspect that useful parallels to the discrete case may exist, as is the case with CFM [13] and DFM [6].
- **Speculative decoding.** This idea was proposed by Zijing Ou. Based on the surprisingly positive result in our data-to-data experiment, we are interested in exploring the feasibility of deploying this technique in speculative decoding. Speculative decoding is used to speed up inference in large auto-regressive models such as transformer based large language models [48]. We envision the use of data-trained discrete flows as decoder models between small, efficient and a more complex and capable, slower models.

5.4 Final Summary

This project was motivated by the idea of applying recent flow-driven advances in generative modelling in the discrete domain to the long-standing challenge of sampling efficiently from high-dimensional EBMs. After proposing and demonstrating the effectiveness of an alternative objective – termed conditional velocity matching – for the data-based case, we resorted to importance sampling to provide energy-based counterparts for both our new objective and the existing discrete flow matching objective. A major challenge that we faced in applying these methods in EBM training lies in determining suitable proposal distributions that are close to the (moving) target. We were able to present a series of innovative workarounds

grounded in both theory and heuristics. Yet, the only algorithm that we found to deliver reasonable results beyond a series of synthetic problems relies heavily on reinforcing the match between the discrete flow and the EBM with MCMC. We hope that the reader is left with a better understanding of discrete flows, EBMs, and why MCMC methods are still incredibly powerful in the age of learned models.

Bibliography

[1] Boltzmann L. In: Hasenöhrl F, editor. Studien über das Gleichgewicht der lebendigen Kraft zwischen bewegten materiellen Punkten. Cambridge Library Collection - Physical Sciences. Cambridge University Press; 2012. p. 49-96.

[2] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al.. Attention Is All You Need; 2023. Available from: https://arxiv.org/abs/1706.03762.

[3] Ho J, Jain A, Abbeel P. Denoising Diffusion Probabilistic Models; 2020. Available from: https://arxiv.org/abs/2006.11239.

[4] Murphy KP. Probabilistic Machine Learning: Advanced Topics. MIT Press; 2023. Available from: http://probml.github.io/book2.

[5] Hinton G. ARTICLE Training Products of Experts by Minimizing Contrastive Divergence. Neural computation. 2002 09;14:1771-800.

[6] Gat I, Remez T, Shaul N, Kreuk F, Chen RTQ, Synnaeve G, et al.. Discrete Flow Matching; 2024. Available from: https://arxiv.org/abs/2407.15595.

[7] Campbell A, Yim J, Barzilay R, Rainforth T, Jaakkola T. Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design; 2024.

[8] Zhang D, Malkin N, Liu Z, Volokhova A, Courville A, Bengio Y. Generative Flow Networks for Discrete Probabilistic Modeling; 2022.

[9] Malkin N, Jain M, Bengio E, Sun C, Bengio Y. Trajectory balance: Improved credit assignment in GFlowNets; 2023.

[10] Grenioux L, Éric Moulines, Gabrié M. Balanced Training of Energy-Based Models with Adaptive Flow Sampling; 2024.

[11] Grathwohl W, Swersky K, Hashemi M, Duvenaud D, Maddison CJ. Oops I Took A Gradient: Scalable Sampling for Discrete Distributions; 2021.

[12] Zhang R, Liu X, Liu Q. A Langevin-like Sampler for Discrete Distributions; 2022. Available from: https://arxiv.org/abs/2206.09914.

[13] Lipman Y, Chen RTQ, Ben-Hamu H, Nickel M, Le M. Flow Matching for Generative Modeling; 2023.

[14] Tong A, Fatras K, Malkin N, Huguet G, Zhang Y, Rector-Brooks J, et al.. Improving and generalizing flow-based generative models with minibatch optimal transport; 2024.

[15] Robert C, Casella G. Monte Carlo Statistical Method. Technometrics. 2000 11;42.

[16] Neal RM. Annealed Importance Sampling; 1998.

[17] Roberts GO, Tweedie RL. Exponential convergence of Langevin distributions and their discrete approximations. Bernoulli. 1996;2(4):341 363.

[18] Parisi G. Correlation functions and computer simulations. Nuclear Physics B. 1981;180(3):378-84. Available from: `https://www.sciencedirect.com/science/article/pii/0550321381900560`.

[19] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by Simulated Annealing. Science. 1983;220(4598):671-80. Available from: `https://www.science.org/doi/abs/10.1126/science.220.4598.671`.

[20] Gabrié M, Rotskoff GM, Vanden-Eijnden E. Adaptive Monte Carlo augmented with normalizing flows. Proceedings of the National Academy of Sciences. 2022 Mar;119(10). Available from: `http://dx.doi.org/10.1073/pnas.2109420119`.

[21] Chen RTQ, Rubanova Y, Bettencourt J, Duvenaud D. Neural Ordinary Differential Equations; 2019.

[22] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition; 2015.

[23] Bengio E, Jain M, Korablyov M, Precup D, Bengio Y. Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation; 2021.

[24] Ackley DH, Hinton GE, Sejnowski TJ. A learning algorithm for boltzmann machines. Cognitive Science. 1985;9(1):147-69. Available from: `https://www.sciencedirect.com/science/article/pii/S0364021385800124`.

[25] Salakhutdinov R, Hinton G. Deep Boltzmann Machines. In: van Dyk D, Welling M, editors. Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics. vol. 5 of Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR; 2009. p. 448-55. Available from: `https://proceedings.mlr.press/v5/salakhutdinov09a.html`.

[26] LeCun Y, Chopra S, Hadsell R, Ranzato M, Huang FJ. A tutorial on energy-based learning. In: Predicting Structured Data. MIT Press; 2006. Available from: `https://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf`.

[27] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization; 2017.

[28] Tieleman T. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient; 2008. p. 1064-71.

[29] Tieleman T, Hinton G. Using fast weights to improve persistent contrastive divergence; 2009. p. 130.

[30] Du Y, Mordatch I. Implicit Generation and Generalization in Energy-Based Models; 2020.

[31] Hyvärinen A. Estimation of Non-Normalized Statistical Models by Score Matching. Journal of Machine Learning Research. 2005;6(24):695-709. Available from: `http://jmlr.org/papers/v6/hyvarinen05a.html`.

[32] Vincent P. A Connection Between Score Matching and Denoising Autoencoders. Neural Computation. 2011;23(7):1661-74.

[33] Song Y, Garg S, Shi J, Ermon S. Sliced Score Matching: A Scalable Approach to Density and Score Estimation; 2019.

[34] Meng C, Choi K, Song J, Ermon S. Concrete Score Matching: Generalized Score Matching for Discrete Data; 2023.

[35] Gutmann M, Hyvärinen A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In: Teh YW, Titterington M, editors. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. vol. 9 of Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR; 2010. p. 297-304. Available from: https://proceedings.mlr.press/v9/gutmann10a.html.

[36] Schröder T, Ou Z, Lim JN, Li Y, Vollmer SJ, Duncan AB. Energy Discrepancies: A Score-Independent Loss for Energy-Based Models. 2023.

[37] Schröder T, Ou Z, Li Y, Duncan AB. Training Discrete Energy-Based Models with Energy Discrepancy; 2023.

[38] Dai H, Singh R, Dai B, Sutton C, Schuurmans D. Learning Discrete Energy-based Models via Auxiliary-variable Local Exploration; 2020.

[39] Gray F. Pulse code communication; 1953.

[40] Gretton A, Borgwardt KM, Rasch MJ, Schölkopf B, Smola A. A Kernel Two-Sample Test. Journal of Machine Learning Research. 2012;13(25):723-73. Available from: http://jmlr.org/papers/v13/gretton12a.html.

[41] Ramachandran P, Zoph B, Le QV. Searching for Activation Functions; 2017. Available from: https://arxiv.org/abs/1710.05941.

[42] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition; 2015. Available from: https://arxiv.org/abs/1512.03385.

[43] Robert C, Casella G. A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data. Statistical Science. 2011 Feb;26(1). Available from: http://dx.doi.org/10.1214/10-STS351.

[44] Xie Y, Shi C, Zhou H, Yang Y, Zhang W, Yu Y, et al.. MARS: Markov Molecular Sampling for Multi-objective Drug Discovery; 2021. Available from: https://arxiv.org/abs/2103.10432.

[45] Grinbaum A, Adomaitis L. Dual use concerns of generative AI and large language models. Journal of Responsible Innovation. 2024 Jan;11(1). Available from: http://dx.doi.org/10.1080/23299460.2024.2304381.

[46] Tian Y, Panda N, Lin YT. Liouville Flow Importance Sampler; 2024. Available from: https://arxiv.org/abs/2405.06672.

[47] Gerlich G. Die verallgemeinerte Liouville-Gleichung. Physica. 1973;69(2):458-66. Available from: https://www.sciencedirect.com/science/article/pii/0031891473900839.

[48] Leviathan Y, Kalman M, Matias Y. Fast Inference from Transformers via Speculative Decoding; 2023. Available from: `https://arxiv.org/abs/2211.17192`.

Appendix A

Proofs

A.1 Proof of Proposition 1

We prove that $\nabla_\theta \mathcal{L}_{\text{VM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CVM}}(\theta)$. Using the results in Equation 3.3, we have

$$\mathcal{L}_{\text{VM}}(\theta) = \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x}_t)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t)\|_2^2 \tag{A.1}$$

$$= \sum_{i=1}^{D} \sum_{z^i=0}^{S} \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x}_t)} \Bigg[\|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta)\|_2^2 \tag{A.2}$$

$$-2\langle \hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta), u_t^i(z^i, \boldsymbol{x}_t) \rangle \Bigg] + c \tag{A.3}$$

$$= \sum_{i=1}^{D} \sum_{z^i=0}^{S} \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x}_t)} \Bigg[\|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta)\|_2^2 \tag{A.4}$$

$$-2\langle \hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta), \sum_{x_0, x_1} u_t^i(z^i, \boldsymbol{x}_t | \boldsymbol{x}_0, \boldsymbol{x}_1) p(\boldsymbol{x}_0, \boldsymbol{x}_1 | \boldsymbol{x}_t) \rangle \Bigg] + c \tag{A.5}$$

$$= \sum_{i=1}^{D} \sum_{z^i=0}^{S} \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x}_t)} \Bigg[\|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta)\|_2^2 \tag{A.6}$$

$$- 2 \sum_{x_0, x_1} p(\boldsymbol{x}_0, \boldsymbol{x}_1 | \boldsymbol{x}_t) \langle \hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta), u_t^i(z^i, \boldsymbol{x}_t | \boldsymbol{x}_0, \boldsymbol{x}_1) \rangle \Bigg] + c \tag{A.7}$$

$$= \sum_{i=1}^{D} \sum_{z^i=0}^{S} \mathbb{E}_{\mathcal{U}[t;0,1],p_t(\boldsymbol{x}_t)} \sum_{x_0, x_1} p(\boldsymbol{x}_0, \boldsymbol{x}_1 | \boldsymbol{x}_t) \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t | \boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2 + c' \tag{A.8}$$

$$= \mathbb{E}_{\mathcal{U}[t;0,1],p(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1),\pi(\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t | \boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2 + c' \tag{A.9}$$

$$= \mathcal{L}_{\text{CVM}} + c' \tag{A.10}$$

where c, c' are constants with respect to the parameter θ and the third equality follows from the results in Equation 3.3 and the change in expectation is due to Bayes, i.e. $p(\boldsymbol{x}_0, \boldsymbol{x}_1 | \boldsymbol{x}_t) = \frac{p(\boldsymbol{x}_t | \boldsymbol{x}_0, \boldsymbol{x}_1) \pi(\boldsymbol{x}_0, \boldsymbol{x}_1)}{p_t(\boldsymbol{x}_t)}$.

A.2 Proof of Proposition 2

In this section, we prove that $\nabla_\theta \mathcal{L}_{\text{VM}}(\theta) = c\nabla_\theta \mathcal{L}_{\text{ECVM}}(\theta)$. Let $Z = \sum_{\mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}_1))$ then $p_{\text{ebm}}(\boldsymbol{x}_1) = \exp(-\mathcal{E}(\boldsymbol{x}_1))/Z$. We then have

$$
\begin{aligned}
&\mathcal{L}_{\text{ECVM}}(\theta) \\
&= \mathbb{E}_{\mathcal{U}[t;0,1],\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D}\sum_{z^i=0}^{S} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{q_1(\boldsymbol{x}_1)} \|\hat{u}_t^i(z^i,\boldsymbol{x}_t;\theta) - u_t^i(z^i,\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)\|_2^2 \quad \text{(A.11)} \\
&= Z\mathbb{E}_{\mathcal{U}[t;0,1],\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D}\sum_{z^i=0}^{S} \frac{p_{\text{ebm}}(\boldsymbol{x}_1)}{q_1(\boldsymbol{x}_1)} \|\hat{u}_t^i(z^i,\boldsymbol{x}_t;\theta) - u_t^i(z^i,\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)\|_2^2 \quad \text{(A.12)} \\
&= Z\mathbb{E}_{\mathcal{U}[t;0,1],\pi(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D}\sum_{z^i=0}^{S} \|\hat{u}_t^i(z^i,\boldsymbol{x}_t;\theta) - u_t^i(z^i,\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1)\|_2^2 \quad \text{(A.13)} \\
&= Z\mathcal{L}_{\text{CVM}}(\theta).
\end{aligned}
$$

We used the fact that $\hat{\pi}(\boldsymbol{x}_0) = p_0(\boldsymbol{x}_0)q(\boldsymbol{x}_1)$, and $\pi(\boldsymbol{x}_0) = p_0(\boldsymbol{x}_0)p_{\text{ebm}}(\boldsymbol{x}_1)$. Thus we have $\nabla_\theta \mathcal{L}_{\text{EVRM}}(\theta) = Z\nabla_\theta \mathcal{L}_{\text{CVM}}(\theta) = Z\nabla_\theta \mathcal{L}_{\text{VM}}(\theta)$ (using also Equation 3.12). In other words, if $\theta^* = \text{argmax}_\theta \mathcal{L}_{\text{ECVM}}(\theta)$, one can draw samples from p_{ebm} by simulating the CTMC defined by p_0 and $\hat{u}_t^i(z^i,\boldsymbol{x}_t;\theta)$.

A.3 Proof of Proposition 3

Here we show $\nabla_\theta \mathcal{L}_{\text{EDFM}} = c\nabla_\theta \mathcal{L}_{\text{DFM}}$, where from Equation 2.45 we recall the original DFM objective from [6]

$$
\mathcal{L}_{DFM} = \sum_{j\in[m],i\in[N]} \mathbb{E}_{t,\pi(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1),w^j(y_j^i|\boldsymbol{x}_0,\boldsymbol{x}_1)} \log \hat{w}_t^j(y_j^i|\boldsymbol{x}_t;\theta) \quad \text{(A.14)}
$$

Starting from Equation 3.15,

$$
\begin{aligned}
\mathcal{L}_{EDFM} &= \sum_{j\in[m],i\in[N]} \mathbb{E}_{t,\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1),w^j(y_j^i|\boldsymbol{x}_0,\boldsymbol{x}_1)} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{q_1(\boldsymbol{x}_1} \log \hat{w}_t^j(y_j^i|\boldsymbol{x}_t;\theta) \quad \text{(A.15)} \\
&= \sum_{j\in[m],i\in[N]} \mathbb{E}_{t,\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1),w^j(y_j^i|\boldsymbol{x}_0,\boldsymbol{x}_1)} Z\frac{p_{\text{ebm}}(\boldsymbol{x}_1)}{q_1(\boldsymbol{x}_1)} \log \hat{w}_t^j(y_j^i|\boldsymbol{x}_t;\theta) \quad \text{(A.16)} \\
&= \sum_{j\in[m],i\in[N]} \mathbb{E}_{t,\pi(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(\boldsymbol{x}_t|\boldsymbol{x}_0,\boldsymbol{x}_1),w^j(y_j^i|\boldsymbol{x}_0,\boldsymbol{x}_1)} Z \log \hat{w}_t^j(y_j^i|\boldsymbol{x}_t;\theta) \quad \text{(A.17)} \\
&= Z\mathcal{L}_{FM} \quad \text{(A.18)}
\end{aligned}
$$

where as in Section A.3, we have $Z = \sum_{\mathcal{X}} \exp(-\mathcal{E}(\boldsymbol{x}_1))$ and used that $\hat{\pi}(\boldsymbol{x}_0) = p_0(\boldsymbol{x}_0)q(\boldsymbol{x}_1)$, and $\pi(\boldsymbol{x}_0) = p_0(\boldsymbol{x}_0)p_{\text{ebm}}(\boldsymbol{x}_1)$. The result follows.

A.4 Proof of Proposition 4

Here we show $\nabla_\theta \mathcal{L}_{\text{VM}} = c\nabla_\theta \mathcal{L}_{\text{ECVM}^*}$. First, we recall Proposition 2, implying that it is sufficient to show $\nabla_\theta \mathcal{L}_{\text{ECVM}} = c'\nabla_\theta \mathcal{L}_{\text{ECVM}^*}$. Here, c and c' are multiplicative constants. Starting from Equation 4.3,

$$\mathcal{L}_{\text{ECVM}^*}(\theta) =$$

$$\mathbb{E}_{\mathcal{U}[t;0,1],\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{\exp(-\mathcal{E}_{q_1}(\boldsymbol{x}_1))} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2 \quad \text{(A.19)}$$

$$= \mathbb{E}_{\mathcal{U}[t;0,1],\hat{\pi}(\boldsymbol{x}_0,\boldsymbol{x}_1),p_t(x_t|\boldsymbol{x}_0,\boldsymbol{x}_1)} \sum_{i=1}^{D} \sum_{z^i=0}^{S} \frac{\exp(-\mathcal{E}(\boldsymbol{x}_1))}{Z_{q_1} q_1(\boldsymbol{x}_1)} \|\hat{u}_t^i(z^i, \boldsymbol{x}_t; \theta) - u_t^i(z^i, \boldsymbol{x}_t|\boldsymbol{x}_0, \boldsymbol{x}_1)\|_2^2 \quad \text{(A.20)}$$

$$= Z_{q_1}^{-1} \mathcal{L}_{\text{ECVM}} \quad \text{(A.21)}$$

where here $Z_{q_1} = \sum_{\mathcal{X}} \exp(-\mathcal{E}_{q_1}(\boldsymbol{x}_1))$ as before $\hat{\pi}(\boldsymbol{x}_0) = p_0(\boldsymbol{x}_0|\boldsymbol{x}_1)q_1(\boldsymbol{x}_1)$. Mutatis mutandis, we obtain the analogue statement $\nabla_\theta \mathcal{L}_{\text{DFM}} = c\nabla_\theta \mathcal{L}_{\text{EDFM}^*}$, where $\mathcal{L}_{\text{EDFM}^*}$ is the unnormalised version of Equation 3.15. Note that combining with Proposition 2 we have $c = \frac{Z_{q_1}}{Z}$.

Appendix B

Further Explorations

B.1 DFS-EBM

We briefly describe an additional idea that proved empirically unsuccessful, which is why we exclude it from the main text. We choose to still present it here as it demonstrates the breadth of exploration undertaken in this project. In DFS-EBM we consider swapping the role of the DFS and EBM in the training process compared to EBM-DFS. The intuition for this is that trained on samples from a *naive* proposal distribution as in Section 4.2 or from its *own* distribution as in Section 4.3, the DFS may fail to learn new local modes present in the energy function due to a lack of coverage of certain parts of the sample space in the proposal distribution (even if we assume full support). Thus, under the assumption that we can learn an accurate discrete flow with data-based objectives such as $\mathcal{L}_{\text{DFS}}$ or $\mathcal{L}_{\text{DFM}}$, it is plausible that instead we could train the EBM to follow the DFS. Of course, under a CD training paradigm this does not remove the need for maintaining $p_\theta \approx p_\phi$ throughout the process to ensure the availability of accurate negative samples. As explained in Section 4.3, this requirement precludes the possibility of stepping the EBM multiple consecutive times. This is not true for the DFS, as this is either learned from data or *some* proposal. We specify one realisation of such a training procedure in Algorithm B.1. Note the presence of an additional step based on $\mathcal{L}_{EDFM^*}$, which is designed to reinforce $p_\theta \approx p_\phi$ through importance sampling after the energy function has been updated. Multiple consecutive steps are feasible in this case, requiring only that the energy $\mathcal{E}_\phi^k$ for each batch be pre-computed prior to the EBM update. Furthermore, note that we mix the EBM loss between a contrastive divergence loss w.r.t. the true data distribution and w.r.t the current DFS distribution. Intuitively, this is intended to provide additional guidance for the EBM to move *towards* the data while moving *with* the DFS. This is controlled by a hyperparameter α balancing the two losses. In exploratory experiments find empirically that when the number of steps of the DFS, M_{DFS}, is low, the negative samples are too close to the positive samples leading the EBM to learn an extremely uneven energy landscape whose modes tend to lie away from the DFS's distribution. This case is shown in Figure B.1. The other case materialises when to prevent the latter we increase M_{DFS}. In that case, alignment between DFS and EBM breaks because to 'catch-up', the EBM now can only perform a single step, since any additional steps will cause a misalignment between EBM and DFS. However, the restriction to a single step does in practice not provide a close enough alignment between the EBM and the DFS at the next iteration. We find that the EBM collapses to random noise.

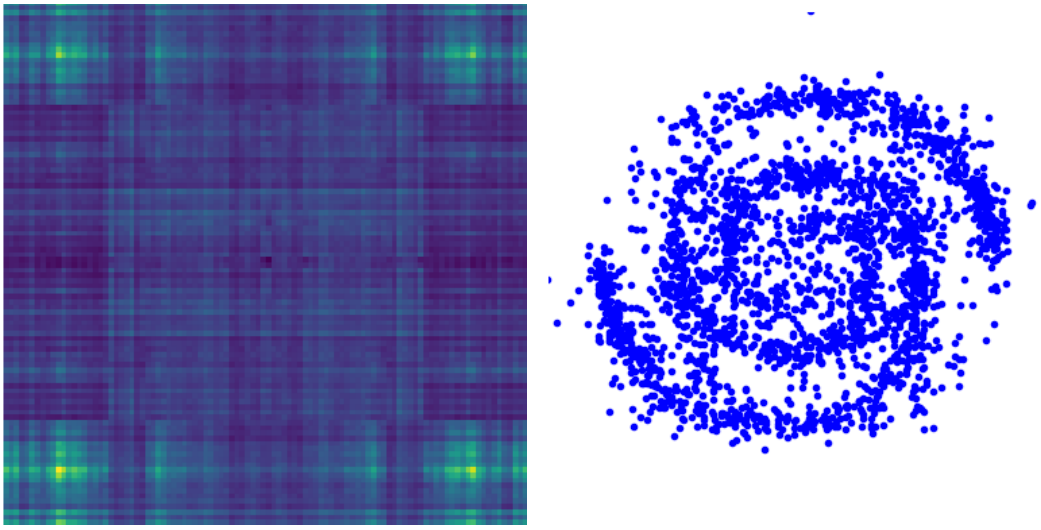


Figure B.1: DFS-EBM: Energy and DFS samples at $i = 15 \times 10^3$ and $M_{\text{DFS}} = 1$

Algorithm B.1 DFS-EBM

Require: Training dataset $\{\boldsymbol{x}_{\text{data}}\}$, DFS $u_t^i(z^i, \boldsymbol{x}_t; \theta)$, EBM $\mathcal{E}_\phi$, M_{DFS} iterations of DFS per iteration of EBM, balance parameter α

Initialise DFS's θ and EBM's ϕ s.t. $p_\theta \approx p_\phi$
$k \leftarrow 0$
repeat
  Generate batch $\boldsymbol{x}_1$ with Algorithm 2
  for $m \leftarrow 1$ **to** M_{DFS} **do**
    Sample $\boldsymbol{x}_{\text{data}} \sim p_{\text{data}}$, $t \sim \mathcal{U}[0,1]$, $\boldsymbol{x}_0 \sim p_0$, $\boldsymbol{x}_t \sim p_t(x_t|\boldsymbol{x}_0, \boldsymbol{x}_{\text{data}})$
    Step DFS via gradient of $\mathcal{L}_{CVM}$ ($\mathcal{L}_{DFM}$)
  end for
  Generate batch $\boldsymbol{x}_1'$ with Algorithm 2
  if optional step **then** Compute $\mathcal{E}_\phi^k(\boldsymbol{x}_1')$
  end if
  Compute EBM vs data loss $\mathcal{L}_{CD-data} = \mathcal{E}_\phi^k(\boldsymbol{x}_{\text{data}}) - \mathcal{E}_\phi^k(\boldsymbol{x}_1)$
  Compute EBM vs DFS loss $\mathcal{L}_{CD-DFS} = \frac{1}{S}\sum_{s=1}^{S} \mathcal{E}_\phi^k(\boldsymbol{x}_1') - \mathcal{E}_\phi^k(\boldsymbol{x}_1)$
  Step EBM via gradient of $\mathcal{L}_{EBM} = \alpha\mathcal{L}_{CD-data} + (1-\alpha)\mathcal{L}_{CD-DFS}$ (obtain $\mathcal{E}_\phi^{k+1}$)
  Sample $t \sim \mathcal{U}[0,1]$, $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x}_0)$, $\boldsymbol{x}_t \sim p_t(x_t|\boldsymbol{x}_0, \boldsymbol{x}_1')$
  Step DFS via gradient of $\mathcal{L}_{ECVM*}$ ($\mathcal{L}_{EDFM*}$) (with $\mathcal{E}_{q_1}(\boldsymbol{x}_{1,m}) = \mathcal{E}_\phi^k(\boldsymbol{x}_{1,m})$)
  $k \leftarrow k + 1$
until convergence

Appendix C

Further Experimental Results

Table C.1: CVM/DFM trained on data; Hamming and Euclidean MMD for combinations of loss, source distribution and conditional path specification; Data: Checkerboard; MMD in units of 1×10^{-4}

	Source	Hamming MMD ↓				Euclidean MMD ↓			
		linear	**quad.**	**cubic**	**noise**	**linear**	**quad.**	**cubic**	**noise**
CVM	mask	**0.330**	1.616	1.624	0.702	**0.156**	1.404	1.041	0.690
	uni	4.539	0.182	0.587	0.516	0.709	1.908	1.851	8.893
	data	0.974	0.654	0.481	0.987	1.026	0.024	1.988	8.537
DFM	mask	9.372	5.745	7.001	8.462	**0.827**	1.169	1.501	0.925
	uni	4.642	**4.630**	10.004	2.569	1.044	1.988	2.647	1.687
	data	6.426	7.370	10.793	5.835	1.034	1.968	2.729	1.780

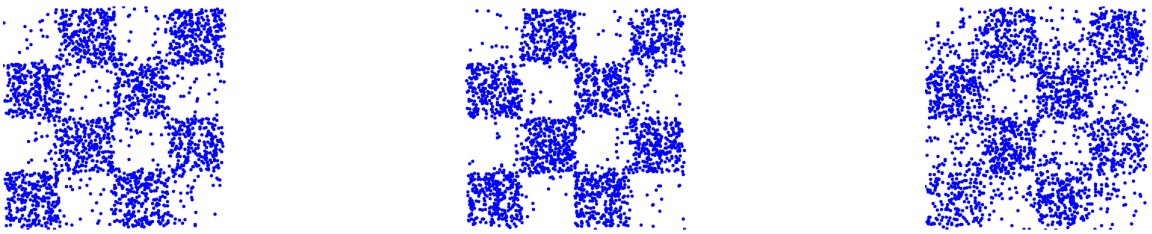


Figure C.1: Best CVM and DFM configurations according to Hamming and Euclidean MMD on checkerboard; left to right: CVM-mask-lin, DFM-mask-lin, DFM-uni-quad.

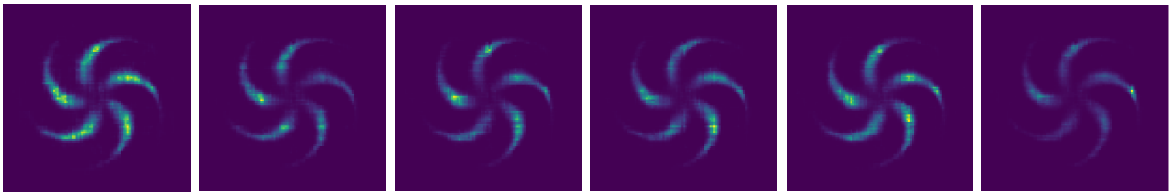


Figure C.2: Periodic mode collapse observed in training EBM-DFS with bootstrapped proposal (Section 4.3). Images show energy landscapes for pinwheel. Model trained with $M_{\text{DFS}} = 5$.

Appendix D

Further Background

D.1 Comparison of Discrete Flow Models and Discrete Flow Matching

This appendix is designed to outline parallels and differences in the closely related papers Discrete Flow Models [7] and Discrete Flow Matching [6]. Since the focus of the main text lies with the more recent latter paper, we briefly outline here the core concepts in [7] before presenting the reader with a table providing a clear juxtaposition to corresponding concepts in [6]. While CFM (2.3.3) establishes a general framework for training generative flow models, this is only applicable to continuous sample spaces. Discrete Flow Models was — to our knowledge — the first proposal, to fill this gap by exploiting the theory of continuous time Markov chains (CTMC) [7].

Aligning with the discussion in [7], we restrict our discussion here to one dimension, i.e. $D = 1$. Similar to CFM [13], the marginal probability path between p_0 and p_1, p_t, is expressed in terms of a data point conditional as $p_t(x_t) := \mathbb{E}_{p_{data}(x_1)}[p_{t|1}(x_t|x_1)]$ to make it tractable. As evident in Subsection 2.3.5, this idea was further generalised in the DFM framework. In [7], two concrete specifications are considered:

$$p_{t|1}^{unif}(x_t|x_1) = Cat(t\delta\{x_1, x_t\} + (1-t)\frac{1}{S}) \tag{D.1}$$

$$p_{t|1}^{mask}(x_t|x_1) = Cat(t\delta\{x_1, x_t\} + (1-t)\delta\{M, x_t\}) \tag{D.2}$$

Both converge to x_1 at $t = 1$. Sampling from this model is governed by Proposition D.1.

Proposition D.1 (Proposition 3.1 in [7]) *If $R_t(x_t, j|x_1)$ is a rate matrix that generates the conditional flow $p_{t|1}(x_t|x_1)$, then $R_t(x_t, j) := \mathbb{E}_{p_{1|t}(x_1|x_t)}[R_t(x_t, j|x_1)]$ is a rate matrix that generates p_t as defined above. The expectation is taken over $p_{1|t}(x_1|x_t) = \frac{p_{t|1}(x_t|x_1)p_{data}(x_1)}{p_t(x_t)}$.*

It should be noted that there are many valid choices for $R_t(x_t, j|x_1)$. A simple choice generating $p_{t|1}$ under reasonable assumptions is given by

$$R_t^*(x_t, j|x_1) := \frac{\text{ReLU}\left(\partial p_{t|1}(j|x_1) - \partial p_{t|1}(x_t|x_1)\right)}{S \cdot p_{t|1}(x_t|x_1)} \tag{D.3}$$

This can be extended to yield a CTMC with higher 'stochasticity' depending on a parameter η, as suggested by Proposition D.2:

Proposition D.2 (Proposition 3.3 in [7]) *Let R_t^{DB} be a rate matrix that satisfies the detailed balance condition for $p_{t|1}$, $p_{t|1}(i|x_1)R_t^{DB}(i,j|x_1) = p_{t|1}(j|x_1)R_t^{DB}(j,i|x_1)$, and let R_t^{η} be defined by R_t^*, R_t^{DB} and parameter $\eta \in \mathbb{R}^{\geq 0}$, $R_t^{\eta} := R_t^* + \eta R_t^{DB}$. Then we have R_t^{η} generates $p_{t|1}(x_t|x_1)$ for all $\eta \in \mathbb{R}^{\geq 0}$.*

One crucial characteristic of Discrete Flow Models is that the rate matrix is not fixed during training, but can be chosen at inference time making the process more flexible than discrete diffusion models. Instead, Discrete Flow Models are simply trained to 'denoise' p_0 based on a cross entropy loss [7]:

$$\mathcal{L}_{ce} = \mathbb{E}_{\mathcal{U}[t;0,1],p_{data}(x_1),p_{t|1}(x_t|x_1)}[log\ p_{1|t}^{\theta}(x_1|x_t)] \tag{D.4}$$

We note that Proposition D.1 corresponds exactly to Theorem L.1, taken from [6] and the idea of the CE loss in Equation D.4 was extended to the CE loss in [6]. Note further that the main adaptation made in [6] is the reframing of the rate matrix as a generating velocity to align more closely with [13] and [14]. Further differences between the two papers can be found in Table D.1.

Component	Discrete Flow Matching (DFM) [6]	Discrete Flow Models [7]
Couplings [1]	u-couplings: $\pi(x_0, x_1) = p_{\text{source}}(x_0)p_{\text{data}}(x_1)$ c-couplings: $\pi(x_0, x_1) = p_{\text{source}}(x_0\|x_1)p_{\text{data}}(x_1)$	$\pi(x_0, x_1) = p_{\text{source}}(x_0)p_{\text{data}}(x_1)$ (implicit)
Cond. Paths	$p_t(x^i\|x_0, x_1)$ s.t. $p_0(x^i\|x_0, x_1) = \delta_{x_0}(\cdot)$ and $p_1(x^i\|x_0, x_1) = \delta_{x_1}(\cdot)$ standard form: $\sum_{j=1}^m \kappa_t^{i,j} w^j(x^i\|x_0, x_1)$ with $\sum_j \kappa_t^{i,j} = 1$ and $\kappa_t^{i,j} \geq 0$ where w^i are conditional probabilities $p_t(x_t\|x_0, x_1) = \prod_{i=1}^D p_t(x^i\|x_0, x_1)$	$p_t(x^i\|x_1^i)$ s.t. $p_0(x^i\|x_1^i) = \delta_{x_0^i}(\cdot)$ and $p_1(x^i\|x_1^i) = \delta_{x_1^i}(\cdot)$ standard form: $(1-t)\delta_{x_0}(\cdot) + t\delta_{x_1}(\cdot)$ additional assumption: $p_t(x^i\|x_1^i) = 0 \implies \partial_t p_t(x^i\|x_1^i) = 0$ $p_t(x_t\|x_1) = \prod_{i=1}^D p_t(x^i\|x_1^i)$
Uncond. Paths	$p_t(x) = \sum_{x_0,x_1} p_t(x_t\|x_0, x_1)\pi(x_0, x_1)$	$p_t(x) = \sum_{x_1} p_t(x_t\|x_0, x_1)p_{data}(x_1)$
Generation [2]	$x_{t+h}^i \sim \delta_{x_t^i}(\cdot) + hu_t^i(\cdot, x_t)$ which defines a valid pmf if $u_t^i(z^i, x_t) \geq 0$ for $z_i \neq x_t^i$) and $\sum_{x^i} u_t^i(z^i, x_t) = 0$; u_t^i is called the **generating probability velocity**.	theoretically: $x_{t+h} \sim \delta_{x_t}(\cdot) + h\sum_{i=1}^D \delta_{x_t^{\bar{i}}}(\cdot)R_t(x_t, \cdot)$ practically: $x_{t+h}^i \sim \delta_{x_t^i}(\cdot) + R_t^i(x_t^i, \cdot)$ $R_t(x_t, z)$ is called the **rate matrix** and induces a valid pmf if $R_t(x_t, z) > 0$ for $x_t \neq z$ and $\sum_z R_t(x_t, z) = 0$.
Flow/Rate	$u_t^i(z^i, x_t) = \sum_{x_0,x_1} u_t^i(z^i, x_t\|x_0, x_1)p_t(x_0, x_1\|x_t)$ generates $p_t(x)$ if $u_t^i(z, x_t\|x_0, x_1)$ generates $p_t(x\|x_0, x_1)$; where the definition of u_t^i generating $p_t(x)$ is that the samples generated as above satisfy $x_{t+h} \sim p_{t+h} + o(h)$ for sufficiently small h, which amounts to satisfying Kolmogorov for $u_t(z, x_t) = \sum_{i=1}^D \delta_{x_t}(z^{\bar{i}})u_t^i(z^i, x_t)$ (and for $u_t(z, x_t\|x_0, x_1) = \sum_{i=1}^D \delta_{x_t}(z^{\bar{i}})u_t^i(z^i, x_t\|x_0, x_1)$ in the conditional case) (compare **Definition 1**, **Theorem 2**, **Theorem 3** in [6]). Closed form: $u_t^i(z^i, x_t \mid x_0, x_1) = \sum_{j=1}^m a_t^{i,j} w^j(z^i \mid x_0, x_1) + b_t^i \delta_{x_t}(x^i)$ where $a_t^{i,j} = \dot{\kappa}_t^{i,j} - \frac{\kappa_t^{i,j}\dot{\kappa}_t^{i,\ell}}{\kappa_t^{i,\ell}}$, $b_t^i = \frac{\dot{\kappa}_t^{i,\ell}}{\kappa_t^{i,\ell}}$ and $\ell = argmin_{j\in[m]}\left[\frac{\dot{\kappa}^{i,j}}{\kappa_t^{i,j}}\right]$	$R_t(x_t, z) = \sum_{i=1}^D \delta_{x_t}(x^{\bar{i}})R_t^i(x_t, z^i)$ with $R_t^i(x_t, z^i) = \sum_{x_1} R_t^i(x_t{}^i, z^i\|x_1^i)p(x_1^i\|x_t)$; This satisfies Kolmogorov (i.e. generates p_t) if $R_t^i(x_t{}^i, z^i\|x_1^i)$ satisfies Kolmogorov for each i.[4] (Compare **Proposition 3.1** and **Appendix E** in [7]) Closed form: $R_t^{*i}(x_t{}^i, z^i\|x_1^i) :=$ $\begin{cases} \frac{\text{ReLU}\left(\partial p_{t\|1}(z^i\|x_1^i) - \partial p_{t\|1}^i(x_t{}^i\|x_1^i)\right)}{\mathcal{Z}_t^i \cdot p_{t\|1}^i(x_t{}^i\|x_1^i)} & (1) \\ 0 & (2) \end{cases}$ where (1) applies for $p_t^i(x_t{}^i\|x_1^i) > 0$ and $p_t^i(z^i\|x_1^i) > 0$ and (2) otherwise and $\mathcal{Z}_t^i = \left\|\{z^i : p_{t\|1}(z^i\|x_1^i) > 0\}\right\|$
Loss	$\mathcal{L}_\theta = -\sum_{j,i} \mathbb{E}_{t,(x_0,x_1),x_t,y_j^i} \log \hat{w}_t^i(y_j^i\|x_t; \theta)$	$\mathcal{L}_\theta = -\sum_i \mathbb{E}_{t,x_1,x_t} \log \hat{p}_t^i(x_1^i\|x_t; \theta)$
Noise/DB [3]	Noise added via w^j, e.g. $p_t(x^i\|x_0, x_1) - \kappa_t^1\delta_{x_1}(x_i) + \kappa_t^2 p_u(x^i) + \kappa_t^3\delta_{x_0}(x^i)$ where $\kappa_0^1 = 0$, $\kappa_1^1 = 1$, $\kappa_0^2 = \kappa_1^2 = 0$	Noise added via additive rate matrix R^{DB} that satisfies the detailed balance condition given by $p_{t\|t}(x_t\|x_1)R_t^{DB}(x_t, z\|x_1) = p_{t\|t}(x\|x_1)R_t^{DB}(z, x_t\|x_1)$
Mask Example [4]	$p_t(x^i\|x_0, x_1) = (1-\kappa_t)\delta_M(x^i) + \kappa_t\delta_{x_1}(x^i)$ with $\kappa_0 = 0$, $\kappa_1 = 1$ and $\partial_t\kappa_t \geq 0$; $u_t^i(z^i, x_t; \theta) = \frac{\partial\kappa_t}{1-\kappa_t}\left[\hat{p}_{1\|t}(x^i\|x_t; \theta) - \delta_{x_t}(x^i)\right]$	$p_{t\|1}^{mask}(x_t^i\|x_1) = (1-t)\delta_M(x^i) + t\delta_{x_1}(x^i)$ $R_t^i(x_t, z^i) = \frac{p_{1\|t}(x^i\|x_t;\theta)}{1-t}\delta_M(x_t{}^i)$
Other	DFM framework includes **sampling backward in time** and **corrector sampling**. Corrector sampling could be used as another way to introduce noise in the process (compare section 'Noise/DB' above).	Discrete Flow Models include notion of **purity sampling** when unmasking.

Table D.1: Comparing Components of DFM and Discrete Flow Models

[1] The notion of couplings stems from DFM and does not explicitly appear in Discrete Flow Models.

[2] The DFM framework assumes from the beginning that each dimension is sampled independently, however, Discrete Flow Models only make that assumption in 'practice'. In defining flows/rates they both assume that only one dimension can change at a time. This is consolidated by allowing some $o(h)$ error between the Euler generated path and the target path. Note also that in Discrete Flow Models the rate matrix can be separated for each dimension, while in DFM the flow is always conditional on the entire current state.

[3] Note that it is assumed that $R_t(x_t, z|x_1) = \sum_{i=1}^D \delta_{x_t^{\bar{i}}}(z)R_t^i(x_t{}^i, z^i|x_1^i)$ so that we can also write $R_t(x_t, z) = \sum_{i=1}^D \delta_{x_t^{\bar{i}}}(z)R_t^i(x_t, z^i) = \sum_{i=1}^D \delta_{x_t^{\bar{i}}}(z)\sum_{x_1} R_t^i(x_t{}^i, z^i|x_1^i)p(x_1|x_t) = \sum_{x_1}\sum_{i=1}^D \delta_{x_t^{\bar{i}}}(z)R_t^i(x_t{}^i, z^i|x_1^i)p(x_1|x_t) = E_{p(x_1|x_t)}R_t(x_t, z|x_1)$.

[4] Note that in the case of a masked source distribution, when we compare with the standard detailed balance rate matrix proposed in Discrete Flow Models, the uniform noise in DFM allows independent state jumps out of the set $\{M, x_1^i\}$, while the standard DB rate matrix does not (DB must still hold).

D.2 Generative Models Overview

Generative modelling concerns the representation of a joint probability distribution $p(\boldsymbol{x})$, for $\boldsymbol{x} \in \mathcal{X}$. Recent advances in generative models, for instance in image or text generation, are based on **deep neural networks** which are used to learn relationships between latent, unobserved variables $\boldsymbol{z}$ and the observed data $\boldsymbol{x}$. Using such models to *generate* new instances of $\boldsymbol{x}$ essentially amounts to sampling from $p(\boldsymbol{x})$. The key difference in comparison to conventional samplers such as MCMC methods (2.2) is that generative models are typically **learned** from data through neural networks, while MCMC methods sample from a given pmf or energy. Popular types of generative models include **variational auto-encoders** (VAE), **generative adversarial networks**, **auto-regressive models** (ARM), **normalising flows** (NF), **diffusion models** and **energy based models** (EBM). Comprehensive introductions to all of these models can be found in the book [4]. Mixtures of these modeling paradigms are increasingly used to exploit respective advantages. [4] provides a useful classification of these models according to five criteria: Density, sampling, training, latents and architecture. We provide below a table summarising this classification (Table D.2) adapted from [4]. Note that both EBM (2.4) and flow models (2.3) appear among those methods with slow sampling properties. We augmented the table by adding in the more recent advances in conditional flow matching (CFM) (Subsection 2.3.4) and discrete flow matching (DFM) (Subsection 2.3.5).

Model	**Density**	**Sampling**	**Training**	**Latents**	**Architecture**
VAE	LB, fast	Fast	MLE-LB	$\mathbb{R}^L$	Encoder-Decoder
GAN	N/A	Fast	Min-max	$\mathbb{R}^L$	Generator-Discriminator
ARM	Exact, fast	Slow	MLE	None	Sequential
NF	Exact, slow/fast	Slow	MLE	$\mathbb{R}^D$	Invertible Flow
CFM/DFM	N/A	Slow	MLE, CE, or Regression	$\mathbb{R}^D$	Flow
Diffusion	LB	Slow	MLE-LB	$\mathbb{R}^D$	Encoder-Decoder
EBM	Approx., slow	Slow	MLE-A	Optional	Discriminative

Table D.2: Characterisation of different generative models adapted from [4]. Explanations: LB stands for lower-bound. MLE stands for maximum likelihood estimation. MLE-LB stands for lower-bound MLE. MLE-A A stands for approximate MLE. CE stands for cross-entropy. DFM stands for discrete flow matching. N/A means not available. L represents the dimensionality of the latent variable $\boldsymbol{z}$, and D represents the dimensionality of the observed variable $\boldsymbol{x}$. Usually, $L \ll D$. Our DFS method falls into Conditional Flow Matching.

Appendix E

Software Archive

The software archive can be found at https://github.com/moritzhauschulz/samplingEBMs.git. Please refer to the `README` for further details.