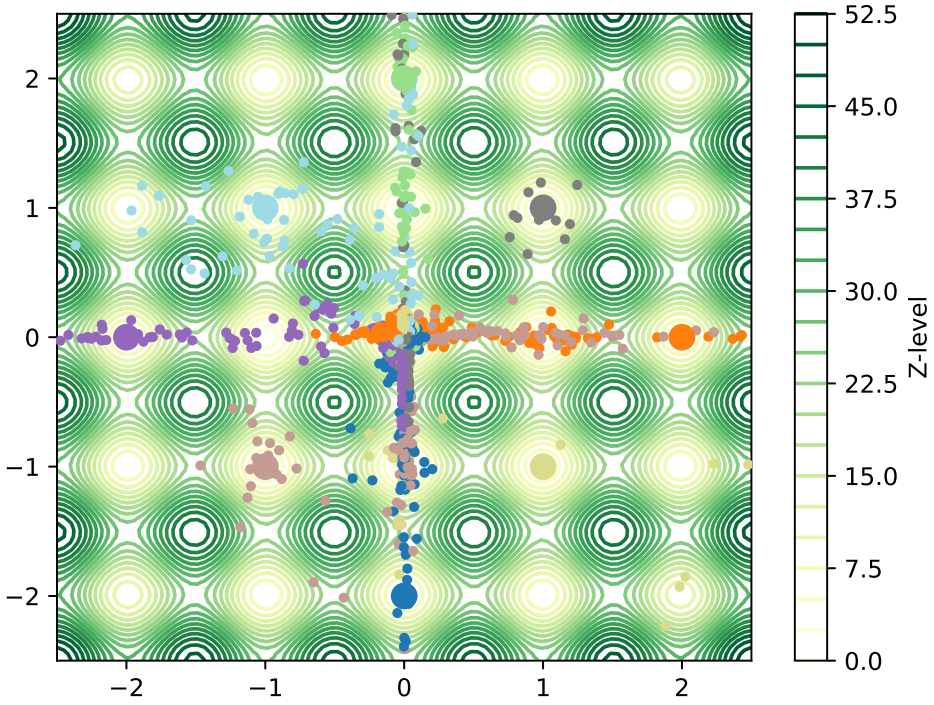


Diffusion based interacting particle system for optimization

Master's thesis in Complex adaptive systems

Gabriel Vevang

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Diffusion based interacting particle system for optimization

Gabriel Vevang

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2024

Diffusion based interacting particle system for optimization
Gabriel Vevang

Supervisor: Akash Sharma, Department of Mathematical Sciences
Examiner: Axel Ringh, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Contour plot of 8 particles finding the global minimum on the Rastrigin function (2.14) using the ensemble Langevin dynamics (3.6). The initial positions of the particles are indicated by the larger sized particles.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Diffusion based interacting particle system for optimization
Gabriel Vevang
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The ability to minimize a function is crucial across various fields, from training neural networks by minimizing loss functions to reducing costs in production lines. This minimization is typically achieved using optimization algorithms where the choice of algorithm impacts both the overall result and the computational cost. In this thesis, we propose new optimization algorithms based upon particle interactions driven by stochastic differential equations (SDEs). Specifically, we aim to adapt first and second-order ensemble Langevin sampling methods to use as optimization algorithms. Additionally, we explore a variant of consensus-based optimization (CBO) that incorporates repulsive forces. Our results demonstrate the effectiveness of the sampling methods with annealing as optimizers and highlight the benefits of repulsion for CBO. Furthermore, we test ensemble Langevin dynamics as an optimizer for training neural networks. We approximate gradient using Kalman approximation that allows for training without the need for back-propagation. The results indicate performance similar to stochastic gradient descent (SGD).

Keywords: Optimization, Interacting particle system, Consensus-based optimization, Ensemble Langevin dynamics.

Acknowledgements

I would be lying if I said that I completed this thesis all on my own. This would never be possible without all the people supporting me through this journey.

First, I would like to thank all of my friends for keeping me sane by providing endless hours of entertainment during this time.

I would also want to thank my family for all the love and support even on the days when I was grumpy and tired.

I am grateful to my examiner Axel Ringh for guiding me through the complicated process of writing a master's thesis.

Finally, I want to give an especial thank you to my supervisor, Akash Sharma for your guidance during this process, for always being patient and available when issues arose and for never doubting me. Thank you.

Gabriel Vevang, Åsa, June 2024

Contents

1

Introduction

Due to wide digitalization across society, availability of data has drastically increased. However, drawing conclusions from the data in itself is practically impossible, and one instead has to resort to data-based models. One way of doing this is via a nonlinear regression model such as neural networks or kernel regression. To utilize these models, one has to train them, which in many cases means solving a large-scale optimization problem. However, the optimization algorithm used to solve the problem affects both the result of the model and also the required computational power. One type of optimization method is particle/agent-based interaction models, for example particle swarm optimization [1]. These models work by distributing N number of particles across the parameter space, and via some communication or interaction, the particles can move closer or further away from each other with the goal of finding the global minimum.

For this project, the optimization methods that will be explored are going to be based on stochastic differential equations (SDEs), making it more approachable to perform theoretical and numerical analysis. The different methods are: ensemble Langevin dynamics based optimization, ensemble kinetic Langevin dynamics based optimization, and consensus based optimization (CBO). These methods have been proposed in prior papers: diffusion based CBO for optimization [2, 3] (and its extension to the jump-diffusion setting [4]) and ensemble Langevin dynamics for solving inverse problems [5, 6, 7].

In this thesis, we work with two classes of optimization methods. In the first setting, we turn ensemble Langevin dynamics into optimization algorithms. In the second class, we introduce a new consensus based optimization with repulsive forces.

The purpose of the thesis is the computational study of these optimization algorithms by testing them on some benchmark functions and evaluating their performance based on different parameters. We also test the methods on the task of training neural networks. In the thesis, we also implement less explored variants of these methods.

2

Background

When it comes to regression models, the goal is to find a relationship between independent variables and the target variable. This could be useful for many different situations, such as finding which type of advertisements should be shown for certain users, or weather forecasts depending on current weather conditions.

The process of finding these relationships is done by modeling a set of preexisting data points and creating a function or model that accurately represents those data points. This is done by trying to minimize the difference between the model prediction and the set of known data points. The difference between the prediction and the target can be described by a function, and minimizing that functions is an optimization problem.

2.1 Optimization

If the goal is to minimize the function, one might simply try to evaluate the function for each parameter in the parameter space. While this is doable for simple functions with small parameter spaces (for example in discrete settings). The issue arises when more complex functions in higher dimensions need to be minimized. The computational time for evaluating each parameter for those functions is infeasible. Instead, one resort to optimization algorithms.

2.1.1 Gradient descent

One of the simplest forms of optimization algorithms is gradient descent. In simple words, gradient descent aims to find the minimum of a convex differentiable function by moving in the opposite direction of the gradient. We start by writing a differential equation describing the dynamics $X(t)$ taking values in $\mathbb{R}^d$ for $t \geq 0$:

$$\frac{d}{dt}X(t) = -\nabla f(X(t)), \quad X(0) = x, \tag{2.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a differentiable convex function and $X(t)$ is the input variable. To then perform numerical calculations, the continuous-time equation is discretized. One starts by considering a time interval $[0, T]$ that is evenly split into n subintervals, meaning that $0 = t_0 < t_1 < ... < t_n = T$ with a time-step $h = t_k - t_{k-1} = t_n/N$. This means that $t_k = hk$. For convenience one can then index X such that

$X(hk) \approx X_k$ where $X(hk)$ is the true solution from (2.1) and

$$\frac{X_{k+1} - X_k}{h} = -\nabla f(X_k), \quad X_0 = x. \tag{2.2}$$

By rewriting the equation, one then gets the forward gradient descent algorithm [8]

$$X_{k+1} = X_k - \nabla f(X_k)h, \quad X_0 = x. \tag{2.3}$$

When performing calculations, the algorithm begins by randomly selecting a starting point. Then simply move a certain distance in the steepest direction which is the opposite direction of the function's gradient. Repeat until the minimum is reached. Note that the true minimum is rarely achieved. This is due to the dynamics being estimated with a discrete time step. Meaning that one usually settles when inside the $\epsilon-$neighborhood of the minimum, where $\epsilon-$neighborhood of a point x is the set of all points which are less than ϵ distance from x and $\epsilon > 0$ is the error due to the estimation.

One drawback with using gradient descent is that it will travel towards the bottom of the valley it starts in. Meaning that the algorithm will not find the global minimum unless it happens to start inside the valley whose bottom is the global minimum. To address this, one approach is to introduce noise into the system. The effectiveness of the noise depends upon several factors, including its type and magnitude. Additionally, the shape of the function that is to be minimized also impacts the effectiveness of the noise. For example, if we look at the Ackley function [9], it features multiple local minima. As one approaches the global minimum, each local minima descends deeper while the heights of the peaks remains similar. This results in the value of the gradients leading towards the global minimum to be higher than those leading away from it. Since the movement of the particle is influenced by both the gradient and the noise, on average the particle will move towards the global minimum.

Example 2.1. Let us consider a variant of the one-dimensional Rastrigin function,

$$f(x) = 1 + x^2 - \cos(2\pi x). \tag{2.4}$$

Due to the function containing a cosine term, it will result in many local minima. Let us try to find the global minimum using gradient descent with Gaussian noise of different magnitudes. The algorithm is as follows:

$$X_{k+1} = X_k - \nabla f(X_k)h + \beta\mathcal{N}(0, 1), \tag{2.5}$$

where k is the iteration, h is the step-size, β is the parameter defining the magnitude of the noise and $x(0) = x_0 = -2.5$. For the simulation, 1000 iterations are performed with a step-size h of 0.01. By looking at the plot in Figure 2.1 one can see the importance of both noise and its magnitude. Here it is clear that noise is important for the particle not to get stuck before reaching the global minimum. However, deciding upon how much noise should be present can be rather difficult. On one hand, as the magnitude of the noise increases, so does the size of the space the

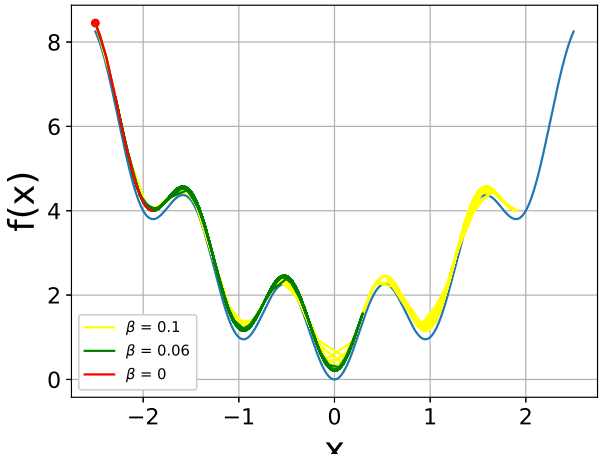


Figure 2.1: Difference in convergence depending on the magnitude of noise while minimizing (2.4) using gradient descent with Gaussian noise (2.5)

particle explores. On the other hand, by introducing too much noise, one might make too large jumps, such that some minima are missed. It is therefore important to be meticulous when deciding upon how noise is introduced into the system, such that a good balance between exploration and exploiting of the algorithm can be achieved.

Another issue with gradient descent is slow convergence. If the shape of the function consists of large areas with small inclination, the gradient will be very small, also known as the vanishing gradient problem in machine learning literature. This means that many computations are needed before reaching the minimum. Intuitively one might simply want to increase the time-step to solve this issue. However, this will not work in all cases, for example, if the shape of the function is similar to a narrow valley, such as in the Rosenbrock function (2.15). In this case, the gradient will be very large when moving down either side of the valley, meaning that the algorithm will jump between either side of the valley instead of moving in the desired direction. Increasing the time-step in this case would add very little to the convergence while increasing the risk of making a step that is too large, which can reverse some of the progress.

Example 2.2. Let us consider the following function:

$$f(x) = 5x_1^2 + 50x_2^2, \quad x = (x_1, x_2) \in \mathbb{R}^2. \tag{2.6}$$

The minimum of the function is located at $(0, 0)$ and the gradient of the function is $\nabla f = [10x_1, 100x_2]$. By using gradient descent to find the minimum for different time-steps, it is clear that a small time-step is necessary for the second dimension to not suffer from exploding gradients, see Figure 2.2. However, having a small time-step forces us to perform many computations for the first dimension to find the minimum. A better approach is to use second order optimization.

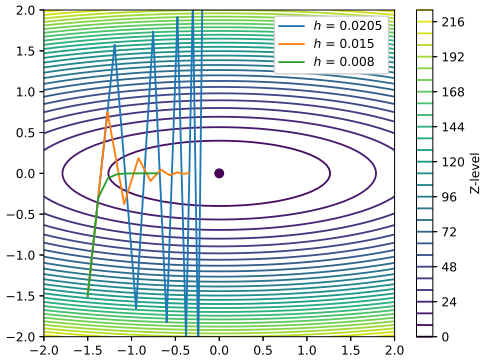


Figure 2.2: Ten iterations of gradient descent (2.3) on example function (2.6) for different time-steps h.

2.1.2 Heavy ball method

The heavy ball method is a second order optimization algorithm based on momentum [10]. It takes into account values from both the current and previous time-steps. By doing so, it not only allows the algorithm to take larger leaps even if the incline of the slope is small, it also reduces the issue of oscillation, since drastic changes in direction is reduced with momentum. The algorithm is described as:

$$V_k = -\nabla f(X_k) + \beta V_{k-1}, \tag{2.7}$$

$$X_{k+1} = X_k + hV_k, \tag{2.8}$$

in which V_k is the momentum at the $k-$th iteration and (X_0, V_0) is the initial guess. Note that this method is used to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$. This means that $X(t), V(t) \in \mathbb{R}^d$. The main difference here compared to gradient descent is the introduction of the auxiliary variable $V(t)$.

Example 2.3. Let us again consider the function in (2.6). Now by utilizing momentum, the performance of the algorithm is much better, see Figure 2.3.

Here, it is clear that a larger step-size can be used without the issue of exploding gradients. One can also see that, even with smaller time-steps the algorithm is making faster progress towards the global minimum, due to it having the benefit of momentum.

We will revisit these concepts in later chapters when discussing and comparing first and second order ensemble Langevin dynamics.

2.2 Stochastic differential equations

Having seen the benefits of noise, gradients, and momentum one might wonder if there is a way of utilizing all these properties simultaneously. That is where stochastic differential equations (SDEs) come into play. SDEs are differential equations with a stochastic process. For a more detailed introduction to SDEs, see, e.g., [11]. In

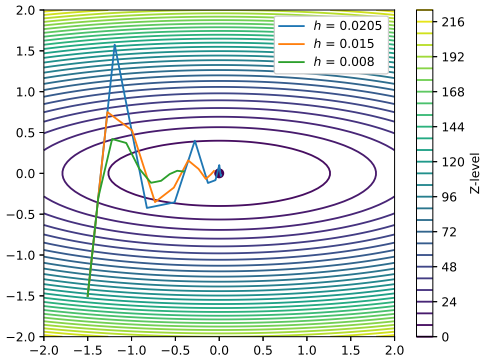


Figure 2.3: Ten iterations of heavy ball method (2.8) on example function (2.6) for different time-steps h.

our setting, the differential equations simply contain some form of relation between the function, its gradient, and a random perturbation.

This relation is the foundational block of the kind of SDEs we are dealing with in this thesis, one can further modify the equations by adding new terms or relations such that new characteristics can emerge. In general terms, we write an SDE as

$$dY(t) = f(Y(t), t)dt + g(Y(t), t)dB(t), \quad Y(0) = y \in \mathbb{R}, \tag{2.9}$$

where the $f(Y(t), t)dt$ represents the deterministic term, $g(Y(t), t)dB(t)$ is the stochastic forcing and $B(t)$ denotes the Brownian motion. Here, $f : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$ and $g : \mathbb{R}^d \times [0, T] \to \mathbb{R}^{d \times d}$.

Brownian motion is a stochastic process that describes the dynamics of a small particle suspended in a fluid. The movement is caused by the repeated collisions occurring between the particle and the molecules in the fluid, resulting in a change of direction for the particle. Each movement of the particle is seen as independent and the overall movement is thought to be random. A natural assumption used in the derivation of Brownian motion is that random bombardment is normally distributed, resulting in the fact that the increments of Brownian motion are normally distributed.

2.3 Euler–Maruyama method

Modeling an optimization method as an SDE does allow one to perform a theoretical analysis of the problem. However, if one wants to simulate the process it needs to be discretized [12]. Previously, when working with ODEs, one might have used the Euler method for discretization. This would mean if one wanted to discretize the equation such as

$$dY(t) = f(Y(t), t)dt, \qquad Y(t_0) = y, \tag{2.10}$$

one would write it as follows:

$$Y_{k+1} = Y_k + hf(Y_k, t_k), \qquad Y_0 = y \tag{2.11}$$

with h being the time-step such that $h = t_{k+1} - t_k$. While this method is perfectly viable for ODEs, it does not work with SDEs, due to the stochastic process not being differentiable in time. Instead, the method has to be slightly altered to account for the stochastic term. Here one approach is to use the Euler–Maruyama method. The method solves the issue of stochasticity by introducing a new randomly distributed time-step for the stochastic term, $\Delta B_k \sim \mathcal{N}(0, h)$. ΔB_k represents the increment of the Brownian motion $B(t)$ over the time interval $[t_k, t_{k+1}]$.

To show how it works, let us look at the SDE,

$$dY(t) = f(Y(t), t)dt + g(Y(t), t)dB(t), \qquad Y(t_0) = y, \tag{2.12}$$

where $B(t)$ is the Brownian motion. To discretize it, one would instead account for the terms separately and write

$$Y_{k+1} = Y_k + hf(Y_k, t_k) + g(Y_k, t_k)\Delta B_k, \qquad Y_0 = y. \tag{2.13}$$

This means that we now have a way of discretizing SDEs which allows us to simulate them.

2.4 Test functions for optimization

When comparing different optimizers, one wants to do so in a manner that mimics real-world applications. This means finding functions that challenge the methods in different ways. For this, different test functions are used. There exists a multitude of different test functions, each having its own unique characteristics. In this thesis, the Rastrigin function, the Rosenbrock function and the Himmelblau function are used as benchmark functions.

The Rastrigin function is a non-convex test function, where the global minimum is located at $(0, 0)$ in the 2-dimensional case. Surrounding the global minimum is a large number of local minima, see Figure 2.4a. The equation describing the function is,

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10 \cos 2\pi x_i \right], \tag{2.14}$$

d being the dimension of $\mathbf{x}$ and $x_i \in [-3, 3]$.

Having a large number of local minima typically makes it challenging to find the global minimum, since one easily gets stuck in the surronding local minima. And since the number of minima is large, small perturbations usually result in movement from one minimum to the next, making little progress toward the global minimum.

The Rosenbrock function is a non-convex test function described by the following equation:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2, \quad (x, y) \in \mathbb{R}^2. \tag{2.15}$$

Here we use the parameter values $a = 1$ and $b = 100$. This means that the global minimum will be located at (1,1), see Figure 2.4b.

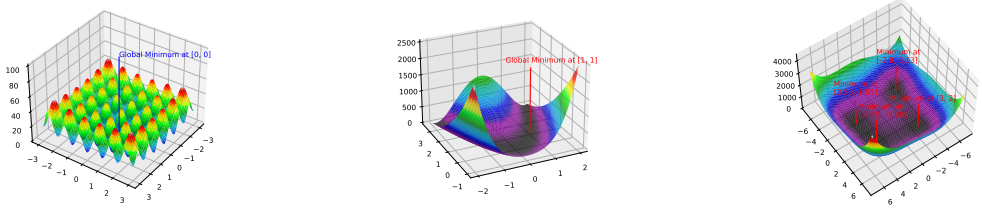
The Rosenbrock function differs from both the Rastrigin and Himmelblau's function in that it only has one single local minimum, which is also the global minimum. Finding the minimum is not so trivial. The minimum is located in a narrow flat valley, resulting in large differences in gradients depending upon which direction one is traversing. This means that one has to be especially careful when choosing the time-step such that one does not overshoot the valley, while still making progress towards the minimum inside the valley.

The Himmelbalu's function is a test function containing four different minima described by the following function:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2, \quad (x, y) \in \mathbb{R}^2. \tag{2.16}$$

What sets this test function apart from the rest, is that the minima are equally valued and that the placements of the minima are asymmetrical, see Figure 2.4c.

Having multiple equally valued minima might be an issue when using particle based optimization. This is due to the interaction between particles close to different minima clashing, causing the particles to move in an undesired direction.



(a) 3D plot of Rastrigin function (2.14).

(b) 3D plot of Rosenbrock function (2.15).

(c) 3D plot of Himmelblau's function (2.16).

Figure 2.4: 3D plots of different test functions

2.5 Notation and Overview

In the following chapters, we use the following notation. For calculating gradients with respect to the variable x, ∇_x is used. When performing discretization, the time-step parameter h is used. By $||\cdot||$ we denote the Euclidean norm of a vector. With $\ominus$, we denote a mathematical operation where each of the columns of the matrix is subtracted by a vector. $\otimes$ is the Kronecker product. $\langle \cdot, \cdot \rangle$ is the dot product.

In Chapter 3, the ensemble Langevin dynamics is introduced, and a first order algorithm is presented. The algorithm is then tested on different benchmark functions for various parameters and the results are then compared to gradient descent. In Chapter 4, a second order variant of the ensemble Langevin algorithm is introduced with some slight modifications to the simulation setup and model. Further tests are

then performed, comparing the second order algorithm to the first order algorithm. In Chapter 5, consensus based optimization with repulsive interaction is detailed. The method is then tested in different environments with different parameters to test the effectiveness of the new variant compared to normal CBO. The results are then also compared with the ensemble Langevin dynamics to determine in which situations the algorithms are favorable. In Chapter 6, a gradient free approach to estimating the ensemble Langevin dynamic is introduced. The method is then tested on a problem of training a neural network for classifying a spiral dataset. We also compare it to training with stochastic gradient descent.

For reproduction of results, we also provide the code on GitHub.

3

Optimization via ensemble Langevin dynamics

In Section 3.1, we present the first model which is based on the overdamped Langevin stochastic differential equation (SDE). We discretize the SDE using the Euler-Maruyama scheme and discuss the implementation part in Section 3.2. In Section 3.3, we showcase numerical illustrations on some benchmark functions.

3.1 Ensemble Langevin dynamics

The optimization method that we will discuss here, is based on the interacting particle version of the Langevin SDE. Compared to the standard overdamped Langevin SDE, a covariance is introduced in the dynamics. This leads to an interaction among multiple particles in the system. The method will be referred to as **ensemble Langevin optimization**. The method is based on the sampling method introduced in [7]. Let $N \in \mathbb{N}$ denote the number of particles. Let $(B^i(t))_{t\geq 0}$, $i = 1, \ldots, N$ be standard $d-$dimensional Brownian motions. The SDE driving the interacting particle system is as follows:

$$dX^{i,N}(t) = -C^N(t)\nabla_x f(X^{i,N}(t))dt + \sqrt{\frac{2C^N(t)}{\beta(t)}}dB^i(t), \tag{3.1}$$

where $X^{i,N}(t) \in \mathbb{R}^d$ denotes the position of $i-$th particle at time t. The ensemble covariance $C^N(t)$ is defined as

$$C^N(t) = \frac{1}{N}Q^N(t)Q^N(t)^\top, \tag{3.2}$$

where $Q^N(t) = [X^{1,N}(t) - \bar{X}^N(t), ..., X^{N,N}(t) - \bar{X}^N(t)]$ with $\bar{X}^N(t)$ being the ensemble mean given by

$$\bar{X}^N(t) = \frac{1}{N}\sum_{j=1}^{N} X^{j,N}(t). \tag{3.3}$$

Moreover, $\beta(t) > 0$ for all t is an increasing function, such that the noise diminishes over time.

The standard overdamped Langevin SDE looks like

$$dX(t) = -\nabla_x f(X(t))dt + \sqrt{\frac{2}{\beta}}dB(t), \quad X(0) = x. \tag{3.4}$$

It is known (see, e.g., [13]) that if we simulate the above SDE for a long time T then it generates a sample from the probability distribution whose probability density is

$$\rho(x) = \frac{1}{Z} e^{-\beta f(x)}, \tag{3.5}$$

where $Z = \int_{\mathbb{R}^d} e^{-\beta f(x)} dx$ is the normalizing constant. If β is large then most of the mass is concentrated near the modes of the distribution, see Figure 3.1.

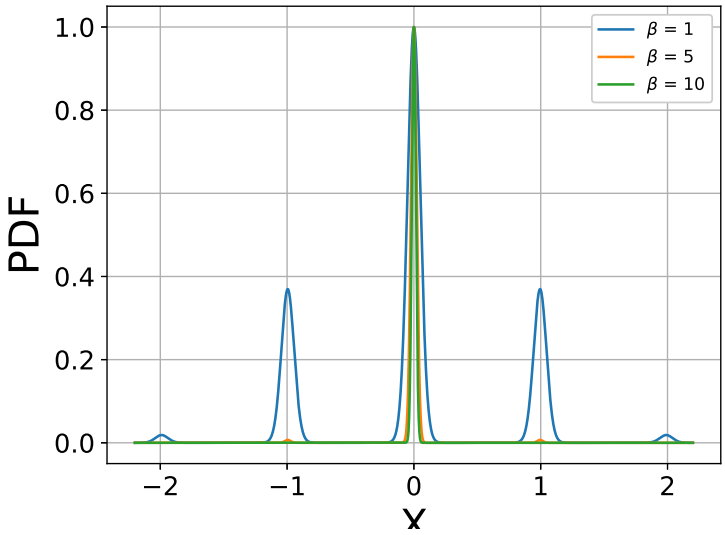


Figure 3.1: Probability distributions function of (3.5) with varying β where f is the $1-$dimensional Rastrigin function (2.14).

If we make β to be time-dependent then SDE (3.4) correspond to simulated annealing [14] for global non-convex optimization.

One can view (3.1) as an $Nd-$dimensional SDE. Another way of looking at it is instead as a system of interacting particles, where N is the number of $d-$dimensional particles interacting with each other, via the ensemble covariance term. In the same spirit as simulated annealing, we aim to utilize the interacting version of the Langevin SDE for optimization purposes.

One of the major benefits is that any particular particle, at any particular time, not only has the local information coming from the gradient evaluation, but also information coming from other particles. This is important for global optimization.

In certain machine learning tasks, some models are extremely high dimensional (for example neural networks), for those tasks finding the global minimum is not the main interest, but rather the convergence to a sufficiently good minimum. Here, a sufficient minimum is one which is good enough for generalization purposes. In such scenarios, the benefit of the ensemble methods is the possibility of utilizing the Kalman approximation for the gradients. The latter makes the method less computationally demanding, while still finding a good enough minimum. We will revisit this in Chapter 6.

3.2 Implementation

To perform numerical simulations, SDE (3.1) is discretized using the Euler-Maruyama scheme. Consider a time interval $[0, T]$ and partition it into evenly split time intervals, such that $0 = t_0 < t_1 < ... < t_n = T$, with h being the discretization step defined as $h = t_k - t_{k-1}$. It is clear that $h = t_n/n$.

Let the initial positions of the particles be $X_0^{i,N} = X^{i,N}(0)$. Here, we treat $Q/\sqrt{N}$ as a valid approximation for the square root of the ensemble covariance $\sqrt{C}$ [15]. Now, we write the Euler-Maruyama scheme

$$X_{k+1}^{i,N} = X_k^{i,N} - C_k^N \nabla_x f(X_k^{i,N})h + \sqrt{\frac{2}{N\beta_k}} Q_k^N \xi_{k+1}^i h^{\frac{1}{2}}, \quad k = 0, \dots, n-1, \tag{3.6}$$

where $\xi_{k+1} \sim \mathcal{N}(0, I_N)$, I_N is an $N-$dimensional identity matrix, $\beta_k := \beta(t_k)$ and

$$C_k^N := \frac{1}{N} Q_k^N (Q_k^N)^\top, \tag{3.7}$$

where $Q_k^N = [X_k^{1,N} - \bar{X}_k^N, ..., X_k^{N,N} - \bar{X}_k^N]$ with $\bar{X}_k^N$ being the ensemble mean given by

$$\bar{X}_k^N = \frac{1}{N} \sum_{j=1}^{N} X_k^{j,N}. \tag{3.8}$$

Remark 3.1. Pseudo-code for the numerical simulations can be seen in Algorithm 1. Note that $*$ is used when element-wise matrix multiplication is performed. Also note that the gradient of the function is evaluated in the algorithm, this will produce a matrix where each of the columns represent the gradient for a different particle.

For reproduction of results, we also provide the code on GitHub

Algorithm 1 Ensemble Langevin dynamics based optimization.

Initialize h, N, T
$n \leftarrow T/h$
$X \leftarrow X_0 \in \mathbb{R}^{dxN}$ ▷ Note, every particle computation is performed in parallel
for $k = 1, 2, ..., n$ **do**
  $\bar{X}_k^N \leftarrow \frac{1}{N} \sum_{j=1}^{N} X_k^{j,N}$ ▷ Sum for axis=0, meaning the dimension axis
  $Q_k^N \leftarrow [X_k^{1,N} - \bar{X}_k^N, ..., X_k^{N,N} - \bar{X}_k^N]$
  $C_k^N \leftarrow \frac{1}{N} Q_k^N (Q_k^N)^\top$
  $\xi_{k+1} \leftarrow \mathcal{N}(0, I_{d \times N})$
  $X_{k+1}^N \leftarrow X_k^N - C_k^N \nabla_x f(X_k^{i,N})h + \sqrt{\frac{2}{N\beta_k}} Q_k^N * \xi_{k+1} h^{\frac{1}{2}}$
end for

One issue is that $C_k \nabla f$ may not have linear growth [16]. This means that the Euler scheme does not necessarily converge for all time-steps. To get rid of this issue, in stochastic numerics literature, certain taming techniques have been proposed [17, 18]. We test two versions of the tamed Euler scheme. We define them below:

- One in which taming happens particle-wise [19] i.e.

$$X_{k+1}^{i,N} = X_k^{i,N} - \frac{C_k^N \nabla_x f(X_k^{i,N})}{1 + h\|C_k^N \nabla_x f(X_k^{i,N})\|} h + \sqrt{\frac{2}{\beta_k N}} Q_k^N \xi_{k+1}^i h^{\frac{1}{2}}, \tag{3.9}$$

  where $\|C_k^N \nabla_x f(X_k^{i,N})\|$ is the Euclidean norm for the coordinates,

- and the other in which they are coordinate-wise tamed (3.10), i.e.

$$X_{k+1}^{i,N} = X_k^{i,N} - \mathbb{B}(h, X_k^{i,N}) C_k^N \nabla_x f(X_k^{i,N}) h + \sqrt{\frac{2}{\beta_k N}} Q_k^N \xi_{k+1}^i h^{\frac{1}{2}}, \tag{3.10}$$

  with

$$\mathbb{B} = \text{Diag}\left(\frac{1}{1 + h|(C_k^N \nabla_x f(X_k^{i,N}))_1|}, \ldots, \frac{1}{1 + h|(C_k^N \nabla_x f(X_k^{i,N}))_d|} \right),$$

  where $(C_k^N \nabla_x f(X_k^{i,N}))_j$ denotes $j-$th coordinate of $C_k^N \nabla_x f(X_k^{i,N})$.

For the sake of brevity, we will first analyze the performance of ensemble Langevin dynamics based optimization by discretizing it via Euler-Maruyama and varying different parameters. However, we will also test tamed versions in the end.

3.3 Numerical illustrations

To determine the performance of the algorithm, it is tested on the Rastrigin function.

The performance is measured in successful attempts. An attempt is successful if the weighted average of the particles $\bar{X}_n$ have an Euclidean distance of less than 0.1 from the minimum. The weighted average $\bar{X}_n$ is calculated as

$$\bar{X}_n = \frac{\sum_{i=1}^N X_n^{i,N} \exp\left(-20 f(X_n^{i,N})\right)}{\sum_{i=1}^N \exp\left(-20 f(X_n^{i,N})\right)}. \tag{3.11}$$

The weight used in (3.11) is determined empirically. One can choose any other value, other than 20, provided it does not result in NaN (not a number) values during simulation (see denominator in (3.11).

Each simulation is run 1000 times, with $h = 0.0015$, $T = 1.5$ which means $n = 1000$. The particles are initialized uniformly in $[-3, 3]^d$.

To get a better perspective of the results and to understand how the covariance and noise affect the movement of particles, we test the following methods:

- Discretized version of ensemble Langevin dynamics (EL) (3.6)
- Discretized version of ensemble model without noise which we call as ensemble covariance affected gradient dynamics (ECAGD)
- gradient descent (GD) (2.3)

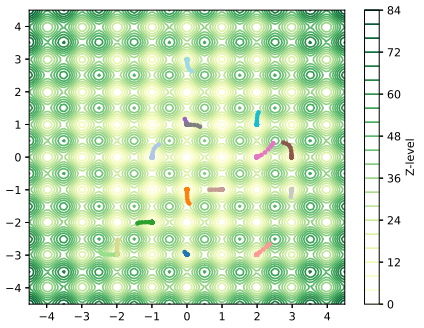
- gradient descent with small Gaussian noise (GDWN) (3.12)

Gradient descent is chosen due to it being both one of the simplest and also one of the most popular optimization algorithms. The gradient descent model with noise that is used can be described as,

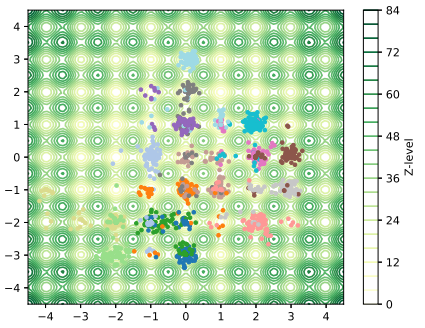
$$X_{k+1}^{i,N} = X_k^{i,N} - \nabla f(X_k^{i,N})h + \sqrt{\frac{2}{\beta_k}}\xi_{k+1}^i h^{\frac{1}{2}}, \tag{3.12}$$

where $\xi_{k+1} \sim \mathcal{N}(0, I_d)$, $X^{i,N}(0) = X_0^{i,N}$ and $h = t_{k+1} - t_k$.

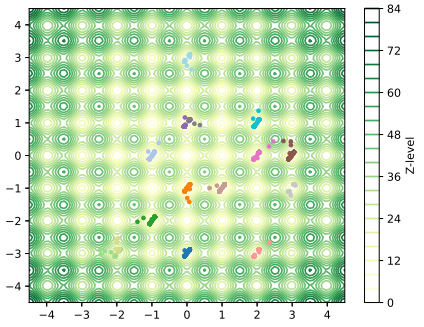
While the equations themselves explain how the different methods work, it can sometimes be difficult to visualize the effect in practice. To showcase that effect, contour plots are shown, see Figure 3.2.



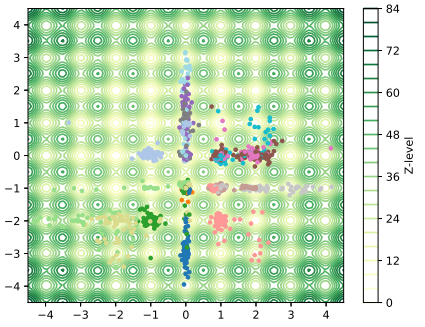
(a) Gradient descent.



(b) Gradient descent with noise.



(c) ECAGD (EL without noise).



(d) Ensemble Langevin (3.6).

Figure 3.2: Contour plots of different optimization algorithms on the Rastrigin function (2.14). $h = 0.0015$, $N = 15$, $T = 1.5$ and $\beta(t)$ from (3.13) with $\beta = 0.3$.

By looking at the contour plots, one can see the different approaches the method takes in finding the global minimum. While gradient descent is successful in finding local minima, it struggles with exploration and will only find the global minimum if it happens to have particles in its close vicinity, see Figure 3.2a. Introducing noise does seem to improve the result. However, it will only improve the results if it happens to move in the right direction. Over the time this does not matter as this

random behavior results in better exploratory behavior. Due to the shape of the Rastrigin function, even if it contains many shallow valleys, the overall shape tends towards a valley leading to the global minimum. This means that even if random noise perturbs the method, over time it will eventually find the global minimum, see Figure 3.2b. In the case of ensemble covariance gradient, it is only slightly different from gradient descent. In this case, the effect of the covariance is too small to be able to escape the local minima, see Figure 3.2c. With the introduction of noise in the ensemble setting, however, multiple particles manage to escape their local minima and find the global minimum. This is due to the intensity of noise in combination with interactions between the particles, see Figure 3.2d.

Effect of β : We first study the effect of $\beta(t)$. As will become clear by the end of this discussion, the choice of $\beta(t)$ is one of the most crucial choices to be made for the method to produce good results. In that pursuit, we fix $N = 15$.

Due to $\beta(t)$ only being present in two out of the four algorithms. Conducting repeated tests on the other two would only yield redundant information. Instead, these results are separated, see Table 3.1.

Table 3.1: Success rate corresponding to $N = 15$.

Method	Success rate
Gradient descent	42.2%
ECAGD	24.3%

Since the methods are gradient based and contain no noise, they are deterministic. This will result in the weighted distance from the minimum decreasing until each particle has reached its closest local minimum, this can be seen in Figure 3.3. Note that the distance of the ensemble method fluctuates for the remaining duration of the algorithm, this is due to the interaction between particles, however, since no noise is present, the interaction by itself is too small for the particles to escape from their current local minimum. This means that neither of the methods can reach the global minimum unless any of the particles starts from inside the slope leading towards the minimum.

Let us now examine how these results changes when noise is affecting the methods. As one can imagine, the choice of noise and its magnitude should have a significant impact on the performance of the algorithms. Our observations also showcase the same. Using constant $\beta(t)$ may not be optimal due to noise always being present and disturbing the algorithm's convergence towards the minimum. This can be seen in Table 3.2.

Comparing Tables 3.1 and 3.2, we see that the performance of ensemble Langevin dynamics is improved. However, the success-rate is not significantly better than that of gradient descent. As we already highlighted, this might be due to non-diminishing noise.

To fix the issue of noise disturbing the convergence towards the end of the algorithm, the noise level $\beta(t)$, $t > 0$ is taken to be a function increasing with time. There is no doubt that there can be many possible choices of $\beta(t)$. In the works related to

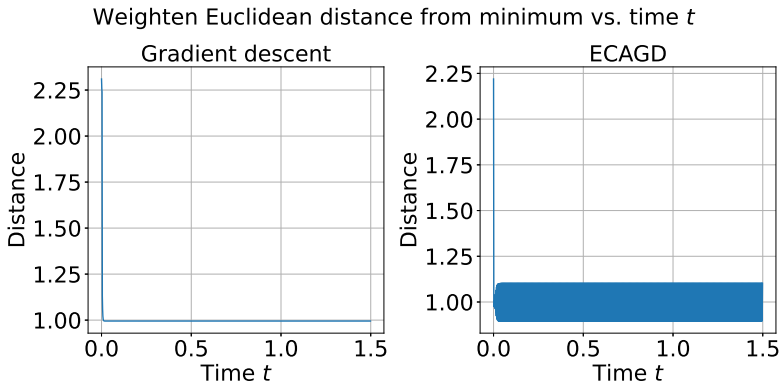


Figure 3.3: Euclidean distance of weighted average from global minimum on the Rastrigin function (2.14) corresponding to $h = 0.0015$, $N = 15$ and $T = 1.5$.

Table 3.2: Success rates for finding the global minimum on the Rastrigin function (2.14) using GDWN (3.12) and EL (3.6) with $N = 15$ and varying β

Method \ β	0.005	0.01	0.05	1.0	25.0
Gradient descent with noise	-	-	-	26.7%	36.4%
Ensemble Langevin	30.1%	**50.5**%	36.2%	19.4%	20.3%

simulated annealing [14], it is proposed to take $\beta(t) = \ln(1 + t)$ which we found in our experiments to be too slow for the convergence. We tested some of the other possible choices of $\beta(t)$ which we discuss below.

Initially, we test by simply having a cut-off at the last 50 iterations in which the $\beta(t)$ increased to 15400. This implies we take a constant $\beta(t) \equiv \beta$ till $t = 1.425$ and then we take $\beta(t) \equiv 15400$ for $1.425 \leq t \leq 1.5$. It is defined as

$$\beta(t) = \begin{cases} \beta, & \text{if } t < 1.425, \\ 15400, & \text{otherwise.} \end{cases} \tag{3.13}$$

Tests are then performed by varying β. This new approach of fixing $\beta(t)$ shows great improvement, as can be seen in Table 3.3. Note here that optimal β is different for the two methods, in order to make a fair comparison in the following simulations, individualized β are used.

Table 3.3: Success rates for finding the global minimum on the Rastrigin function (2.14) using GDWN (3.12) and EL (3.6) with $N = 15$ and $\beta(t)$ from (3.13) and varying parameter β.

Method \ β	0.01	0.015	0.025	25.0	100.0
Gradient descent with noise	6.6%	11.1%	14.2%	**42.3**%	36.8%
Ensemble Langevin	64.6%	**84.1**%	78.2%	22.0%	18.3%

The variance of the methods is also tested by looking at box-plot of the Euclidean

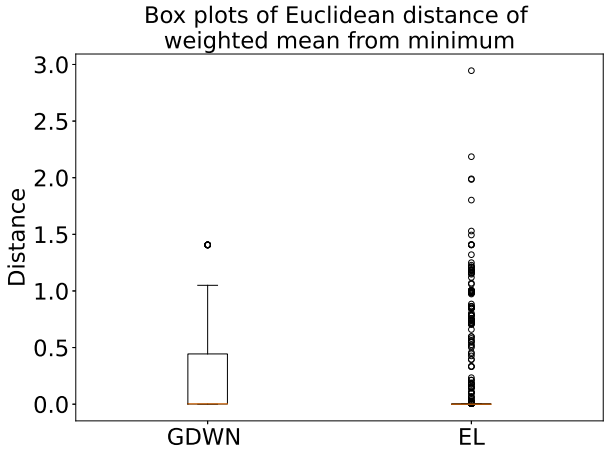


Figure 3.4: Box-plot of Euclidean distance of weighted mean from global minimum on the Rastrigin function (2.14) using GDWN (3.12) and EL (3.6) with $N = 15$ and $\beta(t)$ from (3.13) with $\beta = 0.2$ for GDWN and $\beta = 0.015$ for EL.

distance from the minimum. What is worth noting here is that the variance of ensemble Langevin is significantly lower than that of gradient descent, see Figure 3.4. Bear in mind that these results are problematic, since ensemble Langevin contains a substantial amount of outliers. This is the case due to its high success-rate, meaning that each run which was not a success is seen as an outlier, which makes the variance negligible. One could correct this error by instead comparing the non-successful runs. The issue with this however, is that the result would be skewed toward the method with a lower success-rate with results closer to but not quite at the minimum.

While the results using constant non-diminishing noise with a cut-off looks very promising, it has some issues. Usually when running these algorithms, one would want a stopping criteria, such that the algorithm stops when the goal is achieved. This is useful, not only for saving computational power, but also so that the algorithm does not continue exploring after finding the global minimum, with the risk of not being able to converge back to it. Here, by looking at how the weighted average of the distance converges towards the global minimum, it is unclear if the global minimum is found before the noise is cut-off, which makes it difficult to implement a stopping criteria, see Figure 3.5

Further testing is then performed on differently shaped $\beta(t)$. For the convenience of the reader we plot these functions as can be seen in Figure 3.6. The choice of $\beta = 0.015$ is motivated by the results in Table 3.3 where $\beta = 0.015$ corresponds to the best result. The goal of testing differently shaped $\beta(t)$ is to see if the algorithm would perform better depending on how the magnitude of the noise changes over time. Note that $\beta = 0.2$ will be used for gradient descent with noise instead of $\beta = 0.015$, due to it seemingly performing best with that β.

The $\beta(t)$ in the left of Figure 3.6 linearly decreases the noise instead of reducing it

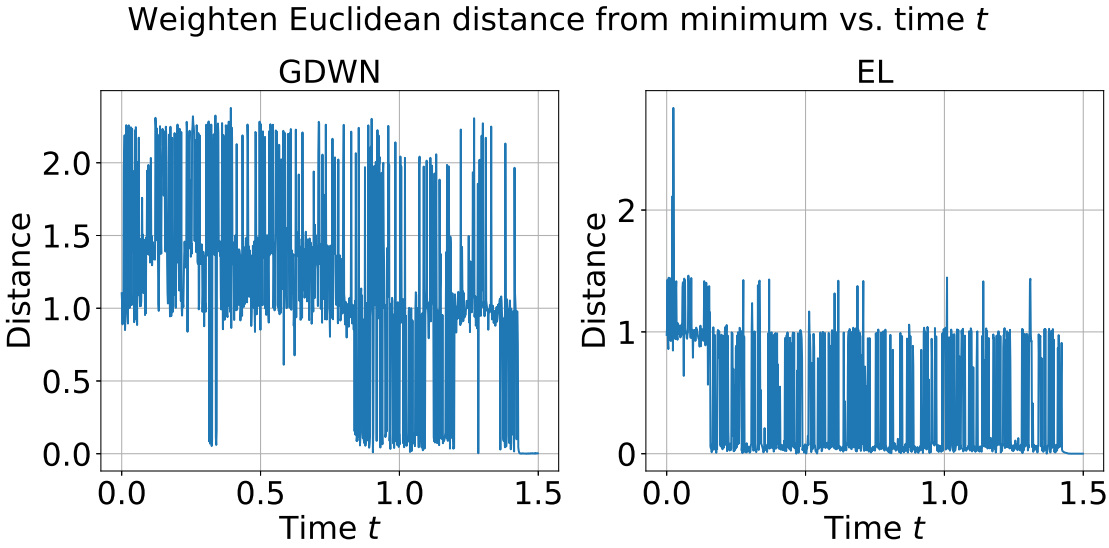


Figure 3.5: Euclidean distance of weighted average from global minimum on the Rastrigin function (2.14) using GDWN (3.12) and EL (3.6) with $N = 15$ and $\beta(t)$ from (3.13) with $\beta = 0.2$ for GDWN and $\beta = 0.015$ for EL.

instantaneously (compared to Table 3.3). It is defined as

$$\beta(t) = \begin{cases} 0.015, & \text{if } t < \alpha, \\ \frac{15400}{15400-\alpha}(t-\alpha) + 0.015, & \text{otherwise.} \end{cases} \tag{3.14}$$

The results for different α can be seen in Table 3.4

Table 3.4: Success rates for finding the global minimum on the Rastrigin function (2.14) using GDWN (3.12) and EL (3.6) with $N = 15$ and $\beta(t)$ from (3.14) and varying parameter α.

Method \ α	0.0	0.15	0.75	1.2	1.35	1.425
GDWN	42.2%	43.7%	56.7%	58.9%	61.7%	**62.6**%
EL	22.7%	58.1%	84.1%	**85.2**%	81.5%	84.3%

The function $\beta(t)$ in the right of Figure 3.6 has low noise in the beginning, slowly increasing the amount of exploration before reaching a defined threshold. The algorithm then explores for a few hundred iterations at this threshold before linearly reducing the noise. It is defined as

$$\beta(t) = \begin{cases} 0.015 - 15400(1 + \frac{1}{\alpha}), & \text{if } t < \alpha, \\ 0.015, & \text{if } \alpha < t < \alpha + 600, \\ 0.015 + \frac{15400}{15400-\alpha-600}(t-\alpha-600), & \text{otherwise.} \end{cases} \tag{3.15}$$

The results for different α can be seen in Table 3.5.

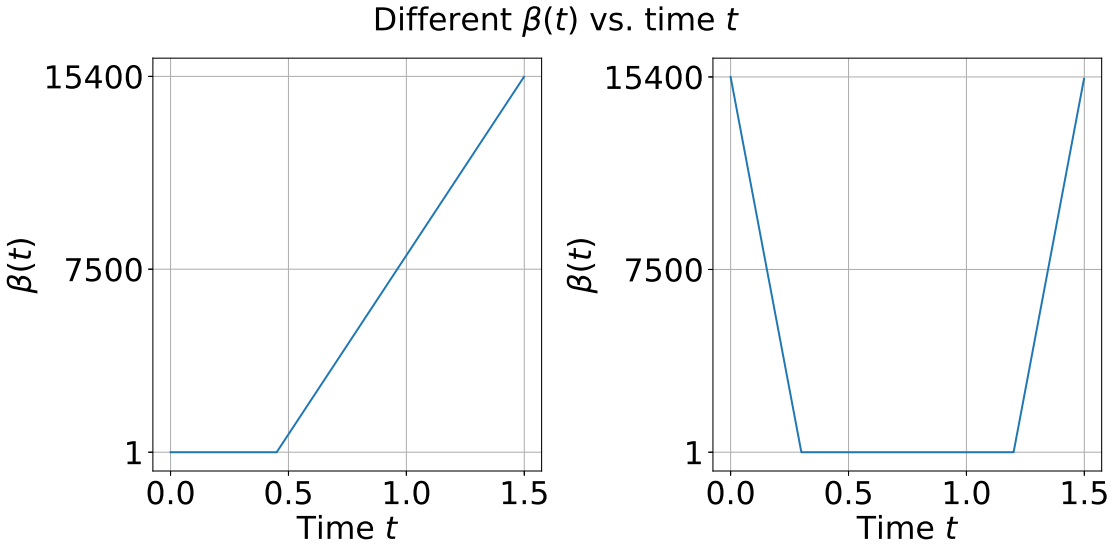


Figure 3.6: Different time-dependent $\beta(t)$, left figure shows a constant $\beta = 0.015$ for a few iterations, before linearly increasing to 15400, from (3.14). The right figure shows a linearly decreasing $\beta(t)$ from 15400 to 0.015 for a few iterations, then remaining at 0.015 for a few iterations before linearly increasing back to 15400, from (3.15).

Table 3.5: Success rates for finding the global minimum on the Rastrigin function (2.14) using GDWN (3.12) and EL (3.6) with $N = 15$ and $\beta(t)$ from (3.15) and varying parameter α.

Method \ α	0.15	0.3	0.45	0.525
Gradient descent with noise	57.8%	55.0%	55.3%	56.7%
Ensemble Langevin	84.1%	84.3%	82.3%	83.9%

If we choose a $\beta(t)$ which allows the dynamics enough time to sufficiently explore the search space, then our results indicate that the $\beta(t)$ from equations (3.14) and (3.15) have comparable performance. This suggests that the performance is affected mostly from two sources, the magnitude of the noise, and the exploration time. The magnitude of the noise should be large enough such that its effect is not nullified by the exploitation, but small enough that it does not skip minima. For the exploration time, it should be maximized and only stopped to give the dynamics enough time to perform the final convergence.

For the sake of analysis, however, unless the performance is significantly worse, it might still be of interest to use a $\beta(t)$ such that the noise decreases slowly over time. This can make it easier to track trajectories of other metrics over time, such as distance from the minimum without having the noise overshadow the results, see Figure 3.5. While the figure clearly shows that the distance from the minimum does decrease with time, it might be difficult to determine the rate at which it does so. Having that information might be critical in determining eventual stopping criteria

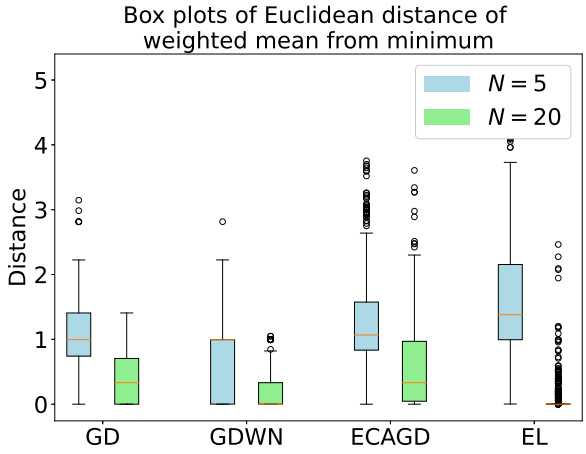


Figure 3.7: Box-plot of Euclidean distance of weighted average from global minimum on the Rastrigin function (2.14) for different number of particles N with $\beta(t)$ from (3.13) with $\beta = 0.015$ for ensemble Langevin and $\beta = 0.02$ for gradient descent with noise.

or optimal parameter values. This is why, even if the results are slightly worse, slowly decreasing the noise can be of value.

Effect of N : The results indicate that increasing the particle number improves the result overall, which is logical since more of the parameter space is explored. The results can be seen in Table 3.6.

Table 3.6: Success rates for finding the global minimum on the Rastrigin function (2.14) with $\beta(t)$ from (3.13) with $\beta = 0.015$ for ensemble Langevin and $\beta = 0.02$ for gradient descent with noise and varying parameter N.

Method \ Number of particles	5	10	15	20
Gradient descent	17.80	30.40	41.50	47.90
Gradient descent with noise	30.90	47.40	59.50	68.10
ECAGD	12.50	18.10	25.20	27.90
Ensemble Langevin	8.60	47.20	82.90	90.90

While each method showed improvements by increasing the number of particles, ensemble Langevin not only improved in success-rate, but the variance also had a significant decrease which was not the case for the other methods, see Figure 3.7.

Effect of h : One of the issues with discretizing SDEs is choosing a suitable time-step. Having a small time-step usually results in accurate representation of the dynamics, however having a small time-step also requires one to perform more computations which is not ideal. Using a too-large time-step also suffers from issues, while it is beneficial computationally, since large times can be computed with few iterations, the accuracy of the method might deteriorate, and exploding gradients can occur. Testing the methods for different time-steps shows this, see Table 3.7. Worth noting is that gradient descent does not have the same issues with larger time-steps

as ensemble Langevin, the results are indisputably worse. However if only decent results are necessary, it might be worth using gradient descent for computational efficiency.

Table 3.7: Success rates for finding the global minimum on the Rastrigin function (2.14) with $N = 15$, $T = 1.5$ and $\beta(t)$ from (3.13) with $\beta = 0.3$ and varying time-step h.

Method \ h	0.0005	0.001	0.0015	0.003
Gradient descent	40.1%	40.3%	41.3%	38.3%
Gradient descent with Brownian noise	63.3%	61.1%	61.3%	57.9%
ECAGD	40.1%	39.3%	25.0%	5.7%
Ensemble Langevin	81.7%	83.9%	80.3%	11.3%

By looking at the box-plots of the non-successful runs for ensemble Langevin it shows that the variance is quite small, see Figure 3.8. There might be a slight benefit in choosing a smaller h since it seems as if the average distance from global minimum at last step is smaller than that of the other time-steps. Here, the larger time-step is excluded due to a high number of NaN values, caused by exploding gradients. These NaN values only occur when increasing the time-step beyond a certain threshold and has not been an issue for the other tests.

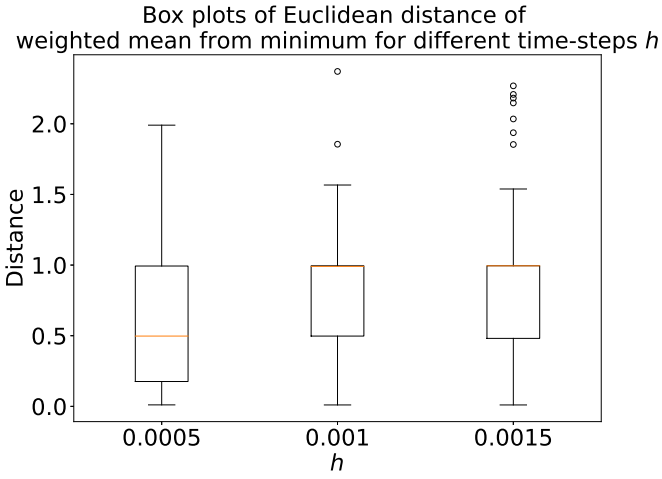


Figure 3.8: Box-plot of Euclidean distance of weighted average from global minimum on the Rastrigin function (2.14) using ensemble Langevin (3.6) for different time-step h corresponding to $N = 15$, $T = 1.5$ and $\beta(t)$ from (3.13) with $\beta = 0.015$ for ensemble Langevin.

Effect of T : Another crucial parameter for solving SDEs is the time T. While it is clear that increasing time usually results in better results, one does not want to unnecessarily perform computations if it yields no benefit. In practice, implementing a stopping criteria such that the algorithm stops when no further improvement has occurred is most logical. However, for the sake of comparison, the methods are tested for different times T. What is most noteworthy from the results, is that ensemble

Table 3.8: Success rates for finding the global minimum on the Rastrigin function (2.14) with $N = 15$, $h = 0.0015$ and $\beta(t)$ from (3.13) with $\beta = 0.015$ for ensemble Langevin and $\beta = 0.02$ for gradient descent with noise and varying time T.

Method \ T	0.25	1.5	5.0	10.0
Gradient descent	41.5%	39.4%	38.5%	39.9%
Gradient descent with Brownian noise	44.7%	61.9%	64.6%	65.8%
ECAGD	24.7%	24.0%	22.2%	24.0%
Ensemble Langevin	64.9%	84.3%	71.5%	64.2%

Table 3.9: Success rates for finding the global minimum on the Rastrigin function (2.14) using ensemble Langevin with particle-wise taming (3.9) and coordinate-wise taming (3.10) with $N = 15$, $T = 1.5$ and $\beta(t)$ from (3.13) with $\beta = 0.015$ for time-step h

Method \ h	0.0015	0.003	0.005	0.01
EL particle-wise taming	77.4%	56.2%	31.3%	6.8%
EL coordinate-wise taming	79.0%	58.6%	27.9%	1.9%

Langevin performance decreases after a certain amount of time. This might be due to the increase in exploration due to the time increase, which leads to the algorithm exploring too far such that it cannot converge. This can be improved with different choice of $\beta(t)$. Another thing worth noting is that decent success-rates are achieved in a relatively short time for ensemble Langevin, compared to the other methods, see Table 3.8.

Effect of taming : As we mentioned already, the Euler-Maruyama scheme may not converge. While this is the case, we still utilized it to discretize (3.1) for the sake of simplicity, in our previous experiments. The ensemble dynamics based optimization is indeed convincing from our previous experiments but it does not allow larger time-steps. To reap the benefit of larger time-steps without having exploding gradients, taming according to both (3.9) and (3.10) is tested. While the results are not perfect and there is a clear decline in performance while increasing the time-step h, comparing the results without taming, one can see that it has a positive impact, see Table 3.9 and Table 3.7. Looking at the tables, one can see that the result for $h = 0.003$ gets a significant increase, and the success-rate goes from 11.3% to a minimum of 56.2%

Using taming does remove the exploding gradient, however using larger h struggles with finding the minimum, which suggests that exploding gradients might not be the only reason behind a drop in performance for larger time-steps. This issue is likely caused by the stringent demands of the success-rate. We count a run as a success if the discretized dynamics reach the ball of radius 0.1 centered at global minimum, however, the tamed Euler scheme provides order of convergence $O(h^{1/2})$. Keeping this in mind, we will relax our success rate criterion in the next chapter.

4

Optimization via ensemble kinetic Langevin dynamics

In Section 4.1, we present ensemble kinetic Langevin SDEs. We discretize the SDEs using tamed Euler-Maruyama scheme and discuss the implementation in Section 4.2. In Section 4.3, we compare the precision and variance as well as the difference in performance of the ensemble kinetic Langevin dynamics to the previously tested ensemble Langevin dynamics for different benchmark functions.

4.1 Second order ensemble Langevin dynamics

While the first order ensemble Langevin dynamics discussed in Chapter 3 show some improvement in minimizing the Rastrigin function over gradient descent with Gaussian noise, there is a possibility of further improvements. One such improvement is to use the second order dynamics instead, which have been proposed and discussed in the sampling regime in [20]. It is described as

$$dX^{i,N}(t) = V^{i,N}(t)dt, \tag{4.1}$$

$$dV^{i,N}(t) = -C^N(t)\nabla_x f(X^{i,N}(t))dt - \gamma V^{i,N}(t)dt + \sqrt{\frac{2\gamma C^N(t)}{\beta(t)}}dB^i(t), \tag{4.2}$$

where $X^{i,N}(t) \in \mathbb{R}^d$ denotes the position of $i-$th particle at time t. The ensemble covariance $C^N(t)$ is defined as

$$C^N(t) = \frac{1}{N}Q^N(t)Q^N(t)^\top, \tag{4.3}$$

where $Q^N(t) = [X^{1,N}(t) - \bar{X}^N(t), ..., X^{N,N}(t) - \bar{X}^N(t)]$ with $\bar{X}^N(t)$ being the ensemble mean given by

$$\bar{X}^N(t) = \frac{1}{N}\sum_{j=1}^{N} X^{j,N}(t). \tag{4.4}$$

Moreover, $V^{i,N}(t)$ is the velocity for the $i-$th particle at time t, $\beta(t) > 0$ for all t is an increasing function, such that the noise diminishes over the time, and $\gamma > 0$ is a friction parameter.

Note that the position is now acquired by integrating the velocity over time and that is why we call such dynamics as second order dynamics. The inclusion of a

velocity term can be especially useful when traversing long distances down a slope (see Example 2.3 and discussion on the heavy ball method in Chapter 2). The benefit is that, since some of the previous motion is kept, it acts as a break if sudden shifts in direction would occur, which is useful in avoiding rapid oscillation when moving down a steep valley, for example, see Section 2.1.1. One does however need to be mindful about the choice of γ. If γ is small, the system will keep most of the previous momentum. This could lead to a potential minimum being skipped over due to large jumps each time-step. If γ instead is too large, the system reverts to a first order dynamics, which is most likely not desired.

4.2 Implementation

For simulations, the SDE is discretized using the tamed Euler-Maruyama scheme. A time interval $[0, T]$ is partitioned into evenly split time intervals, such that $0 = t_0 < t_1 < ... < t_n = T$, with h being the discretization step defined as $h = t_k - t_{k-1}$ and $n = T/h$.

Let the initial positions of the particles be $X_0^{i,N} = X^{i,N}(0)$. As in the previous chapter, the square root of the covariance matrix is estimated as $Q^N/\sqrt{N}$ (see (4.3)). Now, we write the tamed Euler-Maruyama scheme,

$$X_{k+1}^{i,N} = X_k^{i,N} + V_k^{i,N} h \tag{4.5}$$

$$V_{k+1}^{i,N} = V_k^{i,N} - \mathbb{B}(h, X_k^{i,N}) C_k^N \nabla_x f(X_k^{i,N}) h - \gamma V_k^{i,N} h + \sqrt{\frac{2\gamma}{\beta_k N}} Q_k^N \xi_{k+1}^i h^{\frac{1}{2}}. \tag{4.6}$$

with

$$\mathbb{B}(h, X_k^{i,N}) = \text{Diag}\left(\frac{1}{1 + h|(C_k^N \nabla_x f(X_k^{i,N}))_1|}, \dots, \frac{1}{1 + h|(C_k^N \nabla_x f(X_k^{i,N}))_d|} \right),$$

where $(C_k^N \nabla_x f(X_k^{i,N}))_j$ denotes $j-$th coordinate of $C_k^N \nabla_x f(X_k^{i,N})$ and $\xi_{k+1}^i \sim \mathcal{N}(0, I_N)$.

While the Euler-Maruyama discretization of second order dynamics usually has fewer issues of exploding gradients, it does not necessarily mean that it will converge to the continuous dynamics. Therefore, the gradient is tamed. This is done to ensure that the convergence to the continuous-time ensemble kinetic Langevin dynamics as h tends to 0.

4.3 Numerical illustrations

The experiments from the previous chapter show that ensemble Langevin dynamics performs better compared to gradient descent and gradient descent with noise. Therefore, comparisons will only be made with ensemble Langevin dynamics in this chapter. We also realize that our success criteria may not be relaxed enough that

it can take into account the error produced by time truncation i.e. due to finite T and discretization i.e. due to h.

The concern is that the innate error from simulating the dynamics may negatively affect the results. While the exact error is unknown, it is known to depend on both the time-step h and the total time T, i.e.

$$\varepsilon = \varepsilon_T + \varepsilon_h, \tag{4.7}$$

where ε is the error, and $\varepsilon_T \to 0$ as $T \to \infty$ and $\varepsilon_h \to 0$ as $h \to 0$.

Since a small time T is used in the previous chapter, a large error might be present, which can overshadow some of the results, leading to poor performance.

To balance the errors while still keeping the same number of computation, both the time-step h and the time T are modified. This means that the simulation parameters are as follows: the simulations are run 1000 times, with $h = 0.03$, and $T = 30$. The particles are initialized uniformly in $[-3, 3]^d$.

Performance is tested using successful runs. The criteria for success is adjusted as compared to the last chapter. A run is considered successful if the distance of each of the coordinate of the weighted average (3.11) of the position is less than 0.2 to the global minimum.

We also update $\beta(t)$ to accommodate these changes. The new $\beta(t)$ is

$$\beta(t) = \begin{cases} \beta, & \text{if } t < 28.5, \\ 1000000, & \text{otherwise.} \end{cases} \tag{4.8}$$

The second order dynamics does indeed have some features which in theory should perform better in certain cases. It is therefore tested against a few different test functions such that one can assess in which situations using second order dynamics can be beneficial and for which scenarios it is detrimental to the performance.

Remark 4.1. Pseudo-code for the algorithm can be seen in Algorithm 2. Note that notation $*$ is used for element-wise multiplication between matrices. Also note that the gradient of the function is evaluated in the algorithm, this will produce a matrix where each of the columns represent the gradient for a different particle.

The code used to generate all the results and figures can also be found at GitHub.

The test function that will be used are Rastrigin function, Rosenbrock function and Himmelblau's function.

Rastrigin function :

With the change of parameters, tests performed on both the time-step h and the time T seem to show some different results compared to the previous chapter which is expected, see Table 4.1.

Here it holds that an increase in time T yields better performance, which is to be expected. It also hold true that smaller time-steps tend to perform better.

Algorithm 2 Ensemble kinetic Langevin algorithm.

Initialize h, N, T, γ
$n \leftarrow T/h$
Initialize $X_0 \in \mathbb{R}^{dxN}$ and $V \leftarrow V_0 \in \mathbb{R}^{dxN}$ in search space ▷ Note, every particle computation is performed in parallel
for $k = 1, 2, ..., n$ **do**
$\bar{X}_k^N \leftarrow \frac{1}{N} \sum_{j=1}^N X_k^{j,N}$ ▷ Sum for axis=0, meaning the dimension axis
$Q_k^N \leftarrow [X_k^{1,N} - \bar{X}_k^N, ..., X_k^{N,N} - \bar{X}_k^N]$
$C_k^N \leftarrow \frac{1}{N} Q_k^N (Q_k^N)^\top$
$\xi_{k+1} \leftarrow \mathcal{N}(0, I_{d \times N})$
$V_{k+1}^N \leftarrow V_k^N - \mathbb{B}(h, X_k^N) * C_k^N \nabla_x f(X_k^N) h - V_k^N \gamma h + \sqrt{\frac{2}{N\beta_k}} Q_k^N * \xi_{k+1} h^{\frac{1}{2}}$
$X_{k+1}^N \leftarrow X_k^N + V_k^N h$
end for

Table 4.1: Success rates for finding the global minimum on the Rastrigin function (2.14) using ensemble kinteic Langevin dynamics (4.5) and (4.6) with $N = 15$, $\gamma = 1$ and $\beta(t)$ from (4.8) with $\beta = 0.05$ and varying time T and time-step h.

h \ T	1	2	10	25
0.01	16.5 %	24.4 %	52.8 %	**62.6**%
0.03	17.1 %	25.5 %	55.2 %	58.6 %
0.05	13.8 %	22.2 %	40.6 %	21.4 %
0.1	11.1 %	13.8 %	20.5 %	1.0 %

Testing is also performed by varying both γ and β. Here the choice of values is less intuitive. From the results, it seems as if larger β performs better. This might be due to some of the momentum being kept, which already allows an increase in the step-size, see Table 4.2. It is also seen that most of the momentum is kept from the previous time-step, which can be beneficial while moving down the slope of each minima.

While the second order does seem to perform rather well with the right parameters, it seems as if the results are similar to that of the first order, see Table 3.7. Here, the criteria for success are also more lenient, which should further the advantage. The reasoning as to why the performance is not greater for the second order dynamics is unclear. The reason might simply be that the optimal parameters are yet to be found.

To broaden the comparison between the first and second order dynamics, further testing will be performed on another test function.

Table 4.2: Success rates for finding the global minimum on the Rastrigin function (2.14) using ensemble kinteic Langevin dynamics (4.5) and (4.6) with $N = 15$, $T = 30$ and $h = 0.03$ with varying $\beta(t)$ from (4.8) with and varying γ.

γ \ β	0.01	0.03	0.05	0.1
1	0.0 %	8.1 %	55.8 %	69.2 %
3	0.2 %	43.5 %	75.2 %	64.9 %
5	5.2 %	55.3 %	**83.2** %	56.9 %
20	30.8 %	76.4 %	69.5 %	44.6 %

Rosenbrock function :

Initial tests with changing parameters h and T seem promising, see Table 4.3.

Table 4.3: Success rates for finding the global minimum on the Rosenbrock function (2.15) using ensemble kinteic Langevin dynamics (4.5) and (4.6) with $N = 15$, $\gamma = 1$ and $\beta(t)$ from (4.8) with $\beta = 2$ and varying time T and time-step h.

h \ T	1	2	10	25
0.01	11.8 %	25.9 %	94.7 %	**97.3**%
0.03	9.2 %	25.9 %	80.6 %	95.9 %
0.05	11.4 %	19.9 %	69.0 %	92.1 %
0.1	20.6 %	12.2 %	36.3 %	66.6 %

Here the results are quite consistent with the theory. As time T increases, so does the performance. The dynamics also seem to perform better as the time-step h decreases, which is to be expected. The one exception is in the case of small T, which can be explained by the presence of a larger error (4.7).

Testing for β and γ shows a clear difference in parameter choice depending on the test function. For the Rastrigin slightly larger γ seemed optimal with larger noise.

In this case, noise is comparatively less important with a preference for smaller γ, see Table 4.4.

Table 4.4: Success rates for finding the global minimum on the Rosenbrock function (2.15) using ensemble kinteic Langevin dynamics (4.5) and (4.6) with $N = 15$, $T = 30$ and $h = 0.03$ with varying $\beta(t)$ from (4.8) and varying γ.

γ \ β	0.1	0.5	1.0	2.0
0.5	1.3 %	33.5 %	45.2 %	50.5 %
1.0	31.5 %	80.7 %	93.3 %	**97.3** %
3.0	24.0 %	74.5 %	86.3 %	92.8 %
5.0	20.5 %	53.1 %	64.4 %	66.4 %

The reason for this might be the lack of non-global minima. Meaning that following the gradient will lead to the right path. Therefore having essentially a larger step-size by keeping some of the momentum leads to the minimum faster. This might also be why smaller noise is preferred since the particles do not have to explore as much, and moving along the same trajectory is ideal.

The benefits of the second order dynamics seem to have a greater effect on the Rosenbrock function than Rastrigin function.

Table 4.5: Success rates for finding the global minimum on the Rosenbrock function (2.15) using ensemble Langevin dynamics (3.6) with $N = 15$ and $\beta(t)$ from (4.8) with $\beta = 2$ and varying time T and time-step h.

h \ T	0.05	0.1	0.5	1.25
0.0005	61.0 %	63.6 %	76.3 %	83.9 %
0.0015	60.9 %	61.6 %	77.5 %	**86.6** %
0.0025	42.6 %	53.2 %	79.9 %	82.0 %
0.005	8.7 %	14.8 %	50.7 %	65.3 %

Notice that the simulation parameters are not identical but similar, this is due to the fact Euler discretization of first order dynamics does not allow to have a large time-step h. However, the number of computations remained the same, meaning that the time T is smaller for the first order. Increasing the time T to be equal to that of the second order dynamics might have resulted in similar results, however, this would be unfair in the sense that the number of computations needed for the first order dynamics would be higher.

One might also look at the contour plot of the different methods, see Figure 4.1. The two main takeaways from these plots are the following. First, for the first order dynamics, most of the particles moves to the bottom of the valley in only a few iterations. However, while at the bottom of the valley, the change in position is slowed drastically, which results in many iterations needed to move towards the minimum. Second, for the second order dynamics, the particles move at a slower

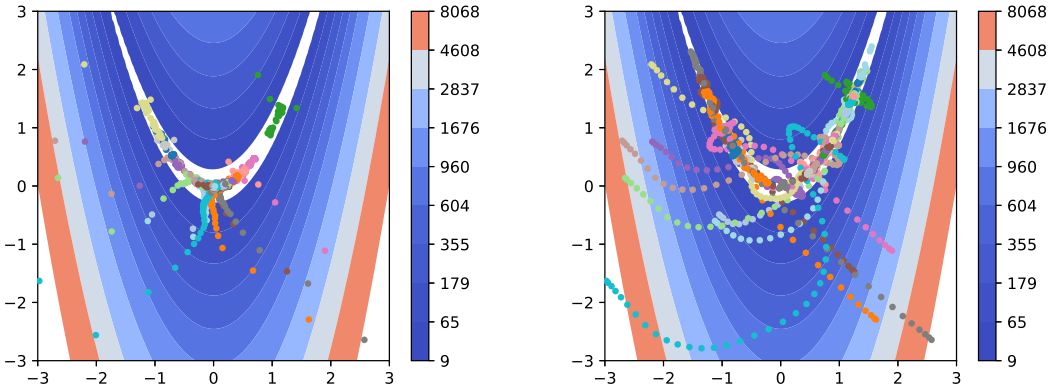


Figure 4.1: Contour plot of first and second order dynamics on the Rosenbrock function 2.15, the left figure is the first order dynamics (3.6), the right figure is the second order dynamics (4.5) and (4.6).

pace initially and overshoot the bottom due to momentum, however, the velocity seems to remain rather high while at the bottom of the valley, resulting in a quicker movement towards the minimum.

Himmelblau's function:

Looking at the Himmelblau function, one might wonder why it is worth testing the algorithm on. It has no local minima to get stuck in and no narrow valleys. Here what is tested is instead if the particles would sabotage each other. Since there are multiple global minima, one issue might be that each particle wants all other particles to move towards its closest minimum. This could result in the particles moving towards a middle ground where no global minimum is located.

Performing tests does however show that this is not an issue. While the algorithm is affected by other particles, the convergence due to the gradient outweighs the interaction between the particles, see Figure 4.2. This means that each particle always finds one of the global minima. In hindsight, testing a gradient-based algorithm on the Himmelbalu's function might not be that interesting, since following the gradient will always lead to one of the minima.

The only issue with this test function is that measuring performance using the weighted average does not make sense since multiple global minima exist. This is due to each of the particles in one of the minima being considered equally important, which leads to the average being somewhere between each of the minima instead of around one global minimum.

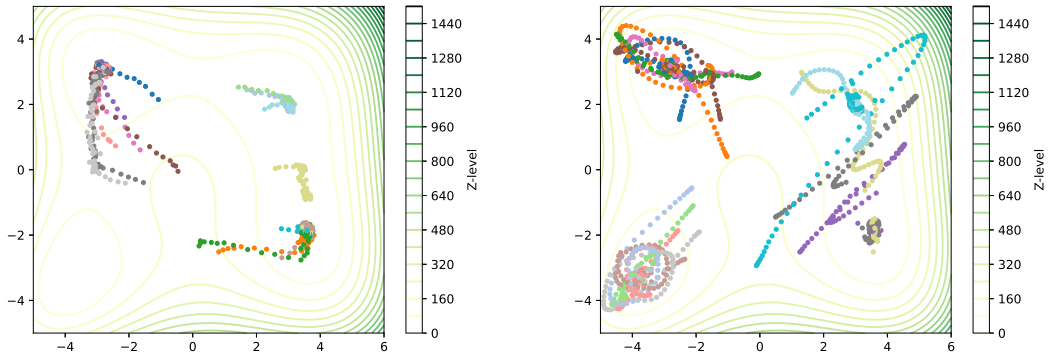


Figure 4.2: Contour plots of first and second order dynamics for the Himmelblau's function (2.16), the left figure is for the first order dynamics (3.6), the right figure is the second order dynamics (4.5)-(4.6).

5

Consensus based optimization with repulsive forces

In Section 5.1, we present a variant of CBO containing repelling exploration and explain its importance and functionality. In Section 5.2, it is explained how the numerical calculations can be performed on the dynamics using the Euler-Maruyama scheme. In Section 5.3, the simulation setup is explained, and the results are presented.

5.1 CBO with repelling exploration

We consider an interacting particle system where particles repulse each other and simultaneously also experience an attractive force to drift towards a weighted mean of the particles. The balance of repulsive and attractive behavior results in exploration and exploitation, respectively. Consider the following SDE driving N particles:

$$\begin{aligned} dX^{i,N}(t) = &-\eta(t)(X^{i,N}(t) - \bar{X}^N(t))dt \\ &+ \frac{\lambda(t)}{N}\sum_{j=1}^{N}(X^{i,N}(t) - X^{j,N}(t))\exp\Big(-\frac{1}{2}\|X^{i,N}(t) - X^{j,N}(t)\|^2\Big)dt \\ &+ \sigma(t)\,\mathrm{Diag}((X^{i,N}(t) - \bar{X}^N(t)))dB^i(t), \end{aligned} \tag{5.1}$$

where

- $\eta(t)$ is a positive non-decreasing or increasing function of t,
- $\lambda(t)$ is a positive decreasing function of t,
- $\sigma(t)$ can be chosen as a constant $\sigma(t) \equiv \sigma$ or can be a positive decreasing function of t,

and $\bar{X}^N(t) = \frac{\sum_{j=1}^{N} X^{j,N}(t)\omega^{\alpha,f}(X^{j,N}(t))}{\sum_{j=1}^{N}\omega^{\alpha,f}(X^{j,N}(t))}$ with $\omega^{\alpha,f}(x) := e^{-\alpha f(x)}$. Note that the mean is different in this chapter compared to the other chapters in that it is weighted by the function value. The Diag operator creates a zero matrix except for the diagonal, which will be the vector $X^{i,N}(t) - \bar{X}^N(t)$. B^i refers to Brownian motion affecting particle i. This dynamic is a variant of CBO-model [21].

From the dynamics in (5.1), it can be seen that it does not depend on any gradient evaluation to find the global minimum. Instead, each particle moves in the direction

leading to the instantaneous weighted mean. Here the mean is weighted by the function value, meaning that particles closer to the global minimum will have a greater effect on the mean-value.

There are a few different ways in which the dynamics explores. Let us start by looking at the first term which is $\eta(t)(X^{i,N}(t) - \bar{X}^N(t))$. This term by itself does not contribute significantly when it comes to exploration, instead, it is what allows the particles to converge towards the weighted mean. It does this by moving in the direction leading directly to the current weighted mean. Note however, that some exploration is being performed while the particles are pathing towards the mean. However, the probability of convergence towards the global minimum by only using this term is extremely low. This will also require the particles to be initialized in the vicinity of the global minimum, which might not always be the case.

The majority of the exploration is instead performed by the other two terms, the first being $\frac{\lambda(t)}{N}\sum_{j=1}^{N}(X^{i,N}(t) - X^{j,N}(t))\exp\left(-\frac{1}{2}\|X^{i,N}(t) - X^{j,N}(t)\|^2\right)$. The idea is that each particle has a repelling force which is proportional to the distance between the particles. Meaning that particles which are close to each other will move further away from each other than those which are already far apart. The magnitude of the force is controlled by λ, with larger values repelling more.

The other term $\sigma(t)\,\mathrm{Diag}((X^{i,N}(t) - \bar{X}^N(t)))dB^i(t)$ introduces noise into the system. Normally with CBO, $\lambda = 0$, meaning that this term contributes the most to exploration. In this case, however, the dynamics already contain exploration, which might make this term less valuable. Notice also that the noise in this case is scaled by a matrix depending on position of each particle and the weighted mean. Also, $\sigma(t)$ is a parameter controlling the magnitude of the noise, with larger values meaning larger noise.

Understanding the purpose of each term in the dynamics is an important part in deciding the parameter values such that optimal results can be achieved. Having a large η would mean that the dynamics converge faster, but it might also lead to the convergence dominating the exploration, which can hurt the results. Having large λ and σ might also be useful, so that the system explores. However, too much exploration can lead to a scenario where the algorithm can have issues with convergence, which also hurts the results. It is therefore important to find a balance with the parameters such that sufficient exploration is performed in early stages and then noise/repulsion decrease with time so that the method converges towards the end.

5.2 Implementation

To perform simulations on the dynamics, discretization is needed. For this the Euler-Maruyama scheme will be used. Time is evenly partitioned in a time interval $[0, T]$ such that $0 = t_0 < t_1 < ... < t_n = T$, with h being the discretization step defined as $h = t_k - t_{k-1}$.

Let the initial positions of the particles be $X_0^{i,N} = X^{i,N}(0)$. Let us denote $\eta_k = \eta(t_k)$,

$\lambda_k = \lambda(t_k)$ and $\sigma_k = \sigma(t_k)$. Now, we write the Euler-Maruyama scheme

$$\begin{aligned} dX_{k+1}^{i,N} = &-\eta_k(X_k^{i,N} - \bar{X}_k^N)h \\ &+ \frac{\lambda_k}{N}\sum_{j=1}^{N}(X_k^{i,N} - X_k^{j,N})\exp\Big(-\frac{1}{2}\|X_k^{i,N} - X_k^{j,N}\|^2\Big)h \\ &+ \sigma_k \operatorname{Diag}(X_k^{i,N} - \bar{X}_k^N)\xi_{k+1}^i h^{\frac{1}{2}}, \end{aligned}$$

where $\xi_{k+1} \sim \mathcal{N}(0, I_d)$, $X^{i,N}(0) = X_0^{i,N}$, and

$$\bar{X}_k^N = \frac{\sum_{j=1}^N X_k^{j,N} e^{-\alpha f(X_k^{j,N})}}{\sum_{j=1}^N e^{-\alpha f(X_k^{j,N})}}. \tag{5.2}$$

Using the discretization one can construct the optimization Algorithm 3.

Note that when running the simulations, it is done using matrices for computational efficiency. This will mean that in some cases, subtractions are performed between a matrix and a vector. In those cases, what is happening is that each column of the matrix is subtracted by the vector. We will denote this subtraction operation with $\ominus$ in Algorithm 3.

The code used to generate the results can also be found at GitHub.

Algorithm 3 CBO-R.

Initialize h, N, T
$n \leftarrow T/h$
$X \leftarrow X_0 \in \mathbb{R}^{d\text{x}N}$ ▷ Note, every particle computation is performed in parallel
for $k = 1, 2, ..., n$ **do**
  $A \leftarrow -\eta_k(X_k^N \ominus \bar{X}_k^N)$
  $B \leftarrow \frac{\lambda_k}{N}\Sigma_{j=1}^N\Big[(X_k^N \ominus X_k^{j,N})\exp\Big(-\frac{1}{2}\|X_k^N \ominus X_k^{j,N}\|^2\Big)\Big]$ ▷ Note that this summation operation can be done without for loop over j
  $\xi_{k+1} \leftarrow \mathcal{N}(0, I_{d\times N})$
  $C \leftarrow \sigma_k \operatorname{Diag}((X_k^N \ominus \bar{X}_k^N))\xi_{k+1}$
  $X_{k+1}^N \leftarrow X_k^N + Ah + Bh + Ch^{\frac{1}{2}}$
end for

5.3 Numerical illustrations

To measure the performance of the method, the same approach used in Section 4.3 is employed. This means that the success-rate of the method will be tested on different test functions. The criterion for success is: if all dimensions of the weighted mean of the particles are at a distance less than 0.2 from the global minimum at final time T. The weighted mean is calculated following (3.11). For this chapter, the Rastrigin function and the Rosenbrock function will be tested and each test consists of 1000 runs.

CBO is different from the other methods by not being gradient based and therefore there is less issue with exploding trajectories and larger time-steps should therefore be considered. Thus for the simulations performed in this chapter, the time $T = 10$ is used with time-step $h = 0.01$. This ensures that the number of calculations are similar to that of previous chapters. The number of particles will remain at $N = 15$.

What remains is figuring out how the parameters should change with time. From previous chapters, we gathered that constant magnitude of the noise until a certain cut-off time performed well. It will therefore be kept in this chapter. The $\sigma(t)$ that is used can be described as

$$\sigma(t) = \begin{cases} 2, & \text{if } t < 0.9T, \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

Looking at a CBO model without repulsive forces suggests that having a constant $\eta(t)$ produces good results. While the methods are not quite the same, initial tests with a constant η did produce good results. It will therefore also be constant for the remaining tests.

For $\lambda(t)$, inspiration from $\sigma(t)$ is taken. It will therefore be kept constant until a certain time in which it will be set to zero:

$$\lambda(t) = \begin{cases} \lambda, & \text{if } t < \kappa T, \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

Rastrigin function : Normally gradient based methods have issues with this function, since the particles get stuck in the surrounding local minima. However, since CBO is not based on gradients but rather the weighted mean, this should be less of an issue. To demonstrate this, let us look at a contour plot of how the system converges without the presence of any noise terms, see Figure 5.1.

Here one can see that the particles are initially moving towards the best performing particle, which in this case is the beige particle. However, as the particles are pathing, they themselves are exploring search space, which changes the weighted mean. This leads to new best values being found resulting in a change of direction for the particles, this repeats until eventually the global minimum is found. In the case of the Rastrigin function, this type of algorithm which is not based on the gradient seems to be favorable, initial results also seem to confirm this, see Table 5.1.

Comparing these results to the first or second order ensemble Langevin dynamics, it is clear that this method performs better. Note that the highest success-rate is achieved when the repulsive term is present (i.e. $\lambda > 0$).

Due to the method almost reaching 100% success-rate it is tested for a more difficult scenario. Instead of having the particles be initialized from $[-3, 3]$, they are instead initialized from $[-8, -2]$. This means that the particles are no longer surrounding the global minimum. This then puts a bigger emphasis on exploration, which might favor the inclusion of the new λ term.

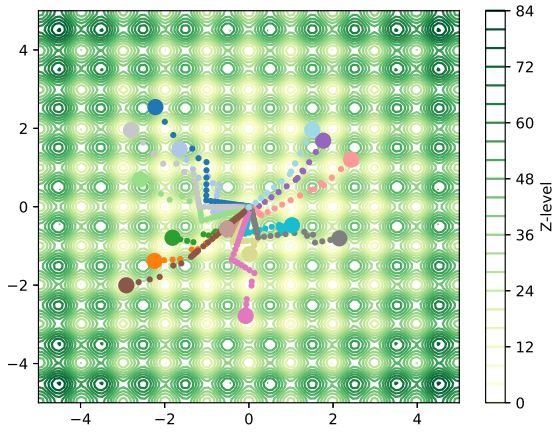


Figure 5.1: Contour plot of CBO (5.1) on the Rastrigin function (2.14) with $N = 15$, $\sigma = 0$, $\lambda = 0$ and $\eta = 10$. The initialized particles can be seen as large colored dots.

Table 5.1: Success rates for finding the global minimum on the Rastrigin function (2.14) using CBO (5.2) with σ from (5.3) and $\eta = 0.1$ for particles initialized randomly between $[-3, 3]^2$.

κ \ λ	0	1	10	100
0.9	96.8 %	99.1 %	48.3 %	11.4 %
0.7	95.9 %	99.2 %	97.2 %	92.0 %
0.5	97.0 %	99.3 %	98.8 %	96.5 %
0.3	97.1 %	98.7 %	98.3 %	97.6 %

See Table 5.2 for results. Here it is clear that the success-rate is significantly higher when the repulsion term is present.

Table 5.2: Success rates for finding the global minimum on the Rastrigin function (2.14) using CBO (5.2) with σ from (5.3) and $\eta = 0.05$ for particles initialized randomly between $[-8, -2]^2$.

α \ λ	0	25	50	100
0.9	1.0 %	18.6 %	9.4 %	6.1 %
0.7	0.5 %	57.4 %	52.3 %	48.8 %
0.5	0.9 %	60.9 %	55.6 %	51.3 %
0.3	0.6 %	59.0 %	56.5 %	51.9 %

Note that the results are worse compared to when the particles are initialized around the global minimum, this is to be expected since the average starting distance from the global minimum is larger. Further testing is then performed to see if the results can be optimized. While the best performing results remained similar, what did change was the importance of the repelling term, see Table 5.3. As can be seen, the results showcase that the performance with or without the repelling term is almost identical.

Table 5.3: Success rates for finding the global minimum on the Rastrigin function (2.14) using CBO (5.2) with σ from (5.3) with $\sigma = 10$ instead of 2 and $\eta = 1$ for particles initialized randomly between $[-8, -2]^2$.

α \ λ	0	25	50	100
0.9	65.9 %	61.4 %	34.9 %	17.8 %
0.7	66.1 %	65.7 %	62.2 %	59.8 %
0.5	66.1 %	65.5 %	64.9 %	64.0 %
0.3	66.2 %	65.6 %	65.6 %	65.1 %

One can also see a clear difference in how the different noise levels approach exploration. The left figure in Figure 5.2 showcases using larger λ with smaller σ. As can be seen, the particles slowly spread out from each other until eventually finding the global minimum. By instead using larger σ as can be seen in the right figure, larger jumps are being made and the exploration is instead being done around the currently best performing particle until the global minimum is found.

Both of these approaches have their benefits and pitfalls. Slowly spreading out yields a safe approach to exploring large parts of the parameter space. However, it takes time in doing so, especially if the initialization is far from the global minimum. Instead, taking larger steps and exploring from the best performing particles can be useful in exploring outside of the initialization quickly, however, if the parameter space has many widely spread deep local minima it might end up being stuck in one of the minima without finding the global minimum.

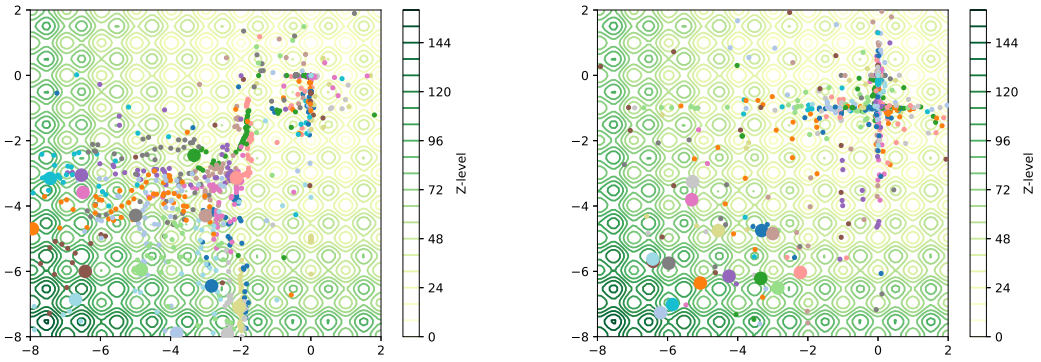


Figure 5.2: Contour plot of CBO (5.1) on the Rastrigin function (2.14) with $N = 15$, $\lambda(t)$ from (5.4) with $\lambda = 25$ and $\kappa = 0.5$. The left figure uses $\sigma = 2$, $\lambda(t)$ from (5.4) and $\eta = 0.05$ and the right figure uses σ from (5.3) with $\sigma = 10$ instead of 2 and $\eta = 1$. The initialized particles can be seen as large colored circles.

To conclude, the repelling term allows for the method to have a greater average performance with less fine-tuning of the model to a certain extent.

Rosenbrock function :

We now test the method on the Rosenbrock function. The difficulty in the Rosenbrock function is due to the bottom of the valley being quite flat. This means that it is easy to move down the side of the valley without finding the global minimum. The advantage in the case of using a gradient based approach is that each move will be towards the global minimum. Since CBO is not gradient based, it might lead to problems. The results also seem to imply this as can be seen in Table 5.4.

Table 5.4: Success rates for finding the global minimum on the Rosenbrock function (2.15) using CBO (5.2) with σ from (5.3) with $\sigma = 1$ and $\eta = 0.5$ for particles initialized randomly between $[-3, 3]^2$.

α \ λ	0	1	10	100
0.9	41.4 %	47.1 %	20.8 %	0.1 %
0.7	37.9 %	53.0 %	49.0 %	0.5 %
0.5	38.9 %	53.4 %	50.5 %	2.5 %
0.3	40.0 %	46.8 %	50.1 %	7.7 %

While the results are not as good as for the Langevin dynamics it still manages to find the global minimum the majority of the time. For this type of function, the importance of the repulsion term is also more clear, here it indicates a significant performance improvement when repulsion is present. Looking at a contour plot of a failed attempt also seems to confirm that the issue is in exploring the bottom of the valley, see Figure 5.3. What can be seen is that all the particles seem to move quite rapidly down the side of the valley, but end up slowing down when approaching the

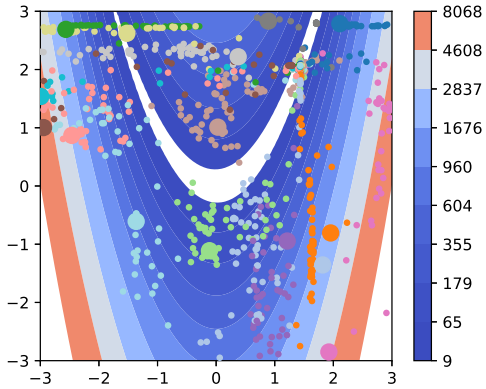


Figure 5.3: Contour plot of CBO (5.1) on the Rosenbrock function (2.15) with $N = 15$, $\sigma = 1$, $\lambda(t)$ from (5.4) with $\lambda = 1$ and $\alpha = 0.5$ and $\eta = 0.05$. The initialized particles can be seen as large colored dots.

bottom. This suggests that not enough exploration is performed at the bottom of the valley.

6

Training Neural Networks

6.1 Introduction

So far, we have tested different SDE based optimization methods on test functions to determine the effectiveness of the methods. We concluded that while the methods were different, they were effective in their rights. The plan for this chapter is to test these SDE based models as optimizers for training neural networks.

The term neural network is very wide and it can mean many different things. For this chapter, we will consider a fully connected feedforward classification model.

So, what is a neural network? Essentially a neural network is a mapping that takes an input and gives an output. An example network can be seen in Figure 6.1.

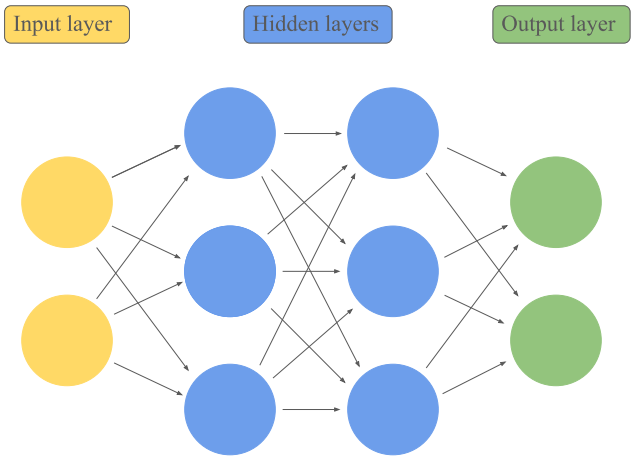


Figure 6.1: An example network with an input of size two, with two hidden layers of size three and an output layer of size two.

Looking at the figure, one sees disks of different colors, these disks are usually referred to as neurons. The color of the disks represents the type of neuron, with yellow being the input neurons, blue ones are the hidden neurons, and green ones represent the output neurons.

The purpose of the neurons is to process signals that are sent from the previous layer before sending them forward toward the next layer. This is done by summing

all of the received signals and also applying a constant parameter called a bias. Here a signal is represented by an arrow. As can be seen in the figure, each neuron has one arrow connection to each following neuron in the next layer. When a layer has this property it is called a fully connected layer.

Each signal is affected by a weight, meaning that the signals can become bigger or smaller depending on the size of the weight. In the illustration, one can imagine the weight as the size of the arrows. Determining the size of the weight is crucial to getting good results from the model, and doing so is typically done by training the model.

After the input signal has moved through every neuron it ends up in the output layer, in which one gets the results. In the case of binary classification the goal is to determine if something belongs to class one or class two. One way of interpreting the results is to give each of the classes one of the output neurons, and whichever neuron gets the higher value is the class the input belongs to.

Here one might realize that currently with the way that the network is explained, it is linear, which is not ideal for all circumstances, to prevent this one usually adds an activation function to each neuron. The activation function alters the processed signal before sending it forward to the next layer. The value of each neuron can now be described as an equation

$$y = \sigma\left(b + \sum_{i=1}^{n} x_i w_i\right), \tag{6.1}$$

where y is the value, σ represents the activation function, b is the bias, n is the number of connected neurons from the previous layer, x_i is the signal from the i-th neuron from the previous layer and w_i is the i-th weight in the current neuron. There exists many different activation functions, each having its benefits, and choosing the right one is essential to achieve good results. Some common activation functions are the ReLU-activation function and the sigmoid function. ReLU is defined as

$$\sigma(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

The function sends forward any positive signals and sets any negative signal to zero. It is very efficient and allows for the network to become non-linear. The sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{6.3}$$

This activation function is especially useful if used on the output layer when doing binary classification. This is due to the function outputting values between 0 and 1. If the model uses a single output neuron, one can then interpret the output as a probability of the input belonging to either of the classes.

Determining the structure of the model, such as which activation functions should be used, how many hidden layers the model should consist of, and what size each of the hidden layers should be, is usually tricky and is something that is either

done through trial and error or making educated guesses following from previous successful models.

In order to make the model such that it makes accurate prediction, one has to determine the weights and biases. The weights and biases are mostly randomized initially and determined through training. It is similar to training any other regression models. One takes a set of data which is known, run it through the network, and then compare the output to the target. Here one constructs a loss function to determine the difference. When choosing a loss function, what is important, is that it gives a large loss when doing a bad prediction and a small loss when doing a good prediction. For binary classification, a common loss function is the binary cross entropy loss, it is described as

$$L = -\left(y \log(p(y)) + (1-y) \log(1-p(y))\right), \tag{6.4}$$

where L is the loss, y is the target and $p(y)$ is the output of the network. In our setting of binary classification, $p(y)$ describes the probability of belonging to each class. For classification, designing a loss function might be difficult, since the output from the network is a probability between 0 and 1, however by using the logarithm, one can achieve large losses when the network is scoring a low probability and small losses as the probability reaches closer to 1.

After calculating the loss, one tweaks the weights and biases such that the error becomes smaller. We already know that gradient descent can be used to find suitable changes for the parameters. The only issue with neural networks is that each neuron affects each coming neuron, which makes calculating gradients a daunting task. Therefore, this chapter aims to test methods that do not necessarily rely on gradient evaluation, and to determine if equal performance can be achieved without having to perform as many difficult computations.

6.2 Simulation

In previous chapters, we used different test function to determine the performance of the optimizers. Since neural networks are a type of regression model, one has to be slightly more careful. Here the capacity of the regression model must be sufficient such that it can, under the right circumstances, learn to classify data. If this is not the case, even the best optimizers would fail to get good results when training the models. To avoid this issue, both the dataset and the architecture of the network are taken from one of the experiments considered in [22]. Note that all the results presented in this section are generated by code that can be found in GitHub.

Stochastic gradient descent : In the paper [22], SGD with a step-size of 0.05 is used to train a neural network to classify whether a particle belongs to class 0 or class 1 depending upon its position. The dataset on which the neural network is trained is a spiral dataset, which contains two classes and is defined by equation (6.5) and (6.6),

$$x = -\sqrt{t}\cos(4\sqrt{t}\pi\theta) + 0.05\mathcal{N}(0,1), \tag{6.5}$$

$$y = -\sqrt{t}\sin(4\sqrt{t}\pi\theta) + 0.05\mathcal{N}(0,1). \tag{6.6}$$

The training and testing datasets are generated by sampling the desired number of data points. Each data point is generated as follows: First, determine the class of the data point by choosing θ. If $\theta = 1$, the data point belong to class 0. If $\theta = 1 + \pi$ the data point belongs to class 1. Next, determine the (x,y) position of the data point by sampling t from a uniform distribution $\mathcal{U}(0,1)$ and inserting it into equation (6.5) and (6.6).

Note that each dataset is unique, due to the random elements when sampling both t and the noise from $0.05\mathcal{N}(0,1)$. Therefore, in this thesis, the same training and testing dataset is used for all of the different methods to ensure a fair comparison.

The architecture of the neural network used is two input neurons with a single fully connected hidden layer of size 500 with the ReLU activation function. The output layer is a single neuron with the sigmoid activation function. To calculate the loss, a binary cross entropy loss function is used. For training, the dataset is split into training and testing data, with 100 data points being used for training and 2000 used for testing. Mini-batches are also utilized with a batch-size of 2. The neural network is created using PyTorch meaning that the weights and biases are initialized as

$$w = \mathcal{U}\Big(-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\Big), \tag{6.7}$$

where $\mathcal{U}(a,b)$ represents the uniform distribution on (a,b) and k is the number of inputs of the neuron. It is then run for 10000 epochs.

In this thesis, the SGD simulations are repeated to use as a benchmark, with the difference being that 5000 epochs are used and that 1900 instead of 2000 points are used for testing. A figure showcasing both the training and testing dataset can be seen in Figure 6.2. Note that the classes are almost balanced, meaning that similar amounts of data points belongs to each of the classes.

Here, the results are clear, showing that SGD can be used to train the neural network to approximately predict the class, see Figure 6.3. Here a training accuracy of 100% is achieved with a test accuracy of 84.5%. Notice, however, that both the test loss and test accuracy performance decrease slightly after 1500 epochs. This might be due to over-fitting and would normally be solved using a stopping criteria or simply saving the best performing parameters for the network. To get a better understanding of how the network classifies each point, one can look at the classification boundaries, see Figure 6.4. These boundaries are generated by classifying a grid of 90000 points between the values 0 and 1 in both directions, coloring the points in different colors depending upon its class. While the boundaries are not necessarily the exact shape of the spiral but one can see a resemblance. Getting significantly better results than this can be challenging with the current training data, since some areas of the spiral are lacking samples.

First order Langevin : Now that the baseline is set, it is compared to first order ensemble Langevin dynamics. It is slightly different from that of equation (4.3), since we have replaced the square root of the covariance term in the diffusion with a time dependent function $\mu(t)$ (see (6.8)). This has been done due to simplicity. This dynamics is inspired from [6].

Let $X^{i,N}$ be the particle representing the weights and biases of the network and let $\mathcal{L}(G(X^{i,N}(t)), y)$ represent the loss function where $y' := G(X^{i,N}(t))$ is the output of the network and y is the target. The new dynamics can be described as

$$dX^{i,N}(t) = C^N(t)\nabla_x \mathcal{L}(G(X^{i,N}(t)), y)dt + \mu(t)dB^i(t), \tag{6.8}$$

where $\mu(t)$ is a non-increasing function of t and gradient is computed with respect to weights and biases. Using the Kalman approximation [7]

$$C^N(t)\nabla_x \mathcal{L}(G(X^{i,N}(t)), y) \approx C_G^N(t)\nabla_{y'} \mathcal{L}(G(X^{i,N}(t)), y), \tag{6.9}$$

we arrive at the following SDE:

$$dX^{i,N}(t) = C_G^N(t)\nabla_{y'} \mathcal{L}(y', y)dt + \mu(t)dB^i(t), \tag{6.10}$$

where

$$C_G^N(t) = \frac{1}{N}\sum_{j=1}^{N}(X^{j,N}(t) - \bar{X}^N(t)) \otimes (G(X^{j,N}(t)) - \bar{G}(t)) \tag{6.11}$$

is empirical cross-covariance, and

$$\bar{X}^N(t) = \frac{1}{N}\sum_{j=1}^{N} X^{j,N}(t), \qquad \bar{G}(t) = \frac{1}{N}\sum_{j=1}^{N} G(X^{j,N}(t)). \tag{6.12}$$

Here, $\otimes$ is the Kronecker product as explained in Section 2.5.

The dynamics can then be rewritten as

$$\begin{aligned} dX^{i,N}(t) = -\frac{1}{N}\sum_{j=1}^{N}\langle G(X^{j,N}(t)) - \bar{G}(t), \nabla_{y'} \mathcal{L}(G(X^{i,N}(t)), y)\rangle X^{j,N}(t)dt \\ + \mu(t)dB^i(t). \end{aligned} \tag{6.13}$$

Here the $\langle \cdot, \cdot \rangle$ is the dot product.

For the simulations, the dynamics then have to be discretized, and for this we use the Euler-Maruyama scheme with initial position $X_0^{i,N} = X^{i,N}(0)$. The discretization is as follow:

$$X_{k+1}^{i,N} = X_k^{i,N} - \frac{1}{N}\sum_{j=1}^{N}\langle G(X_k^{j,N}) - \bar{G}_k, \nabla_{y'} \mathcal{L}(y', y)\rangle X_k^{j,N} h + \mu_k \xi_{k+1}^i \sqrt{h}, \tag{6.14}$$

where $\bar{G}_k = \frac{1}{N}\sum_{j=1}^{N} G(X_k^{j,N})$, $\xi \sim \mathcal{N}(0, 1_D)$, D is the represents the total number of weights and biases parameters, $\mu_k = \frac{1}{1+k0.01}$ and $h = 0.05$.

What makes this dynamics special is that, due to the Kalman approximation, the gradient $\nabla_{y'}$, is a gradient calculated with respect to the output of the network y'. This means that no back-propagation is required for the computations. By inputting the binary cross entropy-loss in equation (6.14), it can be rewritten as

$$X_{k+1}^{i,N} = X_k^{i,N} - \frac{1}{N}\sum_{j=1}^{N}\left\langle G(X_k^{j,N}) - \bar{G}_k, \frac{y - G(X_k^{i,N})}{(G(X_k^{i,N}) - 1)G(X_k^{i,N})}\right\rangle X_k^{j,N} h + \mu_k \xi_{k+1}^i \sqrt{h}. \quad (6.15)$$

For the simulations, the architecture and training configuration is kept the same as that of the SGD with the only difference being the initialization. Here instead the weights are initialized depending on which activation function is used. This means that the sigmoid activation function is kept as

$$w = \mathcal{U}(-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}) \quad (6.16)$$

with k being the number of inputs to the neuron. However, for the ReLU activation function, it is suggested to instead use the initialization from [23]. For that one instead uses a Gaussian distribution as follows:

$$w = \mathcal{G}\left(0, \sqrt{\frac{2}{k}}\right) \quad (6.17)$$

with k being the number of inputs to the neuron. The biases are also not initialized in the same manner, instead, they are all initialized as 0.01, suggested from [24]. For the number of particles, $N = 50$ is used.

Due to the way the method is implemented, using particles, each particle generates one network. This means that one also has to consider how to choose which model to use. The approach we decided on, is to merge the values of the weights and biases using the weighted average of the particles. This means that particles that perform well have a bigger impact, see equation (3.11).

Initial results from the first order method look promising with the performance almost reaching that of the SGD optimizer, see Figure 6.5. Here a training accuracy of 99% is achieved with a test accuracy of 84%. Looking at the decision boundaries, the overall shape seems smoother than that of SGD, see Figure 6.6. This can be due to it having less issue with over-fitting, which can be explained by the test loss getting a significant increase. It does fail in achieving a 100% training accuracy, but the test accuracy is performing similarly to SGD. However, one part of the spiral is assigned to the wrong class. Since the test loss seems to be decreasing slowly, this might be improved upon if it is trained for longer.

Momentum discretization : To see if better results can be achieved, the dynamics is also tested using a different momentum-type discretization, which is taken from [6]. We will also take help from [6] in choosing hyper-parameters. The discretization

is changed as follows:

$$X_{k+1}^{i,N} = Y_k^{i,N} - \frac{1}{N}\sum_{i=1}^{N}\langle G(X_k^{i,N}) - \bar{G}_k, \nabla_{y'}\mathcal{L}(y', y)\rangle Y_k^{i,N} h_k, \tag{6.18}$$

$$Y_{k+1}^{i,N} = X_{k+1}^{i,N} + \lambda(X_{k+1}^{i,N} - X_k^{i,N}) + \mu\xi_{k+1}^{i}, \tag{6.19}$$

where $\xi_{k+1}^{i} \sim \mathcal{N}(0, I_D)$, $Y_0^{i,N} = X_0^{i,N}$, $\lambda \in (0, 1)$, $\mu > 0$ and

$$h_k = \frac{h_0}{(||\langle G(X_k^{i,N}) - \bar{G}_k, \nabla_{y'}\mathcal{L}(y', y)\rangle|| + \epsilon)}.$$

The inclusion of the new h_k is to maximize the step-size without having issues of terms exploding. For the simulations, empirical evidence suggested $h_0 = 10^{-9}$, $\epsilon = 10^{-8}$, $\lambda = 0.7$ and $\mu = 10^{-2}$ to be suitable.

With these changes, the results do see an improvement. By looking at the accuracy, the training accuracy now manages to reach 100%, see Figure 6.7. The test accuracy also increased to 87%, meaning that it outperforms SGD. Looking at the decision boundaries it almost mimics the real spirals, only missing a small part, see Figure 6.8.

While a more extensive study would be needed to be certain, the results clearly indicate the feasibility of training neural networks using optimization algorithms which do not make use of back-propagation. Many improvements can be made, not only in the choice of parameters but also in the methods themselves. Regarding the speed of the program, currently, SGD is faster when using PyTorch. The main reason for this is most likely that the implementation of the neural network, as well as the training algorithm is not as optimized as the PyTorch framework. Another reason for the slower speed could be that only a single hidden layer is being used. The complexity of computing the gradient increases as the number of layers increase. This means that this method might be comparable in speed when the complexity of the model increases.

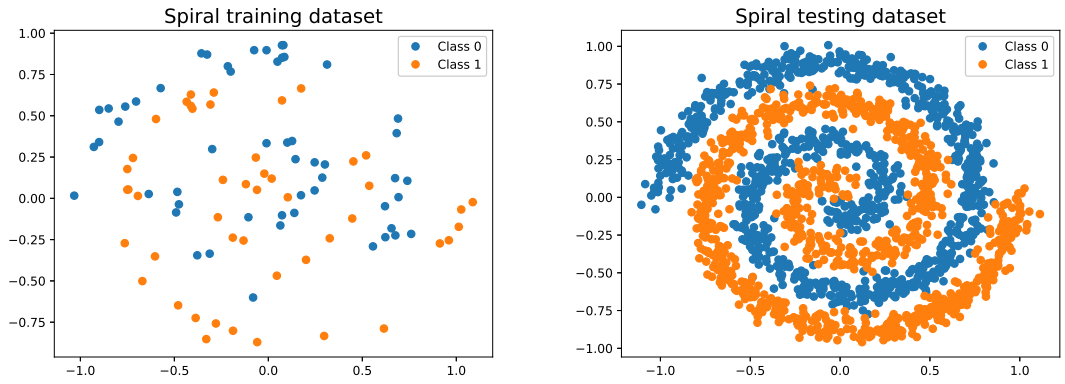


Figure 6.2: Training and testing data of the spiral dataset generated from (6.5) and (6.6). The left figure showcases 100 training data-points and the right figure showcases 1900 testing data-points.

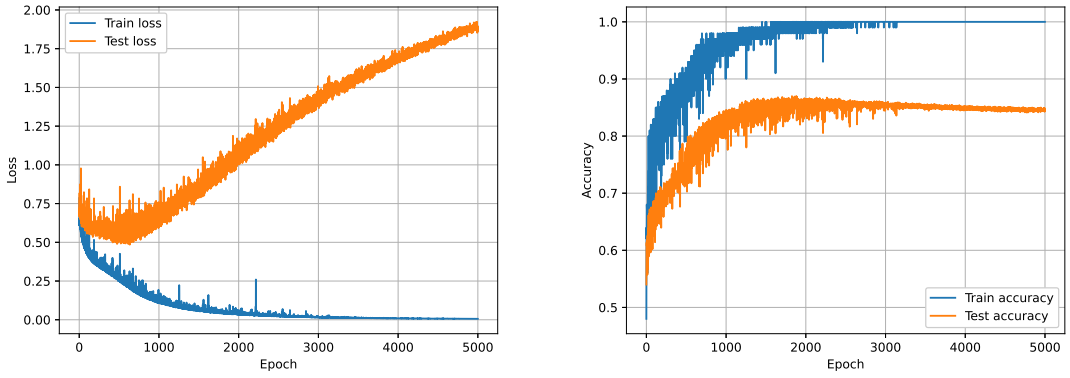


Figure 6.3: Loss and accuracy of both the training and testing data with SGD optimizer over 5000 epochs.

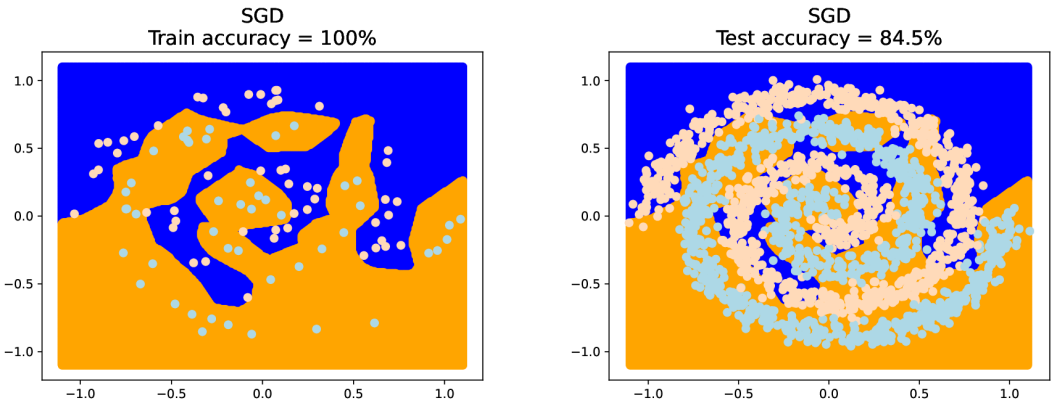


Figure 6.4: Training and testing data plotted over the decision boundaries of the SGD model.

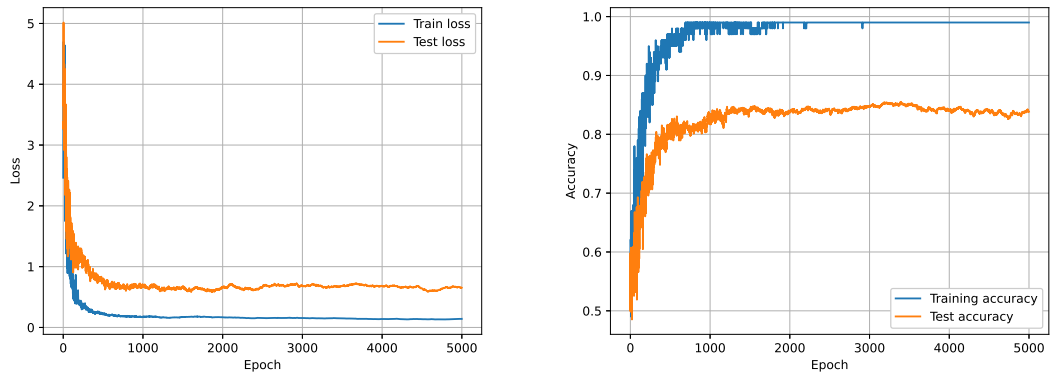


Figure 6.5: Loss and accuracy of both the training and testing data with first order ensemble Langevin optimizer (6.15) over 5000 epochs.

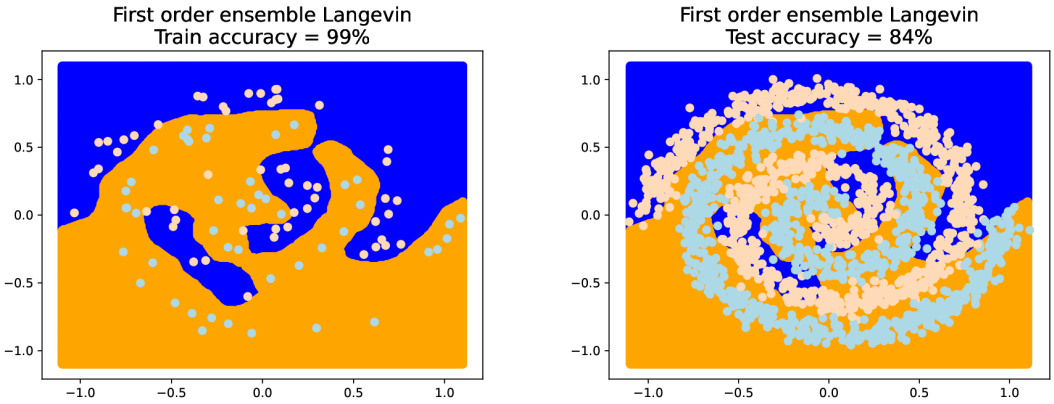


Figure 6.6: Training and testing data plotted over the decision boundaries of the first order ensemble Langevin optimizer (6.15).

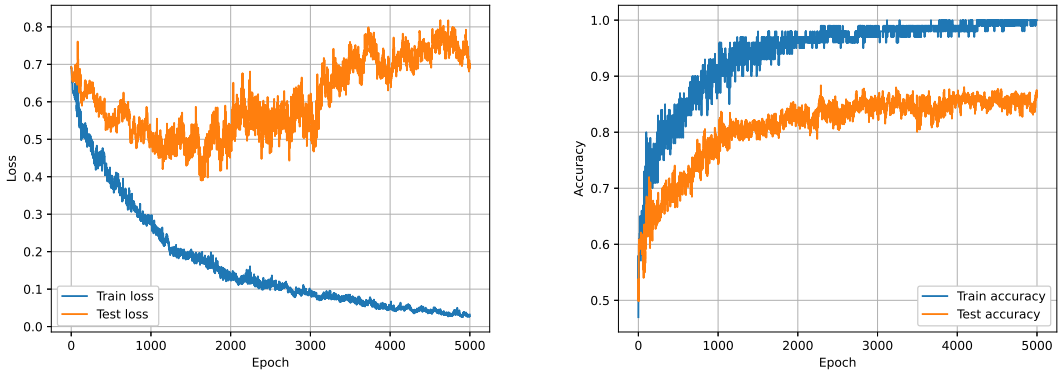


Figure 6.7: Loss and accuracy of both the training and testing data with momentum based discretization of ensemble Langevin optimizer 6.19 over 5000 epochs.

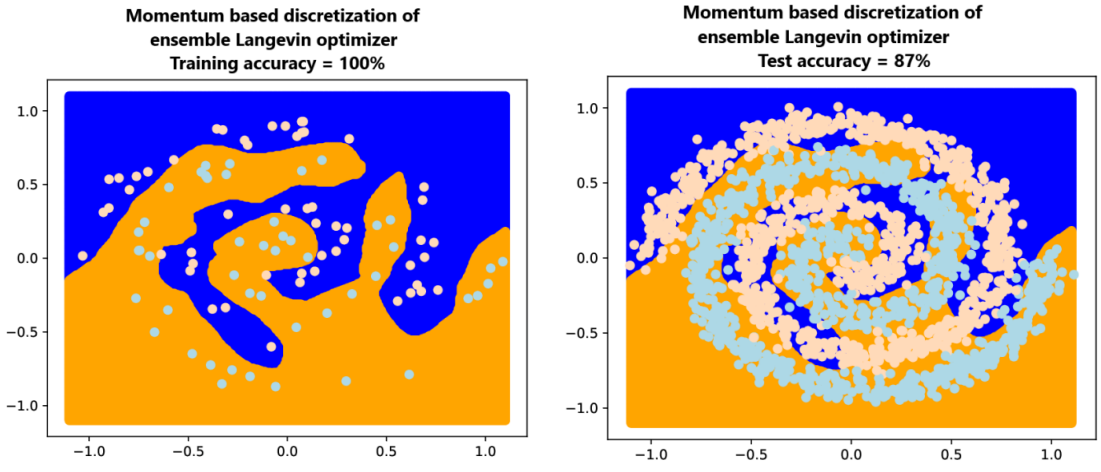


Figure 6.8: Training and testing data plotted over the decision boundaries of the momentum based discretization of ensemble Langevin optimizer 6.19.

7

Summary and conclusions

Having tested both ensemble Langevin dynamics and ensemble kinetic Langevin dynamics, it is clear that the methods can be used not only for sampling but also for optimization. The results also showcase that the methods are flexible, performing well when testing on different types of benchmarking functions. One issue with the methods is that they are quite unstable with respect to the choice of time-step. This, however, can be improved upon by implementing taming.

Comparing the different ensemble Langevin dynamics also showcases that each method has its benefits and neither of the methods outperforms the other in all cases. For situations in which long distances need to be traversed with a small incline, using kinetics is superior due to the added benefit of momentum. However, when dealing with many different local minima, foregoing the momentum seems better.

In regards to consensus based optimization (CBO) with repulsion, it undoubtedly performs well as an optimizer, especially for the Rastrigin function, for which it outperforms both of the ensemble Langevin methods. Including repulsion seems to be beneficial for two reasons. It broadens the parameter choices for achieving good results, which makes it easier to implement with fewer issues of fine tuning. It also seems helpful in cases where functions have areas in which the values are similar, such as the Rosenbrock function. This is promising since the performance of the CBO without the repulsion on the Rosenbrock function is poor. Note, however that on the Rosenbrock function, both methods based on ensemble Langevin dynamics outperforms CBO, even when the latter has a repulsive term.

For the neural networks, derivative free ensemble Langevin dynamics performs very well, achieving similar performance to stochastic gradient descent. What makes this method interesting is that back-propagation can be avoided meaning that it only needs forward passes. This should allow for better computational performance. However, this effect is not observed in the numerical experiments which we run here. There could be several reasons for this: the implementation of the ensemble Langevin dynamics is not optimized for speed, and the neural network used in the test case is relatively small. Further testing on this would be interesting to see if the method would be more computationally efficient, especially as the networks grow in size and the gradients become more complicated to calculate.

Bibliography

[1] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4.

[2] R. Pinnau, C. Totzeck, O. Tse, and S. Martin. "A consensus-based model for global optimization and its mean-field limit". In: *Mathematical Models and Methods in Applied Sciences* 27.01 (2017), pp. 183–204.

[3] J. A. Carrillo, Y.-P. Choi, C. Totzeck, and O. Tse. "An analytical framework for consensus-based global optimization method". In: *Mathematical Models and Methods in Applied Sciences* 28.06 (2018), pp. 1037–1066.

[4] D. Kalise, A. Sharma, and M. V. Tretyakov. "Consensus-based optimization via jump-diffusion stochastic differential equations". In: *Mathematical Models and Methods in Applied Sciences* 33.02 (Feb. 2023), pp. 289–339. ISSN: 1793-6314.

[5] M. A. Iglesias, K. J. Law, and A. M. Stuart. "Ensemble Kalman methods for inverse problems". In: *Inverse Problems* 29.4 (2013), p. 045001.

[6] N. Kovachki and A. Stuart. "Ensemble Kalman inversion: a derivative-free technique for machine learning tasks". In: *Inverse Problems* 35.9 (Aug. 2019), p. 095005.

[7] A. Garbuno-Inigo, F. Hoffmann, W. Li, and A. M. Stuart. "Interacting Langevin diffusions: Gradient structure and ensemble Kalman sampler". In: *SIAM Journal on Applied Dynamical Systems* 19.1 (2020), pp. 412–441.

[8] N. Parikh and S. Boyd. "Proximal Algorithms". In: *Foundations and Trends® in Optimization* 1.3 (2014), pp. 127–239. ISSN: 2167-3888.

[9] D. H. Ackley. "A Connectionist Machine for Genetic Hillclimbing". In: *The Kluwer International Series in Engineering and Computer Science* (1987).

[10] B. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553.

[11] X. Mao. *Stochastic differential equations and applications*. Elsevier, 2007.

[12] G. N. Milstein and M. V. Tretyakov. *Stochastic numerics for mathematical physics*. Vol. 39. Springer.

[13] P. J. Rossky, J. D. Doll, and H. L. Friedman. "Brownian dynamics as smart Monte Carlo simulation". In: *The Journal of Chemical Physics* 69.10 (1978), pp. 4628–4633.

[14] T.-S. Chiang, C.-R. Hwang, and S. J. Sheu. “Diffusion for global optimization in $\mathbb{R}^n$”. In: *SIAM Journal on Control and Optimization* 25.3 (1987), pp. 737–753.

[15] A. Garbuno-Inigo, N. Nüsken, and S. Reich. “Affine invariant interacting Langevin dynamics for Bayesian inference”. In: *SIAM Journal on Applied Dynamical Systems* 19.3 (2020), pp. 1633–1658.

[16] M. Hutzenthaler, A. Jentzen, and P. E. Kloeden. “Strong and weak divergence in finite time of Euler’s method for stochastic differential equations with non-globally Lipschitz continuous coefficients”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 467.2130 (2011), pp. 1563–1576.

[17] M. Hutzenthaler, A. Jentzen, and P. E. Kloeden. “Strong convergence of an explicit numerical method for SDEs with nonglobally Lipschitz continuous coefficients”. In: *The Annals of Applied Probability* 22.4 (2012), pp. 1611–1641.

[18] M. V. Tretyakov and Z. Zhang. “A fundamental mean-square convergence theorem for SDEs with locally Lipschitz coefficients and its applications”. In: *SIAM Journal on Numerical Analysis* 51.6 (2013), pp. 3135–3162.

[19] G. dos Reis, S. Engelhardt, and G. Smith. “Simulation of McKean–Vlasov SDEs with super-linear growth”. In: *IMA Journal of Numerical Analysis* 42.1 (2021), pp. 874–922.

[20] Z. Liu, A. M. Stuart, and Y. Wang. “Second order ensemble Langevin method for sampling and inverse problems”. In: *preprint arXiv:2208.04506* (2022).

[21] J. A. Carrillo, S. Jin, L. Li, and Y. Zhu. “A consensus-based global optimization method for high dimensional machine learning problems”. In: *ESAIM: Control, Optimisation and Calculus of Variations* 27 (2021), S5.

[22] B. Leimkuhler, T. J. Vlaar, T. Pouchon, and A. Storkey. “Better Training using Weight-Constrained Stochastic Dynamics”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 6200–6211.

[23] J. Brownlee. “Weight Initialization for Deep Learning Neural Networks”. In: *Machine Learning Mastery* (2021).

[24] A. Sahai. *Lecture 6*. `https://inst.eecs.berkeley.edu/~cs182/sp23/assets/notes_new/lec6.pdf`. 2023.

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY