POLITECNICO DI TORINO
Department of Applied Science and Technology

MASTER'S DEGREE IN PHYSICS OF COMPLEX SYSTEMS

MASTER'S THESIS

# SAMPLING OF MULTI-MODAL DISTRIBUTIONS ASSISTED WITH MIXTURES OF NORMALIZING FLOWS

**Supervisors**
Prof. Gabrié Marylou
Prof. Dolcini Fabrizio
Prof. Biroli Giulio

**Candidate**
Pierannunzi Elena
290193

Academic Year 2022/2023

# Contents

# 1 Introduction

One of the core challenges in scientific computing is to have access to probability distributions which are too complex to be manipulated.

This is the case, for example, in the field of statistical inference: when dealing with the large datasets we are able to collect nowadays from almost all aspects of our lives, the high-dimensional distributions used as priors, or obtained as posteriors, are typically analytically intractable.

The same is true in statistical physics, which is founded on the study of physical observables of systems containing a very large number of interacting degrees of freedom. These quantities are accessible by means of probability measures defined on a phase space which inherits the huge dimensionality of the systems themselves.

The traditional answer to this computational issue lies in the *Monte Carlo methods* (MC). These techniques provide a way of approximating the moments of *target* probability distributions $\rho_*(x)$ by *sampling* them, while avoiding the expensive numerical integration originally required.

Despite the extremely significant contribution these methodologies have brought in computational science, there are still problems for which their performance is not efficient enough. This is the case of high-dimensional, *multi-modal* probability distributions, defined by the presence of multiple metastable *basins*, or *modes*, *i.e.* regions of high probability density isolated by areas of lower probability density (an example is shown in Figure 1).

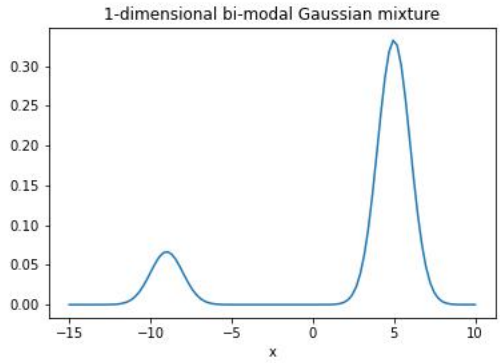



Figure 1: 1-dimensional bi-modal Gaussian mixture. Means: $\mu_1 = -9$, $\mu_2 = 5$; variances: $\sigma_1^2 = \sigma_2^2 = 1$; statistical weights of the modes: $\omega_1 = 0.2$, $\omega_2 = 0.8$.

An interesting way to deal with this challenging task is to assist MC sampling with *machine learning* (ML) techniques, such as the *generative models*, which are proved to be efficient in sampling complex high-dimensional probability distributions. This is the idea developed in the *Adaptive MC algorithm augmented with normalizing flows* proposed by [Gabrié et al., 2022], where the traditional MC local kernel is combined with a non-local one, parametrized by means of a generative model, the *normalizing flow* (NF), which is trained on the fly with the generated samples.

This method leads to a notable acceleration in sampling. Moreover, its performances can be further improved in specific cases which are often encountered in physics, like, for example, when the modes have very different fine structures or statistical weights. The idea for dealing more efficiently with these conditions is to extend the original adaptive algorithm by replacing the unique NF with a *mixture* of normalizing flows, where each component is trained to represent a single mode.

This Master's thesis aims to illustrate the theory behind this idea, its implementation and the performances of the resulting algorithm, compared to the existing methods.

The Thesis is organized as follows. In Section 2 we illustrate the background of our work: after a brief overview of the traditional MC, we point out why it fails in sampling accurately multi-modal distributions and the strategies suggested over the years for addressing this problem. Among these various proposals, special attention is paid to the Adaptive MC algorithm augmented with normalizing flows and to the specific machine learning techniques employed for its realization.

In Section 3 we get to the heart of the work around which this Thesis is centered: the *Mixture of normalizing flows* algorithm. First, we discuss its structural differences from the original Adaptive MC, that is, the ingredients required to design a combination of multiple flows, individually weighted by a parameter which is trained to adapt to the weight of a specific mode in the target. Next, we focus on the details of the code that implements these differences.

Section 4 is dedicated to the performances of the new algorithm, evaluated through the interpretation of the results obtained in three different examples, and through their comparison with the original algorithm predictions. This analysis allows us to confirm the ability of the mixture of NFs in sampling effectively all the known modes of a multi-modal distribution according to their correct statistical weights, even in a high-dimensional framework.

# 2 Markov Chain Monte Carlo algorithm

## 2.1 Classical version of the Monte Carlo algorithm

One of the best known MC techniques is the *Markov Chain Monte Carlo algorithm* (MCMC). A Markov Chain is a stochastic process where the *transition probability* from the current state to the following one is determined exclusively by the current state, and not by how the current state is attained. This feature is often referred to as *Markov* or *memoryless property*, and it is what makes a Monte Carlo algorithm of the Markov Chain type: indeed, the sequence of samples it produces satisfies this property, and, as a consequence, can be identified as a Markov Chain itself. Given so, the algorithm will sample correctly from the desired *target* distribution $\rho_*(x)$ if the *stationary* distribution of the chain is proportional to $\rho_*(x)$.

Before looking at the details of this algorithm and at how, specifically, it manages to target the right distribution, let's illustrate a possible framework where it is useful, which is the one we will stick to in this report: the statistical physics framework.

In this picture, the statistical ensemble most commonly chosen for the description of the system under analysis is the *canonical ensemble*, whose associated probability measure is the *Boltzmann* distribution[1]:

$$\rho_*(x) = \frac{e^{-\beta U_*(x)}}{Z_*}, \tag{1}$$

where $x$ is the variable describing the particular microstate of the system, and $U_*(x)$ is the *Hamiltonian*, *i.e.* the function representing the total energy of the system, which in this context will be often referred to as *potential*. Here, the computational challenge is represented by the normalization factor, called *partition function*:

$$Z_* = \sum_{x \in \Omega} e^{-\beta U_*(x)}, \tag{2}$$

which reads (in the case of discrete systems) as a sum over all the possible configurations $x$ living in the phase space $\Omega$ of the system.

The algorithm proceeds in the following way:

1. Given an initial state $x_t$, a trial move $x_{trial}$ is proposed, according to a rule that ensures the ergodicity of the algorithm (typically by following a local dynamics like the Hamiltonian one);

2. The *acceptance ratio* $\alpha$ is computed. There are different ways of defining it, the most common one is the *Metropolis-Hastings rule*:

$$\alpha(x_t, x_{trial}) = \min\left[1, \frac{\rho_*(x_{trial})}{\rho_*(x_t)} \frac{p_{x_{trial} \to x_t}}{p_{x_t \to x_{trial}}}\right]. \tag{3}$$

   The ratio $\frac{\rho_*(x_{trial})}{\rho_*(x_t)}$ is where the calculation of the partition function is avoided, by solving the original computational issue. $p_{x \to y}$ is the *proposal distribution*, *i.e.* the conditional probability of proposing a move to the state $y$ starting from $x$. This distribution is

[1] $\beta = \frac{1}{K_B T}$ is the *inverse temperature*, $K_B$ is the Boltzmann constant.

related to the *transition kernel*[2] of the Markov Chain:

$$\pi_{x\to y} = \alpha(x,y)p_{x\to y} + \delta(x,y)\int_\Omega (1-\alpha(x,y))p_{x\to y}dy, \tag{4}$$

which is carefully chosen in order to satisfy the *detailed balance* condition[3], which ensures the target distribution is the stationary distribution we are going to converge to asymptotically by following the chain [Tierney, 1998].

3. The *acceptance test* is performed. This consists of drawing a uniform random number $\xi$ between 0 and 1: if $\alpha > \xi$, $x_{trial}$ is accepted as the next state $x_{t+1}$ of the Markov chain, otherwise the state of the chain is not updated, and $x_{t+1} \leftarrow x_t$.

As anticipated, these methods struggle when dealing with multi-modal probability distributions. For the sampling of this kind of densities the local transition kernel used in MCMC is not sufficient: once the chain reaches a mode, it will commonly get stuck there, and the probability landscape will not be explored in its entirety. This happens because the local moves proposed along the chain will not be sufficiently far to reach a different mode, and the same result could not be achieved by a series of subsequent steps: this would correspond to cross a lower probability region, which is strongly disfavoured by the Metropolis-Hastings rule. Using more chains, one for each of the different modes, would not be a proper solution, because we would still be unable to capture the relative statistical weights of the basins.

Over the years, different approaches have been proposed to tackle this problem. Probably, the most popular is based on *tempering*, where different replicas of the system at different temperatures are simulated independently and simultaneously, and configurations are exchanged between replicas: this procedure has the effect of warming up and cooling down a state, which in this way is able to overcome low probability barriers [Hukushima and Nemoto, 1996]. The main drawback of such method lies in its significant computational expensiveness.

More recently, in accordance with the increasing interest and development in machine learning techniques, different *adaptive* variants of MCMC have been designed, where the transition kernel is optimised while the algorithm run, using the samples previously generated in the chain. In the context of statistical physics, one of the first attempts in this direction has been done by [Albergo et al., 2019]: there, Markov chain autocorrelation times are systematically improved thanks to an update scheme which uses a flow-based generative model optimized through the minimization of the *reverse Kullback-Leibler divergence*. A similar attempt has been done in the context of *Variational Inference*[4] by [Zhang et al., 2022], who focus on the optimisation of a specific version of MCMC, the *Hamiltonian Monte Carlo* algorithm, by means of an adaptive map which is improved by minimizing the *direct Kullback-Leibler divergence*.

[2] It is a map that describes the probabilities of specific transitions. In the particular case of a Markov process with a finite state space, it coincides with the *transition matrix*, whose entries correspond to the *transition probabilities* between each pair of states.

[3] $\rho_*(x)\pi_{x\to\bar{y}} = \rho_*(\bar{y})\pi_{\bar{y}\to x}$.

[4] It is a group of techniques devoted to find the best approximation of a probability distribution among a parametrised family, following an optimisation procedure over the parameters that only requires the knowledge of the target distribution up to a factor.

The *Adaptive MC algorithm augmented with normalizing flows* by [Gabrié et al., 2022] follows this trend, and its novelty lies in the choice of keeping, along with the new adaptive kernel, the original local one, useful to improve the robustness of the algorithm.

## 2.2 Adaptive Monte Carlo augmented with normalizing flows

As mentioned previously, this algorithm optimizes MCMC sampling by means of a *normalizing flow*, which is an example of what in machine learning is called a *generative model*.

Devising a generative model is an *unsupervised*[5] ML task that aims at learning a representation of an intractable probability distribution $\rho_*(x)$, typically defined on a high-dimensional space, using its samples as training data. Differently from classic statistical inference, where one looks for a mathematical expression for the target distribution, generative modeling has the goal of finding a transformation $T$ that maps samples $z$ from a tractable distribution $\rho_B(z)$ (*e.g.*, a standard normal), called *latent* or *base* distribution, into samples $x$ of the target distribution, in such a way that $T(z) \approx x$ [Ruthotto and Haber, 2021].

In the case of a normalizing flow, the transformation $T : \Omega \to \Omega$ we look for must be *invertible* (*i.e.* the inverse map $\bar{T}$ has to satisfy the condition $\bar{T}(T(x)) = T(\bar{T}(x)) = x$) and *differentiable* [Papamakarios et al., 2019].

### 2.2.1 Sampling

The combination of the local kernel with a global one $\pi_T$ (built to satisfy detailed balance as well) reads as replacing the original $\pi$ with a new

$$\hat{\pi}(x, y) = \int_\Omega \pi(x, z)\pi_T(z, y)dz. \tag{5}$$

Using this kernel reflects in practice in making the chain alternately evolve according to:

- $\pi$: the moves are proposed following a local dynamics (*e.g.* Langevin dynamics) and are accepted or rejected according to Eq. (3);
- $\pi_T$: the moves $x_{trial} = T(z)$, $z \sim \rho_B$, are proposed following the non-local rule defined by the normalizing flow, and are accepted or rejected according to the following:

$$\hat{\alpha}(x_t, x_{trial}) = \min\left[1, \frac{\rho_*(x_{trial})}{\rho_*(x_t)}\frac{\hat{\rho}(x_t)}{\hat{\rho}(x_{trial})}\right], \tag{6}$$

where

$$\hat{\rho}(x) = \rho_B(\bar{T}(x)) \det|\nabla_x \bar{T}| \tag{7}$$

is the *push-forward* distribution, which pushes the samples from the trivial $\rho_B(z)$ through $T$. This map is trained in such a way that $\hat{\rho}(x) \approx \rho_*(x)$, but even if the adaptation to the target is not perfect, the acceptance test makes it possible to draw samples with the correct statistical weights.

[5] *Unsupervised* ML algorithms have the goal of finding a structure, or patterns in the unlabeled input data they are provided with. They differ from *supervised* ML algorithms, which are trained with labeled data with the aim of performing classification tasks.

The idea of using two cooperating kernels is present also in [Pompe et al., 2018], where the multi-modality issue is addressed by combining a similar local kernel with a global one which takes care of making the chain jump between regions explicitly associated with different basins.

### 2.2.2 Implementation of the normalizing flow

The normalizing flow map $T$ is devised as a *Real-valued Non-Volume Preserving transformation* (Real NVP) [Dinh et al., 2016].

This transformation corresponds explicitly to Eq. (7), and the choice for its parametrization is related to the accessibility of the computation of the Jacobian determinant $\det|\nabla_x \bar{T}|$: in order for it to be easily tractable, the map is built by stacking a series of simple bijections called *affine coupling layers* (ACL), characterized by triangular Jacobian matrices, whose determinants are the trivial product of the diagonal entries.

In particular, an ACL takes a $D$-dimensional input $x$ and computes the output $y$ in the following way:

- the first $d < D$ entries are not modified:

$$y_{1:d} = x_{1:d}; \tag{8}$$

- the last $D - d$ entries are transformed according to:

$$y_{d+1:D} = x_{d+1:D} \odot e^{s(x_{1:d})} + t(x_{1:d}), \tag{9}$$

  where $\odot$ is the element-wise product, and $s$ and $t$ are, in turn, further maps, respectively called *scale* and *translation*.

The $s$ and $t$ transformations are parametrized by *multi-layer perceptrons* (MLP), which are the elementary blocks of deep neural networks. Further details about their structure and how they work are illustrated in Appendix A.

### 2.2.3 Training

The parameters $\bar{\theta}$ of the whole flow, which we can now refer to as $T_{\bar{\theta}}$ (*i.e.* the combination of all the coupling layers, each of them including the single $s_{\bar{\theta}}$ and $t_{\bar{\theta}}$ maps), are optimised by means of the *Gradient Descent algorithm*: this algorithm minimizes an objective function, called *loss function* $L(\bar{\theta})$, which quantifies the discrepancy between the target distribution $\rho_*(x)$ and the push-forward $\hat{\rho}_{\bar{\theta}}(x)$.

This is done during *training*, which is here executed concurrently with the sampling described previously. At each iteration $k$ of the training, the algorithm performs $n$ Monte Carlo steps (following in part the local kernel, in part the non-local one) and the resulting samples $x_i(k)$ are used to compute the loss function $L(\bar{\theta})$, here chosen as the *direct Kullback-Leibler divergence*[6] between $\rho_*(x)$ and $\hat{\rho}_{\bar{\theta}}(x)$:

[6] $D_{KL}(p(x)||q(x))$ is a non-symmetric measure of how two probability distributions $p(x)$ and $q(x)$ are different. In optimization problems, where $p(x)$ is the target distribution and $q(x)$ is the approximate distribution, its *direct* version is distinguished from the *reverse* $D_{KL}(q(x)||p(x))$ by the fact that $p(x)$ and $q(x)$ are switched. As a consequence, the direct $D_{KL}$ reads as an expectation over the target distribution, while the reverse is computed as an expectation over the approximate distribution. In the case of Eq. (10), $c = \int \rho_*(x) \log \rho_*(x)$ is a constant which does not depend on the parameters of the network $\bar{\theta}$, and hence is not interesting to compute for the improvement of the network.

$$
\begin{aligned}
L(\bar{\theta}) = D_{KL}(\rho_*(x)||\hat{\rho}_{\bar{\theta}}(x))(\bar{\theta}) &= \int \rho_*(x) \log \frac{\rho_*(x)}{\hat{\rho}_{\bar{\theta}}(x)} dx \\
&= c - \int \rho_*(x) \log \hat{\rho}_{\bar{\theta}}(x) dx \\
&\approx c - \frac{1}{n} \sum_{i=1}^{n} \log \hat{\rho}_{\bar{\theta}}(x_i(k)).
\end{aligned}
\tag{10}
$$

This loss function is then derived with respect to all the parameters, accordingly to the *back-propagation algorithm*, and the resulting gradient $\nabla_{\bar{\theta}} L(\bar{\theta})$ is used to optimize the map by updating:

$$
\bar{\theta} \leftarrow \bar{\theta} - \eta \nabla_{\bar{\theta}} L(\bar{\theta}), \tag{11}
$$

where $\eta$ is the *learning rate*, *i.e.* the size of the steps taken to approach the minimum of the function.

# 3 Mixture of normalizing flows: model and methods

## 3.1 The model

As stated in the *Introduction*, the unique NF in the adaptive MCMC in [Gabrié et al., 2022] may not be sufficient for sampling accurately modes with highly different statistical weights. Hence, in the mixture of normalizing flows algorithm the original NF is replaced with a mixture of different NFs, one for each known mode of the target distribution.

If we think of a multi-modal distribution as a weighted superposition of separated distributions $\rho_{*,i}(x)$, one describing each of the $N$ basins:

$$\rho_*(x) = \sum_{i=1}^{N} \omega_{*,i}\rho_{*,i}(x), \text{ with } \sum_{i=1}^{N} \omega_{*,i} = 1, \tag{12}$$

then, devising a mixture of flows would read as replacing the original push-forward distribution $\hat{\rho}_{\bar{\theta}}(x)$ with a new one:

$$\hat{\rho}_{\bar{\theta},\omega}(x) = \sum_{i=1}^{N} \omega_i\hat{\rho}_{\bar{\theta},i}(x), \text{ with } \sum_{i=1}^{N} \omega_i = 1, \tag{13}$$

which mimics the structure of the target distribution.

By doing so, we let each of the flows adapt to the single modes' distributions by following the same procedure described earlier, while concurrently learning, further than the original parameters $\bar{\theta}$, the statistical weights $\omega_i$ that characterize the modes. This choice allows the model to detect and sample also the so-called *weak modes*, whose weights are significantly smaller with respect to the other ones. By using a single flow, in fact, the risk is that the whole probability mass concentrates onto the predominant modes, forgetting the "lighter" ones; if, instead, we devote a specific flow to each of the basins, then all of them keep being accounted for during training, even when their relative weights reach small values.

## 3.2 Implementation of the model: network structure and code specifics

The code has been implemented in Python [Van Rossum and Drake, 2009], using the machine learning library *PyTorch* [Paszke et al., 2019]. The network is designed as a list of RealNVPs, each of them implemented according to the code for the single flow devised in [Gabrié et al., 2022].

### 3.2.1 Parameters

As mentioned before, the new network is associated to a set of parameters $\{\bar{\theta}, \omega\}$, where $\bar{\theta} = \{\bar{\theta}_i, i = 1, ..., N\}$ correspond to the original parameters of the RealNVPs, while $\omega = \{\omega_i, i = 1, ..., N\}$ are the new weights.

In particular, the actual additional parameters defined for this network are not the weights themselves, but their logarithm $\log \omega_i$: this ensures the real $\omega_i = e^{\log \omega_i}$ remain always positive during training, even if the parameters are updated to negative values.

The information about the statistical weights associated to each flow is accounted for by the model in the *sampling* and the *loss function*. In particular, when it comes to sampling, the

number $n$ of samples to be drawn is split into $N$ values $n_i$ (such that $\sum_{i=1}^{N} n_i = n$) by sampling a *multinomial distribution*: in this way we are able to draw from the entire distribution while respecting the current weights of the modes. Concerning the loss function (which is just an adapted version of Eq.(10)):

$$L(\bar{\theta}, \omega) = D_{KL}(\rho_*(x)||\hat{\rho}_{\bar{\theta},\omega}(x))(\bar{\theta}, \omega) = \int \rho_*(x) \log \frac{\rho_*(x)}{\hat{\rho}_{\bar{\theta},\omega}(x)} dx = c - \int \rho_*(x) \log(\hat{\rho}_{\bar{\theta},\omega}(x)) dx, \tag{14}$$

here the weights are taken into account by simply expliciting $\hat{\rho}_{\bar{\theta},\omega}(x)$ according to expression (13).

### 3.2.2 Training

The training algorithm is the same used in the original version of the model, *i.e.* the one involving a unique NF devised in [Gabrié et al., 2022], where the *Adam optimization algorithm*[7] is chosen.

### 3.2.3 Initialization

Concerning the initialization of the model before training, two features have to be specified:

- the initial weights of the modes: a neutral choice would be to assign to each mode the same weight, but if we have some prior knowledge about the distribution of the probability mass in the target, we can ease the learning by setting the initial weights to values qualitatively similar to the target ones;
- the base or *prior* distribution: according to the shape of the target, if known, different choices can be made for the prior distribution. In this code, the options are:
  - *'standn'*: $\rho_B(x)$ is a mixture of multivariate standard Gaussians, with 0 means and unitary covariances;
  - *'white'*: $\rho_B(x)$ is a mixture of multivariate Gaussians with means and covariances given by the user;
  - *'coupled'*: $\rho_B(x)$ is designed to target the probability measure $\rho_*[\phi] = \frac{e^{-\beta U_*[\phi]}}{Z_*}$ associated to the *stochastic Allen-Cahn model*. It is still a mixture of multivariate Gaussians, with means given by the user and covariances built in such a way to encode the spatial coupling of the field $\phi$ in $U_*[\phi]$.

The means and covariances are provided by the user when it is possible to perform a "preliminary training" to inform the prior distribution. An option would be to take:

[7] Adam is a widely used variant of *stochastic gradient descent* algotihm (SGD), where the learning rate is adapted to each parameter using the training data, leading to a speed-up in the convergence [Kingma and Ba, 2014]. SGD is, in turn, a variant of the classic GD algorithm, which differs from it in how much data is used to compute the gradient. In the classic GD, the whole training dataset is used, while SGD performs a parameter update for each training data-point. This variant is less accurate and could result in heavy fluctuations of the loss function, but it is much faster [Ruder, 2016].

- *GD initial means*: computed by the GD algorithm (implemented through the *automatic differentiation* function provided by PyTorch [Paszke et al., 2017]). Starting from a point $x_0$ in the vicinity of the interested mode, the algorithm iterates for $n_{steps}$ with the goal of minimizing the potential $U_*(x)$ of the target. The value found at the last step is taken as the $x$ where the potential is minimum, hence where the associated probability measure has a maximum: this can be assumed as the center of a mode $\mu_i$;

- *MALA initial covariances*: estimated as *sample covariances*

$$\Sigma_i = \frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})(x_j - \bar{x})^T \tag{15}$$

  where $\bar{x} = \mu_i$ is the mean of the $i$-th mode (either the exact or the GD one), and the $x_j$ are samples obtained by letting $n$ random walkers evolve starting from the $\mu_i$ according to the *Metropolis-adjusted Langevin algorithm* (MALA)[8].

[8] MALA is a MCMC algorithm where the moves are proposed according to the *overdamped Langevin equation* (in $1d$):

$$\frac{dx}{dt} = -\frac{1}{\gamma}\frac{dU_*(x)}{dx} + \xi(t) = -\frac{1}{\gamma}\frac{d}{dx}(-\frac{1}{\beta}\log\rho_*(x)) + \xi(t), \tag{16}$$

where $\gamma$ is the *friction coefficient* and $\xi(t)$ is gaussian white noise. The acceptance test is still performed according to Metropolis-Hastings rule.

# 4 Results and discussion

## 4.1 1-dimensional Gaussian mixture

With the aim of testing the performances of the model, we first started studying the simplest possible case: the 1-dimensional Gaussian mixture.

A Gaussian mixture is a multi-modal Gaussian distribution, which can be interpreted as a superposition of uni-modal Gaussian distributions centered in different points of the space. Before proceeding with the illustration of the training results, it is useful to focus on the structure of the loss function, in order to point out the importance of the *normalization* of the weights $\omega_i$ of the network.

**Loss function analysis** As mentioned before, the loss function is what in machine learning quantifies the difference between the current state of the model and the target, informed by data. Given so, its analysis, and in particular the visualization of its landscape in the space of the parameters, gives us an insight about whether the devised model will learn accordingly to our expectations.

Considering that the original code for the single NF behaves correctly, the only concern for having a good performance in our model is related to the new parameters $\omega_i$, which must:

- respect the proper normalization: $\sum_{i=1}^{N} \omega_i = 1$;
- respect the correct target ratio (for bi-modal distributions: $\frac{\omega_2^*}{\omega_1^*} = \frac{\omega_2}{\omega_1}$).

The key for respecting these two requirements is to account for the normalization of the weights in two points of the code:

(a) after each GD step of the training: once the $\omega_i$ are updated, we modify them manually by dividing them by their sum;

(b) during the GD step: indeed, it is not sufficient to let the parameters evolve freely and to normalize them only after the optimization, as specified in (a), but we need to constrain the optimization itself to this condition. This means the weights in $L(\bar{\theta}, \omega)$ should be specified as $\frac{\omega_i}{\sum_{i=1}^{N} \omega_i}$.

In the following we show how this happens by looking at the landscape of the loss function for the 1-dimensional Gaussian mixture.

Let's take:

- a 1-dimensional bi-modal Gaussian target distribution

$$\rho_*(x) = \omega_1^* \mathcal{N}_1(x; \mu_1^*, \sigma_1^{*2}) + \omega_2^* \mathcal{N}_2(x; \mu_2^*, \sigma_2^{*2}), \tag{17}$$

  with $\mathcal{N}_i(x; \mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{\frac{(x-\mu_i)^2}{2\sigma_i^2}}$, means: $\mu_1^* = -9$, $\mu_2^* = 5$, variances: $\sigma_1^{*2} = \sigma_2^{*2} = 1$, weights: $\omega_1^* = 0.2$, $\omega_2^* = 0.8$ (illustrated in Fig. 1);

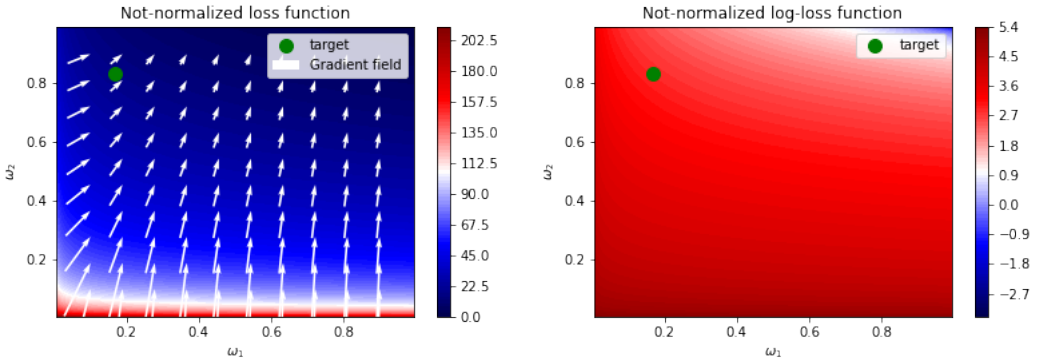
- a 1-dimensional bi-modal Gaussian prior distribution, initialized in the exact target means and covariances

$$\rho_B(x) = \omega_1 \mathcal{N}_1(x; \mu_1^*, \sigma_1^{*2}) + \omega_2 \mathcal{N}_2(x; \mu_2^*, \sigma_2^{*2}). \tag{18}$$

Satisfying only (a) corresponds to using the *not-normalized* loss function, where:

$$\hat{\rho}_{\bar{\theta},\omega}(x) = \sum_{i=1}^{N} \omega_i \hat{\rho}_{\bar{\theta},i}(x) = \omega_1 \hat{\rho}_{\bar{\theta},1}(x) + \omega_2 \hat{\rho}_{\bar{\theta},2}(x). \tag{19}$$

The associated landscape for the loss function in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane is the one represented in Figure 2. We observe that its minimum is realized for values of $\omega_1, \omega_2$ which do not seem to coincide with the target ones (in green). By manually normalizing the weights after each GD step, projecting them back to the straight line $\omega_2 = 1 - \omega_1$ where this condition is enforced, the model might in the end converge to the target, but excessively slowly.




(a) (b)

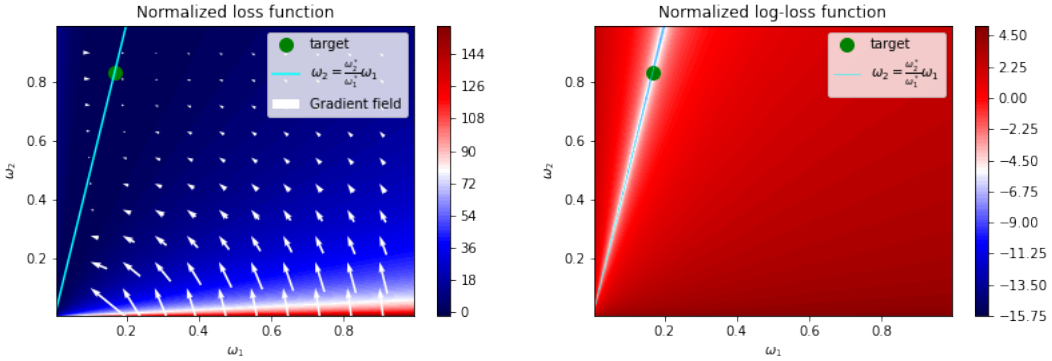
Figure 2: (a): Not-normalized loss function landscape in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane. (b): Logarithm of the not-normalized loss function landscape in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane. The green dot represents the couple of target weights $(\omega_1^*, \omega_2^*)$; the white arrows represent the gradient vector field $\nabla_{\omega_1,\omega_2} L(\bar{\theta}, \omega)$.

Conversely, if we constrain the GD itself to the normalization as suggested in (b), by using the *normalized* $L(\bar{\theta}, \omega)$, where:

$$\hat{\rho}_{\bar{\theta},\omega}(x) = \sum_{i=1}^{N} \frac{\omega_i}{\sum_{i=1}^{N} \omega_i} \hat{\rho}_{\bar{\theta},i}(x) = \frac{\omega_1}{\omega_1 + \omega_2} \hat{\rho}_{\bar{\theta},1}(x) + \frac{\omega_1}{\omega_1 + \omega_2} \hat{\rho}_{\bar{\theta},2}(x), \tag{20}$$

the picture is far more promising. In Figure 3a we observe that, indeed, in this case the loss function captures the correct weight ratio: as it is more evidently showed in the plot of the logarithm in Fig. 3b, the minimum is located precisely along the straight line $\omega_2 = \frac{\omega_2^*}{\omega_1^*} \omega_1$. This allows the GD to follow the appropriate direction. Notice that ensuring the normalization in the loss function computation does not mean that the learned weights are straightforwardly normalized too: this is the reason why (a) and (b) must be implemented both.

There is, however, a third option for the computation of the loss function, which allows the network to learn both the correct ratio and the normalization with a unique manipulation. In fact, it is possible to add to the normalized $L(\bar{\theta}, \omega)$ a *penalty* contribution that in principle, as it is done in the *Lagrange multiplier method*, constrains the learning to satisfy the normalization




(a) (b)

Figure 3: (a): Normalized loss function landscape in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane. (b): Logarithm of the normalized loss function landscape in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane. The green dot represents the couple of target weights $(\omega_1^*, \omega_2^*)$; the light blue straight line has slope equal to the target weight ratio; the white arrows represent the gradient vector field $\nabla_{\omega_1,\omega_2} L(\bar{\theta}, \omega)$

without resorting to (a). The resulting expression would be:

$$L(\bar{\theta}, \omega; p) = D_{KL}(\rho_*(x)||\hat{\rho}_{\bar{\theta},\omega}(x))(\bar{\theta}, \omega) + p\left(\sum_{i=1}^{N} \omega_i - 1\right)^2, \tag{21}$$

where $p$ is the *penalty coefficient*, which can be tuned in order to enforce the normalization condition more or less strongly. Minimizing this new loss function would then mean to find the parameters $\omega_i$ that simultaneously minimize the original normalized $D_{KL}$ and the penalty contribution $(\sum_{i=1}^{N} \omega_i - 1)^2$, hence the parameters that satisfy the condition $\sum_{i=1}^{N} \omega_i = 1$.

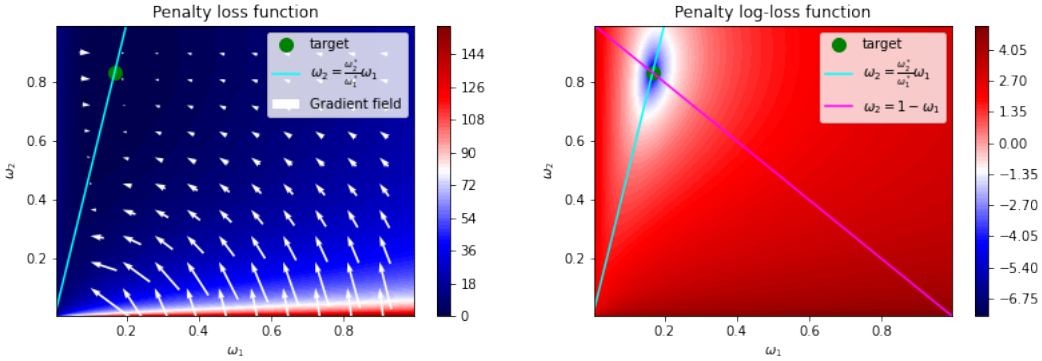
This is confirmed by the plots represented in Fig. 4: now the minimum of the loss function is found at the exact intersection between the straight line associated to the target weight ratio, and the one associated to the normalization condition.

From the training experiments it emerges that, in practice, using this variation of the loss function allows us to omit the manipulation described in (a) only if we accept an error on the sum of the weights of the order of 0.01. Also, this modification does not speed up the convergence of the algorithm, so in the following we will stick to the original normalized loss function.

**Training on a *weak mode* distribution** Now that the details of the construction of the loss function have been provided, we can focus on the model performances. In this first test we use the following parameters:

- Target distribution, represented in Fig. 5a: means $\mu_1^* = -9$, $\mu_2^* = 5$; variances $\sigma_1^{*2} = \sigma_2^{*2} = 1$; weights $\omega_1^* = 0.01$, $\omega_2^* = 0.99$;
- Initial configuration of the push-forward distribution, which corresponds to the prior distribution[9], represented in Fig. 5b: *'white'* distribution with means $\mu_1 = -5.4$,

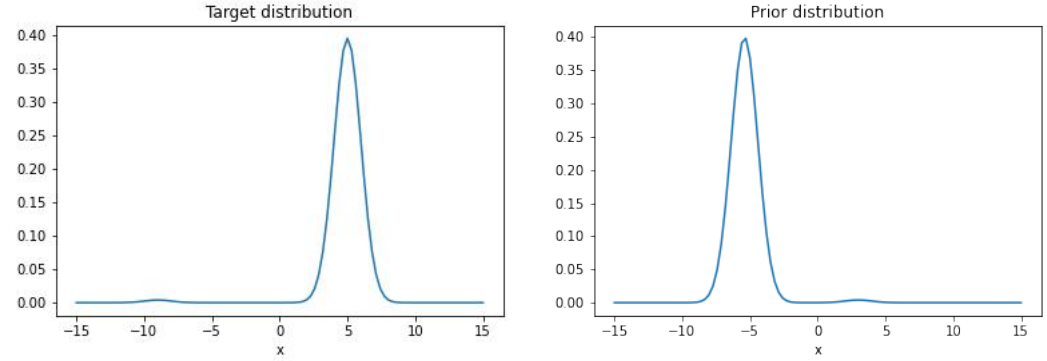
[9]The maps $T_{\bar{\theta}}$ are initialized to the identity transformation, so, before training, Eq.(7) becomes $\hat{\rho}_{\bar{\theta}}(x) = \rho_B(x)$.




(a) (b)

Figure 4: (a): Penalty loss function landscape in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane. (b): Logarithm of the penalty loss function landscape in the $(\omega_1, \omega_2) \in \mathbb{R}^2$ plane. The green dot represents the couple of target weights $(\omega_1^*, \omega_2^*)$; the light blue straight line has slope equal to the target weight ratio; the pink straight line represents the normalization condition; the white arrows represent the gradient vector field $\nabla_{\omega_1,\omega_2} L(\omega, \theta)$

$\mu_2 = 3$; variances[10] $\sigma_1^2 = 1.03, \sigma_2^2 = 1.04$; weights $\omega_1 = 0.99$, $\omega_2 = 0.01$. These initialization values are chosen in order to verify whether network is able to adapt to the target even starting from a very different configuration;




(a) (b)

Figure 5: (a): Target distribution for the training test on a 1-dimensional Gaussian mixture. (b): Prior distribution for the training test on a 1-dimensional Gaussian mixture.

- Training parameters: learning rate $\eta = 0.01$, batch size $bs = 300$, number of iterations $n = 5000$.

The evolution of the push-forward distribution during training can be observed in Figure 6: unfortunately, due to the weak nature of the mode we are trying to sample, the variations

[10] These values are obtained as sample covariances over 10 random walkers starting from the exact means and evolving for 10000 MALA steps.

in the push-forward at different iterations are not always visually appreciable. For this reason, it is better to look at the evolution of the weights values in Fig. 7: here we can see that the network is indeed learning, since the parameters are changing in the correct direction, and that already at the 1000-th iteration most of the adaptation to the target has been completed. After that, the weights keep approaching the target values with increasing accuracy, until the last iteration, when the correspondence to $\omega_{*,1} = 0.01$, $\omega_{2,*} = 0.99$ is reached with a negligible error of $\pm 0.0001$. The comparison between the final configuration of the push-forward distribution and the target one is shown in Figure 8.

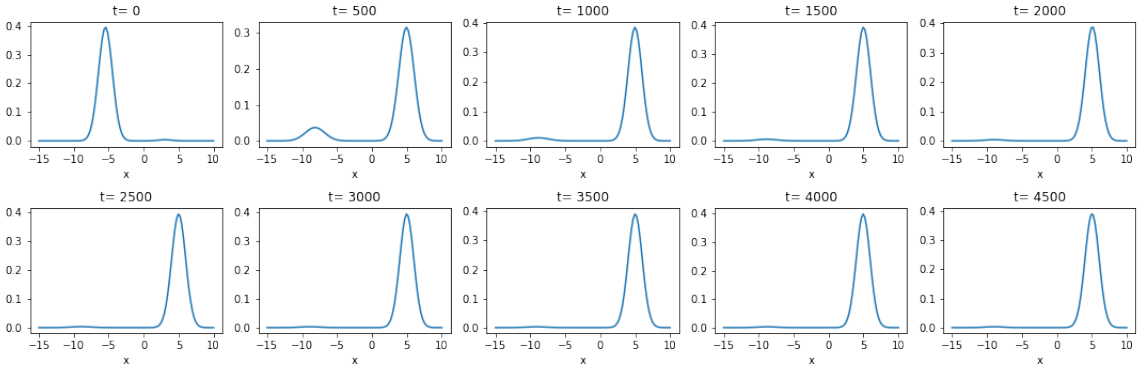



Figure 6: Evolution of the push-forward distribution during the training test on a 1-dimensional Gaussian mixture.

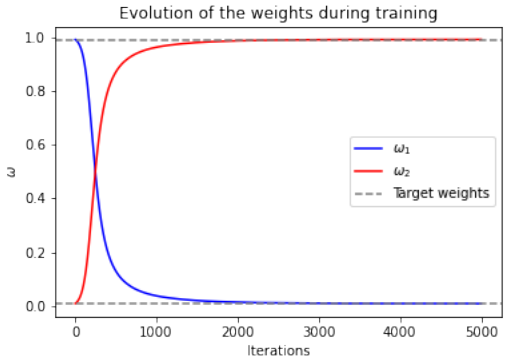



Figure 7: Evolution of the weights of the 1-dimensional Gaussian mixture during training.

In Fig. 9 we can look at the evolution of the loss function value: coherently to what we see in Fig. 7, we observe a fast significant drop in the first $\approx 1000$ iterations, and after that it remains approximately constant until the end of the training, indicating that learning continues more slowly.

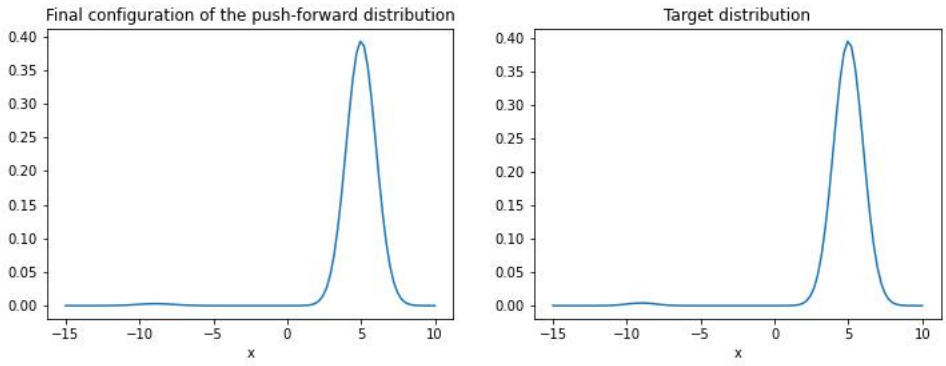



Figure 8: Comparison between the final configuration of the push-forward distribution and the target for the training test on a 1-dimensional Gaussian mixture.

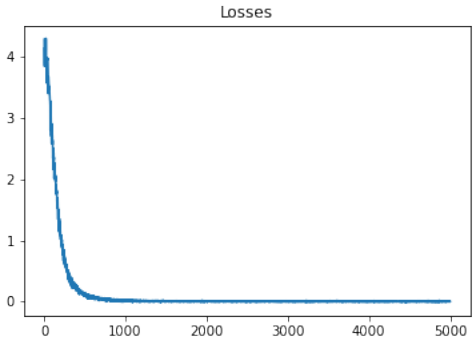



Figure 9: Plot of the loss function values during the training test on a 1-dimensional Gaussian mixture.

### 4.2 2-dimensional Gaussian mixture

**Training on a *weak mode* distribution** The second test involved a 2-dimensional Gaussian mixture.

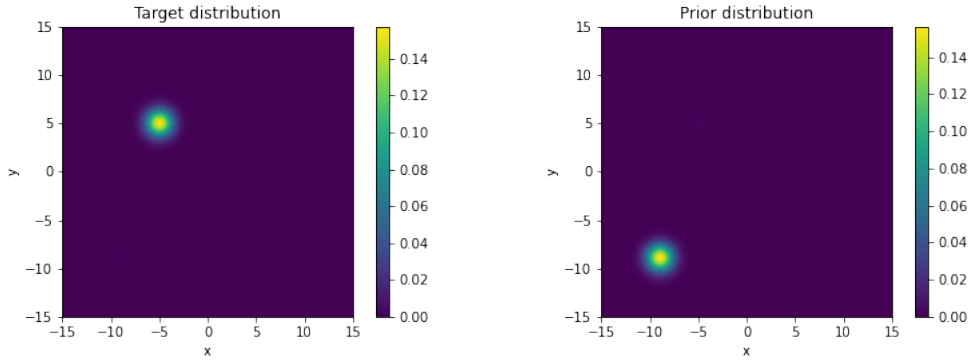
In this case we use the following parameters:

- Target distribution, represented in Fig. 10a: means $\mu_1^* = (-9, -9)$, $\mu_2^* = (-5, 5)$; variances $\Sigma_1^* = \Sigma_2^* = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$; weights $\omega_1^* = 0.01$, $\omega_2^* = 0.99$;
- Prior distribution, represented in Fig.10b: *'white'* distribution with means[11] $\mu_1 = (-9, -9)$, $\mu_2 = (-5, 5)$; covariance matrices[12] $\Sigma_1 = \left(\begin{smallmatrix} 0.99 & 0.00 \\ 0.00 & 0.99 \end{smallmatrix}\right), \Sigma_2^2 = \left(\begin{smallmatrix} 1.00 & 0.00 \\ 0.00 & 1.00 \end{smallmatrix}\right)$; weights $\omega_1 = 0.99$, $\omega_2 = 0.01$;
- Training parameters: learning rate $\eta = 0.01$, batch size $bs = 300$, number of iterations

[11] Values found by running GD for 10000 steps with $dt = 0.1$ starting from $\mu_1 = (-5.4, -5.4)$, $\mu_2 = (3, 3)$. The resulting values correspond exactly to the target ones.

[12] Values obtained as sample covariances over 10 random walkers starting from the GD means and evolving for 10000 MALA steps.

$n = 5000$.




(a) (b)

Figure 10: (a): Target distribution for the training test on a 2-dimensional Gaussian mixture. (b): Prior distribution for the training test on a 2-dimensional Gaussian mixture.

The evolution of the push-forward distribution during training can be observed in Figure 11: similarly to what happens in the test on the 1-dimensional Gaussian mixture, also here the variations in the push-forward are not actually visible. Hence, for determining whether the network is learning according to our expectations, we can look at the evolution of the weights values in Fig. 12: the adaptation to the target is occurring, and it is qualitatively evident already in the first 1000 steps; at the last iteration the exact value of the target ratio is reached.

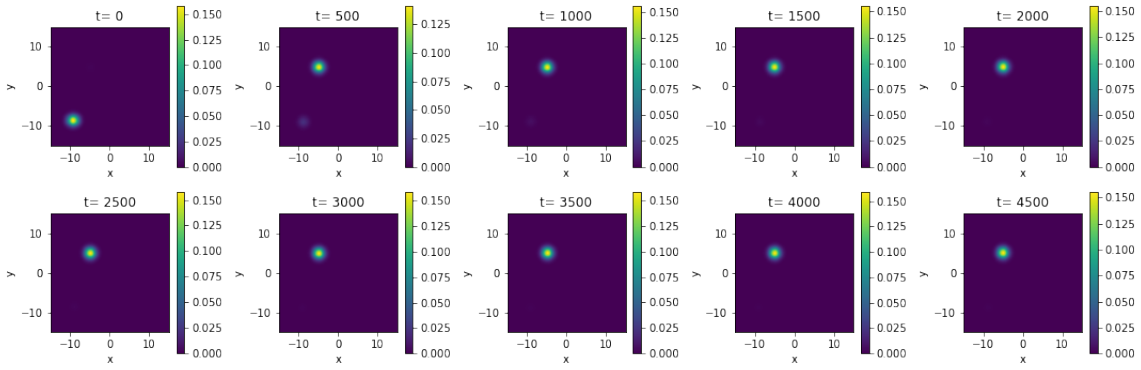



Figure 11: Evolution of the push-forward distribution during the training test on a 2-dimensional Gaussian mixture.

This is confirmed by Figure 13, where we can see the comparison between the final configuration of the push-forward distribution and the target one.

From Figure 14 we observe that the loss function, after a sudden increase in the very first iterations, decreases smoothly and reaches an approximately constant value around iteration

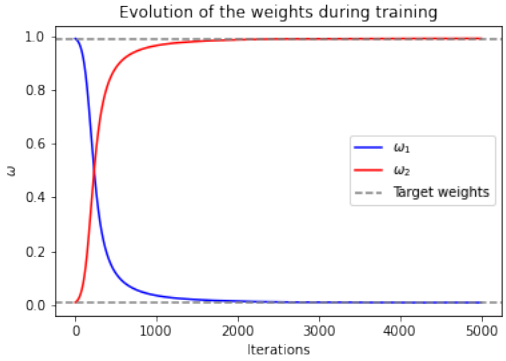



Figure 12: Evolution of the weights of the 2-dimensional Gaussian mixture during training.

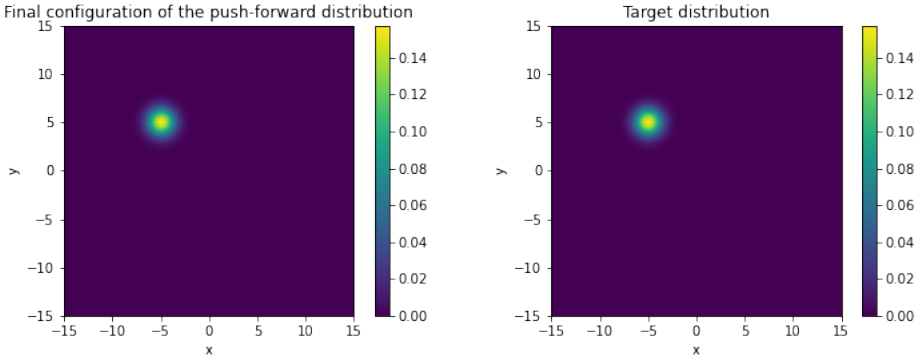



Figure 13: Comparison between the final configuration of the push-forward distribution and the target for the training test on a 2-dimensional Gaussian mixture.

n° 1000: this suggests that, accordingly to what we see in Fig. 12, at this point the push-forward has adapted qualitatively to the target, and in the following iterations, as it happened in the 1-$d$ case, it keeps improving, but more slowly.

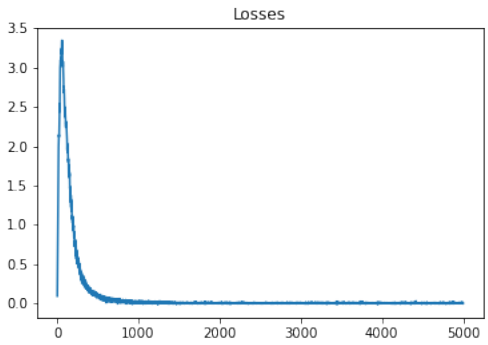
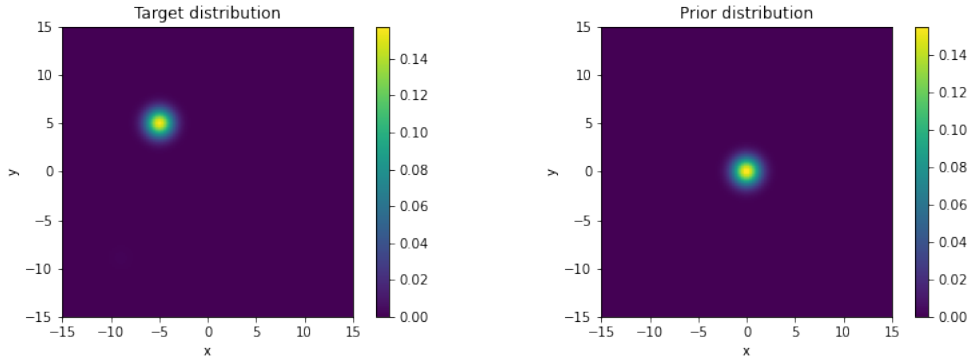



Figure 14: Plot of the loss function values during the training test on a 2-dimensional Gaussian mixture.

### 4.2.1 Comparison with the original code performances

It is interesting to have a closer look on how the version of the code using a unique NF fails in the sampling of highly unbalanced probability distributions.

In particular, that model uses a unique flow and, consequently, a prior distribution which reflects this configuration: in Figure 15b we can see the one chosen in this test is a simple unimodal Gaussian centered in $(x, y) = (0, 0)$, with unitary covariance. The target distribution is the same chosen for the test on the mixture of flows, and it is represented in Fig. 15a.




(a) (b)

Figure 15: (a): Target distribution for the training test on a 2-dimensional Gaussian mixture using a unique NF. (b): Initial configuration of the push-forward distribution for the training test on a 2-dimensional Gaussian mixture using a unique NF.

By looking at the evolution of the push-forward distribution in Fig. 16 and at the comparison between its final configuration and the target distribution in Fig. 17, we are not able to tell whether the weak mode has been properly sampled.

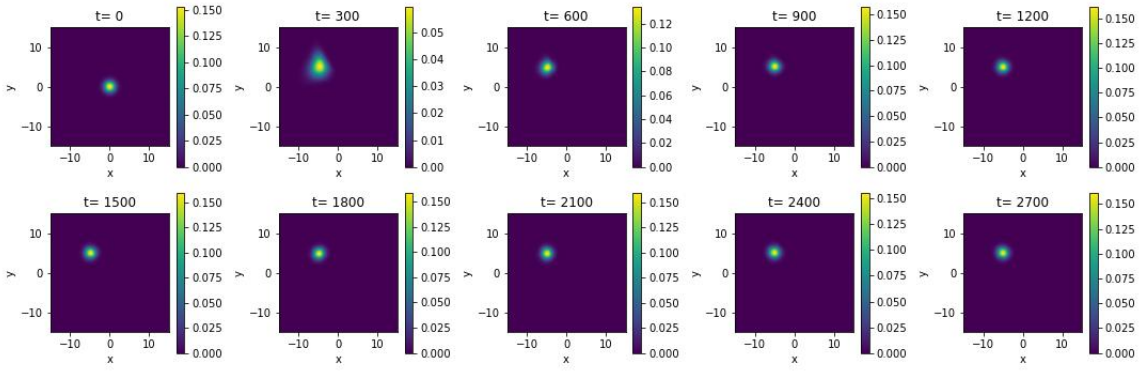



Figure 16: Evolution of the push-forward distribution during the training test on a 2-dimensional Gaussian mixture using a unique NF.

Contrarily to what happens in the mixture of flows algorithm (described in Section 3), in this case we do not have explicit access to the weights of the modes in the final configuration, because these are not learned as parameters of the network. Hence, in order to detect whether

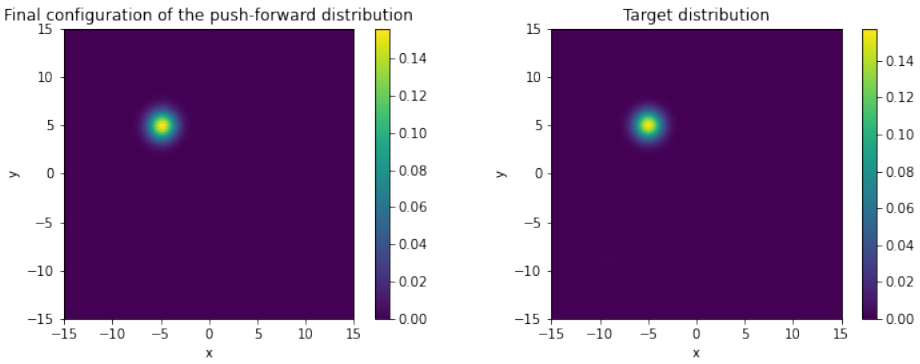



Figure 17: Comparison between the final configuration of the push-forward distribution and the target for the training test on a 2-dimensional Gaussian mixture using a unique NF.

the statistical weights of the target are correctly reproduced by the trained model, we can look at the spatial distribution of a sufficient number $n$ of samples drawn from the final configuration of the push-forward distribution. This is shown in Figure 18, where $n = 10^5$: among the samples, none of them is located close enough to the mean of the weak mode to be associated with it.

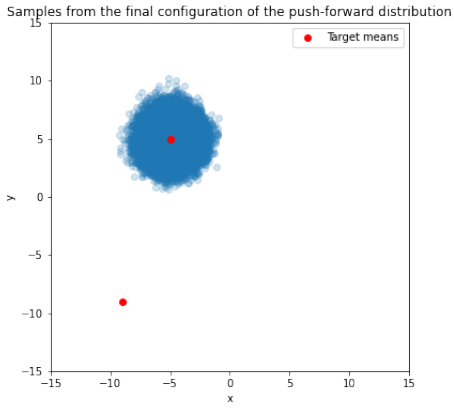



Figure 18: Scatter plot of the samples drawn from the final configuration of the push-forward distribution for the training test on a 2-dimensional Gaussian mixture using a unique NF. Number of samples: 100000.

From this evaluation we can safely confirm that, as anticipated in the description of the motivation of our work, the original adaptive MC is not able to keep track of weak modes, since the whole flow ends up concentrating uniquely in the neighbourhood of the predominant one.

### 4.3 1-dimensional stochastic Allen-Cahn model in presence of an external field

The last model considered for testing our network is the *stochastic Allen-Cahn model*, used in condensed matter physics for the study of phase transitions. According to this model, the time evolution of the random field $\phi : x \in [0, 1] \to \mathbb{R}$ is determined by the following stochastic partial differential equation:

$$\partial_t \phi = a\partial_x^2 \phi + \frac{1}{a}(\phi - \phi^3) + \sqrt{\frac{2}{\beta}}\xi(t, x), \tag{22}$$

where $a > 0$ is a parameter, $x$ the spatial variable and $\xi(x, t)$ is a white noise.

In 1 dimension, the associated Hamiltonian reads as:

$$U_*[\phi] = \beta \int_0^1 \left[\frac{a}{2}(\partial_x \phi)^2 + \frac{1}{4a}(1 - \phi^2(x))^2\right] dx, \tag{23}$$

where the first term is a *coupling* term which disfavours spatial variations of $\phi$ and, at sufficiently low temperature, constrains it to align in positive or negative direction in its entire spatial extension . Analytically, this means the Hamiltonian shows two minima in correspondence of the values $\phi_+$ and $\phi_-$, separated by a free energy barrier. It is this barrier that makes the traditional local MCMC methods not suited for the sampling of this model.

The adaptive MCMC algorithm in [Gabrié et al., 2022] is able to sample efficiently from the probability distribution associated to Eq. (23), by allowing mixing across the energy barrier.

In this configuration of the model, the two modes are characterized by the same statistical weight and it is easy for the normalizing flow to spread its probability mass accordingly to the target distribution. The situation changes when an external field term is added:

$$U_*[\phi, b] = \beta \int_0^1 \left[\frac{a}{2}(\partial_x \phi)^2 + \frac{1}{4a}(1 - \phi^2(x))^2 + b\phi(x)\right] dx. \tag{24}$$

and the symmetry $\phi \to -\phi$ present in Eq. (23) gets broken. Depending on the strength of the field, in fact, weak modes could appear, and the single push-forward, initialized in $\phi = (0, 0, ..., 0, 0)$, could initially spread across both modes, but would end up collapsing onto the dominant one, similarly to what happened for the 2-dimensional Gaussian mixture.

Then, a mixture of flows could help in detecting the correct weight ratio between the basins.

Notice that, even if the real spatial dimension of the model is 1, in order to practically deal with it we need to discretize the space (and so the field $\phi(x) \to \phi_i$) in a grid of N sites: hence, the map we need to train $T : \mathbb{R}^N \to \mathbb{R}^N$ is high-dimensional. This reflects also in the discretization of Eq. (22) into the corresponding Langevin equation, used as the dynamics for proposing the local moves.

Before showing the performances in this particular instance of the model we devised, it is useful give more details about the specific base distribution we use.

**Informed base distribution** As anticipated in Subsection 3.2.3, for the stochastic Allen-Cahn model we make use of an *informed*, or *'coupled'* base distribution [Gabrié et al., 2022], which encodes the information about the spatial coupling of the field $\phi(x)$. This is needed because a Gaussian field with uncoupled spins, corresponding to a system with Hamiltonian of the form:

$$U_B[\phi] = \beta \int_0^1 \frac{1}{2}\phi(x)^2 dx, \tag{25}$$

is not sufficient to learn the map efficiently. In particular, this alternative base measure is a Gaussian random field with a local coupling, called *Ornstein-Uhlenbeck bridge*, and its Hamiltonian is the following:

$$U_{B,\text{informed}} = \beta \int_0^1 \left[\frac{a}{2}(\partial_x \phi(x))^2 + \frac{1}{2a}\phi^2(x)\right] dx. \tag{26}$$

The discretized version reads as:

$$U_{B,\text{informed}} = \beta \sum_{i=1}^{N} \left[\frac{a}{2}(\phi_{i+1} - \phi_i)^2 + \frac{1}{2a}\phi_i^2\right], \tag{27}$$

and it is implemented as a multivariate Gaussian with:

- *mean*: corresponding to the mean of the interested mode. This is found by minimizing the target potential through the GD algorithm, starting from the "ideal" configuration $\phi_{+,-} = (0, \pm 1, \pm 1, ..., \pm 1, 0)$ (Dirichlet boundary condition $\phi(x = 0) = \phi(x = 1) = 0$ are implemented), of length $N$;
- *precision matrix*:

$$\Sigma^{-1}_{\text{informed}} = \beta \begin{bmatrix} \gamma - c & -c & 0 & \cdots & \cdots & \cdots & 0 \\ -c & \gamma - c & -c & \ddots & & & \vdots \\ 0 & -c & \gamma - c & -c & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -c & \gamma - c & -c \\ 0 & \cdots & \cdots & \cdots & 0 & -c & \gamma - c \end{bmatrix} \tag{28}$$

  with $c = aN$, $\gamma = 3ac + \frac{1}{c}$, and size $N$x$N$.

**Training on a *weak mode* distribution** For the training test on the Allen-Cahn model we chose the following parameters: *coupling coefficient* $a = 0.1$, *local field* $b = 0.025$, *grid size* $N = 10$, *inverse temperature* $\beta = 10$.

Concerning the initialization of the network:

- the means with which the prior distributions are initialized are computed by running 10000 steps of the GD algorithm, with a $dt = 0.1$. The final configuration is shown in Figure 19;

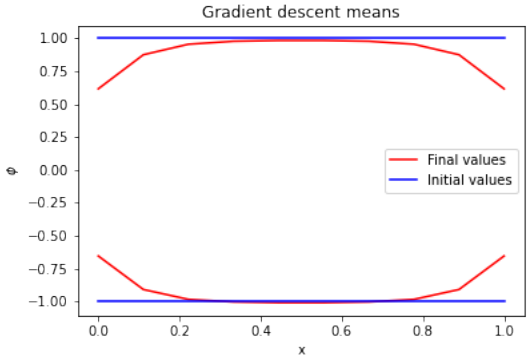



Figure 19: Values of the modes $\phi_+$,$\phi_-$ of the stochastic Allen-Cahn model in 1 dimension, obtained by minimizing the target potential through the GD algorithm. Parameters of the target: $b = 0.025$, $a = 0.1$, $\beta = 10$. Grid size: $N = 10$.

- the initial weights of the priors associated to the modes $\phi_+, \phi_-$ are $w_+ = 0.035, w_- = 0.965$. These values are chosen by performing preliminary tests[13] on configurations with smaller $b$: their results, shown in Fig. 20, confirm the expected decreasing trend of $w_+$ at increasing $b$, so we decide to ease the learning for $b = 0.025$ by starting from the values obtained at $b = 0.020$. In Figure 21 we can see 10000 samples from the global prior distribution obtained by the weighted sum of the single flows' ones.

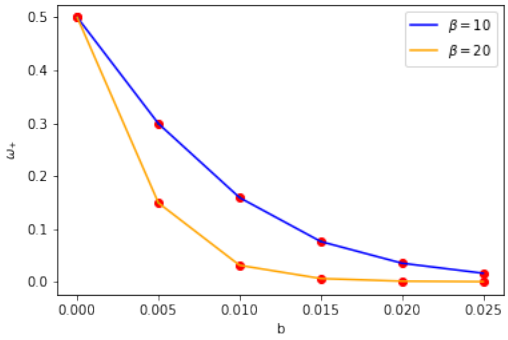



Figure 20: Values of the weights to which the network converge in the test on the stochastic Allen-Cahn model in 1-$d$ for different $b$, $\beta$, $N = 10$, $a = 0.1$.

After training the mixture for $n = 15000$ steps, with $bs = 100$ and $\eta = 0.001$, we converge to the final values of the weights $\omega_+ = 0.016$, $\omega_- = 0.984$, computed as the mean of the values

[13]These preliminary tests are done on target distributions associated to increasing values of $b \in \{0.005, 0.010, 0.015, 0.020\}$, $a = 0.1$, $N = 10$ and $\beta \in \{10, 20\}$. The training parameters are: $n_{\text{steps}} = 3000$, $bs$=100, $\eta = 0.001$. The prior distributions are *'coupled'* and are initialized with means computed by the GD algorithm, run for 10000 steps with $dt = 0.1$. For each test, the initial values of the push-forward weights are chosen equal to the values to which the simulation for the immediately smaller $b$ has converged. In particular, the final weights considered here do not coincide exactly with the value these parameters assume at the last iteration of training, but they are computed as the mean of the values taken in the last 1000 iterations; the associated standard deviations are negligible in all the simulations.

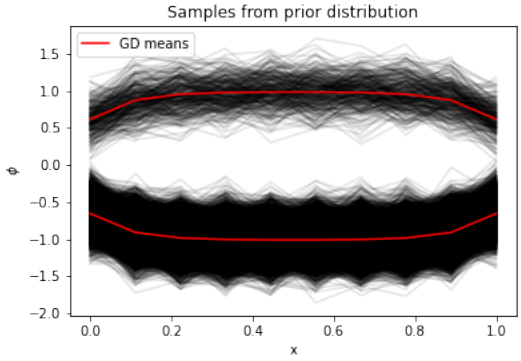



Figure 21: Samples from the prior distribution for the stochastic Allen-Cahn model in 1 dimension. Parameters of the target: $b = 0.025$, $a = 0.1$, $\beta = 10$. In red: means of the modes computed by GD. Initial weights of the modes: $\omega_+ = 0.035, \omega_- = 0.965$. Number of samples: 10000. Grid size: $N = 10$.

of these parameters along the last 1000 iterations; the associated standard deviations are of the order of 0.00001, so the error can be considered negligible. In Fig. 22 we observe 10000 samples from the related final push-forward distribution.

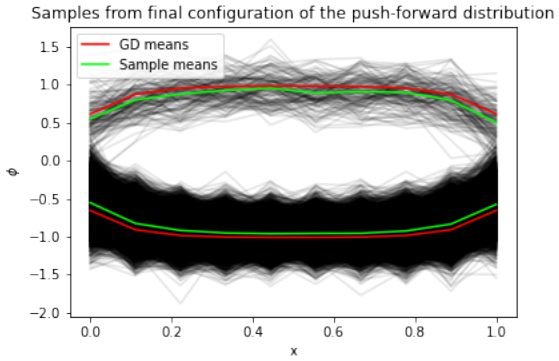



Figure 22: Samples from the final configuration of the push-forward distribution for the stochastic Allen-Cahn model in 1 dimension. Parameters of the target: $b = 0.025$, $a = 0.1$, $\beta = 10$. In red: means of the modes computed by GD. In green: means of the modes computed empirically from the samples drawn from the final push-forward. Number of samples: 10000. Grid size: $N = 10$.

From Fig. 23 we see that the loss function along the simulation is very noisy. This behaviour can be attributed to two factors: first, the batch size, whose value affects significantly the cost of the computation, is chosen to be pretty small in order to have faster simulations, thus determining heavy fluctuations of the loss; second, the push-forward distribution is initialized in an excellent way (thanks to the GD computation of the means, to the informed covariance matrices and to the weights obtained from the preliminary tests), hence, being it very close to the target distribution already in its initial configuration, it is possible that

the difference the GD needs to cover to reach the loss function minimum is smaller than the amplitude of the oscillations due to the batch size. This behaviour makes it difficult to see whether the loss is actually decreasing: even the *moving average*[14] is not able to capture this expected trend.

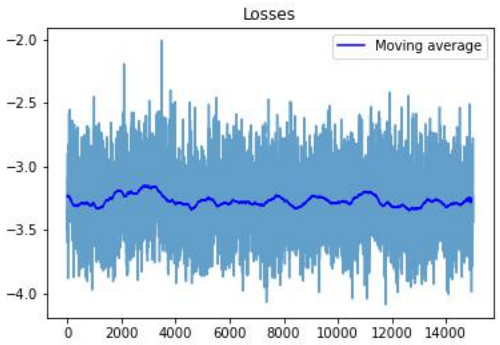



Figure 23: Plot of the loss function values during the training test on the stochastic Allen-Cahn model in 1 dimension. The moving average is computed using a *window size*=700.

Anyway, the improvement and convergence of the weights is confirmed by the plot of their values along training shown in Fig. 24: precisely, in order to appreciate the difference between the initial and the final value, which is approximately 0.02, this plot shows only the value of $\omega_+$; the complementary behaviour of $\omega_-$ can be easily deduced thanks to the fact that their sum is constrained to 1.

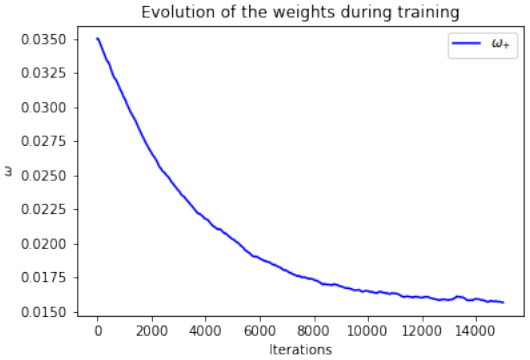



Figure 24: Evolution of the weight $\omega_+$ of the push-forward distribution for the stochastic Allen-Cahn model in 1 dimension during training.

In all the tests (including the ones reported in Fig. 20) we are confident that the learning has converged not only because the parameters stop adapting and start oscillating with negligible variance around a specific value, but also because the free energy differences computed

[14]The *moving average* $m_i$ at iteration $i$ of a time series $\{f_i\}, i \in \{i = 1, ..., N\}$ is computed as the mean of the values of $\{f_i\}$ in the following $ws$ iterations: $m_i = \frac{1}{ws} \sum_{i=1}^{i+ws} f_i$. $ws$ stands for *window size*, and in the calculation for the loss function it is chosen as $ws = 700$.

using the associated push-forwards (see the following Subsection) are coherent with the results found in [Gabrié et al., 2022].

**Free energy difference** The last analysis performed on the stochastic Allen-Cahn model involved the computation of the free energy difference $-\Delta F = F_- - F_+$ between the modes $\phi_-, \phi_+$. Explicitly, this difference is estimated as:

$$-\Delta F = \log \frac{\int_{\mathbb{R}^d} \mathbb{1}_+(x) e^{-\beta U_*(x)} dx}{\int_{\mathbb{R}^d} \mathbb{1}_-(x) e^{-\beta U_*(x)} dx} = \log \mathbb{E}_*[\mathbb{1}_+(x)] - \log \mathbb{E}_*[\mathbb{1}_-(x)], \tag{29}$$

where $\mathbb{1}_+(x)$ is the indicator function of the set $\{\phi : \int_0^1 \phi(x)dx > 0\}$, $\mathbb{1}_-(x)$ is the indicator function of the set $\{\phi : \int_0^1 \phi(x)dx < 0\}$, and $\mathbb{E}_*$ indicates the expectation over the target distribution $\rho_*[\phi(x)]$.

Since we have access to the exact expression of the target distribution, but we are not able to directly sample from it, this computation has been done by means of the *importance sampling*, which is a technique that allows to estimate expectations on a target measure (here $\rho_*[\phi(x)] = \frac{e^{-\beta U_*[\phi(x)]}}{Z_*}$) through independent samples from a proposal measure (here $\hat{\rho}_{\bar{\theta},\omega}[\phi(x)]$ in its configuration at the last training step, which we are able to sample)[Tokdar and Kass, 2010].

Precisely, we take $n$ samples $\phi_i(x_1, .., x_N)$ from $\hat{\rho}_{\bar{\theta},\omega}[\phi(x)]$ and we estimate the expectations over $\rho_*[\phi(x)]$ as a weighted average over these random draws:

$$\mathbb{E}_*[\mathbb{1}_+(x)] \approx \frac{Z_{IS,+}}{Z_*} = \frac{1}{Z_*} \sum_{i=1}^{n} \mathbb{1}_+(\phi_i)\hat{w}(\phi_i), \tag{30}$$

where $\hat{w}(\phi_i) = \frac{e^{-}\beta U_*(\phi_i)}{\hat{\rho}(\phi_i)}$ are called the *unnormalized weights*, and the unknown $Z_*$ is canceled out in the free energy estimator:

$$-\Delta F \approx \log(\frac{1}{Z_*} \sum_{i=1}^{N} \mathbb{1}_+(\phi_i)\hat{w}(\phi_i)) - \log(\frac{1}{Z_*} \sum_{i=1}^{n} \mathbb{1}_-(\phi_i)\hat{w}(\phi_i)) = \log \frac{\sum_{i=1}^{N} \mathbb{1}_+(\phi_i)\hat{w}(\phi_i)}{\sum_{i=1}^{n} \mathbb{1}_-(\phi_i)\hat{w}(\phi_i)}. \tag{31}$$

The estimator of the variance associated to the computation in Eq. (30) is given by:

$$\mathrm{Var}\left(\frac{Z_{IS,+}}{Z_*}\right) \approx \frac{\hat{\mathrm{Var}}(\mathbb{1}_+(\phi_i))}{n_{\mathrm{eff}}}, \tag{32}$$

where $n_{\mathrm{eff}}$ is the *effective sample size*:

$$n_{\mathrm{eff}} = \frac{n}{\tau} = \frac{(\sum_{i=1}^{n} \hat{w}(\phi_i))^2}{\sum_{i=1}^{n} \hat{w}^2(\phi_i)}, \tag{33}$$

with $\tau$ corresponding to the *autocorrelation time* of the chains [Gabrié et al., 2022].

In Figure 25 we can see the results we obtain following this method for $-\Delta F(b)$ at different $\beta$. For each combination of the parameters, we used the associated final configuration of the push-forward distribution obtained in the preliminary tests mentioned in the previous paragraph, whose related converged weights are represented in Fig. 20. The variance estimator gives negligible values for all the calculations, so the associated error bars are not visible.

As anticipated, these results are in accordance with the ones found using the adaptive MCMC augmented with a unique NF.

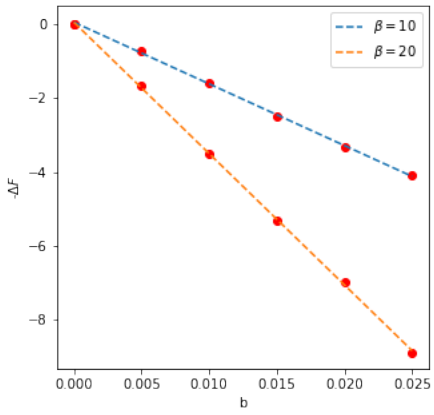



Figure 25: Free energy difference $-\Delta F$ computed for different values of the external field $b$ and inverse temperature $\beta$.

### 4.4 Discussion of the results

The first two tests involved mixtures of Gaussians in 1 and 2 dimensions: this family of functions might be one of the most easy and common examples when dealing with probability distributions, but it is precisely for their simplicity and ubiquity that it was fundamental to verify that our algorithm is suited to their sampling. The choice for the third test fell on the stochastic Allen-Cahn model: it is with this model that, further than the adaptation to multi-modality, we checked the behaviour of our algorithm in a high-dimensional framework.

The main finding we can gather from these experiments is that, as expected, assigning a single normalizing flow to each of the known modes of the target distribution ensures sampling takes place accurately regardless of the relative statistical weights of the modes, differently from what happens in the original version of the algorithm. The key feature for this to be true is the normalization of the weights in the loss function.

Moreover, we can state that the combination of multiple normalizing flows does not affect the power of the single flow: the network is able simultaneously to learn the new introduced parameters, *i.e.* the weights, and to adapt to the shape of the basins, identifying the correct means and covariances as well as the original code does.

A big contribution to the efficiency of the algorithm is given by a proper initialization of the prior distribution and, even before that, by some form of prior knowledge concerning the modes of the target: with this algorithm, indeed, the issue of exploring the unknown landscape of a probability distribution is not addressed, and we limit our analysis to the cases where the location of the basins has already been identified.

Together with the initialization, something that strongly impacts the learning process, as in any machine learning algorithm, is the tuning of the hyperparameters[15]. In our case,

[15]In machine learning, the hyperparameters are those parameters which are external to the model and are not derived via training. They are typically specified by the user and chosen using heuristics.

for example, the need of adapting the weights to very little values makes the tuning of the learning rate fundamental: the closer we are to the target and the smaller is expected to be the target weak mode weight, the smaller the learning rate needs to be for having a smooth convergence. Also the batch size plays, in principle, an important role, but even if we set it to a particularly small value for relieving the computational burden of the simulation, the network keeps its ability to find the loss function minimum.

Bear in mind that there is still a limit for the precision on the highly unbalanced weights this algorithm is able to detect: beyond a certain value, it is the intrinsic representation limit of the computer that prevents the network from adapting properly to the target. This is the case, for example, in the stochastic Allen-Cahn model in presence of relatively big external fields: here the lighter modes could reach a weight of the order of $10^{-7}$ or less, which, being hardly distinguishable from 0, would lead the whole mixture of flows to collapse on the predominant basin. When dealing with this kind of configurations, an alternative approach would be to use more suited techniques, like *rare events sampling*.

Anyway, despite from these first tests the algorithm seems promising, there is still need to perform experiments on more complex distributions, in higher dimension, where the number of modes is higher than two.

# 5 Conclusion

The project described in this Master's thesis had the goal of contributing to the solution of a crucial issue in scientific computing: the manipulation of complex probability distributions.

An algorithm meeting the challenge of this task must satisfy a series of conditions: it must take into account the presence of different modes, it must capture their relative weights, whatever their values are, and it must be computationally efficient, even in high-dimensional frameworks.

The *Mixture of normalizing flows* algorithm fulfills all the above requirements.

Starting from the innovative strategy proposed in [Gabrié et al., 2022], *i.e.* the Adaptive MC augmented with a unique normalizing flow, the replacement of the single flow with a weighted mixture seemed the best idea to address the problem of sampling significantly unbalanced modes. Looking through the existing literature concerning the topic and the instruments provided by PyTorch to deal with generative models, I chose to combine the multiple NFs in a new, outer list, where each module is implemented as the unique flow in [Gabrié et al., 2022]. A further possibility could have been to realize the mixture at a different level of the algorithm: not externally, as I did, but inside the unique starting NF, *e.g.* by replacing its prior distribution with a weighted combination of priors.

Translating the chosen design in an actual, functioning code has been for me the most stimulating challenge of the project. Indeed, the main effort has been to implement the algorithm consistently with the existing code and the constraints of its structure, to which each of my new modifications had to match in order for the resulting algorithm to function smoothly.

Hence, the writing of the code has not been straightforward, but in fact frequently interspersed with a series of tests on simple examples, which allowed me to locate the precise source of any possible error in the long, nested architecture of the network, together with analytical computations. In particular, to the aim of this Thesis work, the most significant has been the one related to the Loss function analysis (reported in Section 4): only the analytical treatment of the backpropagation algorithm revealed the proper structure of this function and the relevance of the role of its normalization.

The decisive proof that the algorithm was properly functioning has been the test on the stochastic Allen-Cahn model: due to its high-dimensionality, it took more creativity to analyze the results, which are not intuitively interpretable, but it was this same feature to make the strengths of the algorithm emerge.

We can thus conclude that the *Mixture of normalizing flows* algorithm is able to sample high-dimensional, multi-modal probability distributions even in presence of basins with highly unbalanced statistical weights.

Upon further testing in more elaborate settings, this method could represent an alternative to the existing techniques devoted to the sampling of complex probability landscapes. One of the most common is *tempering*: its major limitation derives from its computational cost, which is partly solved in the adaptive MC algorithms, but, despite this, it still represents an adequate approach to discover the geometry of a distribution when no prior knowledge is given about it.

Hence, a perspective for future research could be to further extend the adaptive MC augmented with mixtures of normalizing flows, incorporating a tool for the detection of unknown modes.

# 6 References


[Albergo et al., 2019] Albergo, M., Kanwar, G., and Shanahan, P. (2019). Flow-based generative models for markov chain monte carlo in lattice field theory. *Physical Review D*, 100(3).

[Dinh et al., 2016] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp.

[Gabrié et al., 2022] Gabrié, M., Rotskoff, G. M., and Vanden-Eijnden, E. (2022). Adaptive monte carlo augmented with normalizing flows. *Proceedings of the National Academy of Sciences*, 119(10):e2109420119.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Hukushima and Nemoto, 1996] Hukushima, K. and Nemoto, K. (1996). Exchange monte carlo method and application to spin glass simulations. *Journal of the Physical Society of Japan*, 65(6):1604–1608.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

[Papamakarios et al., 2019] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing flows for probabilistic modeling and inference.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pompe et al., 2018] Pompe, E., Holmes, C., and Łatuszyński, K. (2018). A framework for adaptive mcmc targeting multimodal distributions.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

[Ruthotto and Haber, 2021] Ruthotto, L. and Haber, E. (2021). An introduction to deep generative modeling.

[Spagnol et al., 2016] Spagnol, S., Galesso, S., and Avanzini, F. (2016). Stima di feature spettrali di hrtf mediante modelli antropometrici non lineari per la resa di audio 3d. Figure 5. Representation of a Multi-Layer Perceptron.

[Tierney, 1998] Tierney, L. (1998). A note on metropolis-hastings kernels for general state spaces. *The Annals of Applied Probability*, 8(1):1–9.

[Tokdar and Kass, 2010] Tokdar, S. and Kass, R. (2010). Importance sampling: A review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2:54 – 60.

[Van Rossum and Drake, 2009] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[Zhang et al., 2022] Zhang, L., Naesseth, C. A., and Blei, D. M. (2022). Transport score climbing: Variational inference using forward kl and adaptive neural transport.

# Appendices

## A Multi-Layer Perceptron

The MLP is the elementary block of deep neural networks. It consists of a system of interconnected *perceptrons*, also referred to as *neurons* or *nodes*, which maps an input vector $x \in \mathbb{R}^n$ into an output vector $y \in \mathbb{R}^m$ through a non-linear transformation.

The neurons are organized into *layers*. The typical structure of the MLP, for which we can see an example in Fig. 26, presents:

- a set of source neurons forming the *input layer*, which has no computational role but just passes the input vector to the network. Its dimension corresponds to the dimension $n$ of the input $x$;
- a set of computation nodes forming one or more *hidden layers*, each of them with arbitrary dimension $d_i$;
- an *output layer*, whose dimension $m$ defines the dimension of the output vector $y$.

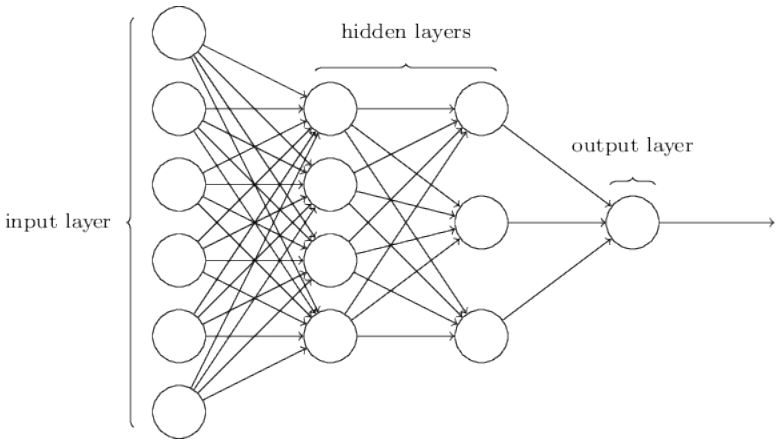



Figure 26: Example of the structure of a Multi-Layer Perceptron. The dimensions of the input and the output layer are: $n = 6$, $m = 1$. The number of hidden layers is 2, and they have respectively dimension $d_1 = 4$, $d_2 = 3$. Source: [Spagnol et al., 2016].

A MLP is fully-connected, meaning that each of the neurons constituting a layer is linked to all the neurons in the adjacent layers. These links are associated to parameters called *weights* $\theta_{i,j} \in \mathbb{R}$ ($i$ is the index of the neuron where the link starts, $j$ is the index of the target neuron in the following layer).

In order to understand how the MLP non-linear transformation takes place, it is useful to look at the principle of operation of the single perceptron. In particular, the $j$-th perceptron computes a single output $y_j$ from multiple real-valued inputs $\{x_i\}_{i=1,\dots,N}$ by linearly combining them according to the input weights $\{\theta_{i,j}\}_{i=1,\dots,N}$, by adding a bias $b_j$ to this combination,

and, in the end, by pushing the resulting sum into a *non-linear activation function* $f(x)$. Explicitly, this reads as:

$$y_j = f\left(\sum_{i=1}^{N} \theta_{i,j} x_i + b_j\right). \tag{34}$$

There are many choices for the activation function: in the original paper written by [Rosenblatt, 1958], creator of the MLP, it is a simple Heaviside step function, but common alternatives are the sigmoid function or the hyperbolic tangent. In the case of the MLPs building the normalizing flows in [Gabrié et al., 2022], the chosen activation function is a *rectified linear unit* (ReLU) $f(x) = \max[0, x]$.

The single neurons are used in the MLP to propagate the input signal $x$ through the network, layer by layer, from the input to the output. Specifically, the output of each node $j$ belonging to a hidden layer or to the output one is computed following Eq. (34), where $x_i$ are the outputs of all the $N$ nodes constituting the immediately previous layer. This implies a direction of the information processing, hence we can refer to the MLP as a *feed-forward* neural network.

The parameters $\{\theta\}$, that are arbitrarily initialized, can be optimized with the purpose of resolving accurately the map the network aims to model. This is what enables the MLP to learn, by means of a process called *training*. Training consists of minimizing, typically through *Gradient descent algorithm*, a *cost*, or *loss function* $L(\bar{\theta})$ which quantifies the difference between the desired and the actual output.

A widely used algorithm for the training of feed-forward neural networks is the *back-propagation algorithm*. This acts through two main steps:

- *forward pass*: the predicted outputs $y_{\text{pred}} \in \mathbb{R}^m$ corresponding to given inputs $x_{\text{train}} \in \mathbb{R}^n$ are computed, layer by layer, according to (34);
- *backward pass*: the partial derivatives $\partial_{\theta_k} L(\bar{\theta})$ of the loss function with respect to all parameters $\theta_k$ (including the biases) are computed, according to the *chain rule*. Then, the parameters are adjusted according to the following:

$$\theta_k \leftarrow \theta_k - \eta \partial_{\theta_k} L(\bar{\theta}), \tag{35}$$

  where $\eta$ is a hyperparameter called *learning rate*, which measures the steps taken to approach the minimum of $L(\bar{\theta})$.

The success of the MLP architecture in deep neural networks lies in its representation power: as stated in [Hornik et al., 1989], *"standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense"*. The key for this ability is the non-linearity at the level of the single neurons: if this was not present, then the combination of multiple neurons would be reducible to a unique, collapsed, linear transformation, and only linear functions would be reproducible.