THE UNIVERSITY *of* EDINBURGH

Higher-Order Damping Mechanisms with Applications in Optimisation and Machine Learning

Aikaterini Karoni

Doctor of Philosophy

The University of Edinburgh

2024

Για την

Μαρία και τον Ηλία

Abstract

The main focus of this thesis is the development of new optimisation methods. The design of our methods is guided by a dynamical systems perspective and utilises concepts commonly used in molecular dynamics such as thermal controls, which regulate the temperature (i.e. the mean kinetic energy) of the system. This idea, of introducing a feedback loop based on the integral of the squared speed, is rooted in the work of James Clerk Maxwell [84] on governors and control theory. In our work, the mechanism by which we influence the physical system is dissipation, controlled by an adaptive friction variable; this is tantamount to varying the so-called "momentum" hyperparameter in optimisation algorithms. Increasing the friction has the efffect of cooling the system more rapidly.

Theoretical convergence guarantees are provided for both the continuous and discrete optimiser dynamics. We test our family of optimisers on a two-dimensional Rosenbrock function, which is a "stiff" landscape consisting of a high-gradient region and a narrow valley, making convergence of standard methods such as momentum gradient descent challenging. We then turn our attention to particle cluster optimisation problems arising in molecular dynamics where we are able to show that our methods outperform momentum gradient descent for most parameters. We show that adapting the friction parameter based on the integral of the kinetic energy is essentially equivalent to adding a cubic damping term to the standard equations of motion of linearly damped Hamiltonian dynamics.

This last observation further motivates the introduction of two new optimisers, presented in Chapter 4. We take the continuous dynamics of two popular optimisation methods, Adam [17] and mSGD [122] and augment them by adding a cubic damping term to the momentum equation for each optimiser. Momemtum Stochastic Gradient Descent (mSGD) is an alternative to Stochastic Gradient Descent which is adapted to a kinetic framework (based on a Hamiltonian system formulation with positions and momenta), and includes a linear dissipation mechanism in the momenta which forces convergence to a minimizer. Adam [61] uses the same dissipation mechanism as mSGD but further includes an adaptive step size mechanism designed to accelerate convergence in low curvature regions by adapting the step depending on a recent history of the squares of the gradient components. We provide theoretical convergence guarantees for our methods and show that introducing a cubic damping mechanism into the dynamics of Adam and mSGD can lead to improved performance on several standard machine learning benchmarks.

The final part of this thesis concerns itself with continual learning mechanisms. The ability of a model to learn in an incremental fashion is key for many real-world applications. Continual learning refers to a setting where a model is updated as new data becomes available, while preserving knowledge acquired from previous tasks. The challenge one faces in such a scenario is to maintain old knowledge, even if old data is no longer available, or retraining on it is not computationally viable. Elastic weight consolidation (EWC) [62] is a popular continual learning method that uses a regularisation approach in order to constrain the change of model parameters that are deemed important for preserving previous knowledge. EWC uses the Fisher information matrix (FIM) to quantify the importance of each of the model parameters and then uses the corresponding Fisher information matrix entry as a regularisation coefficient for each model parameter. The work presented here draws inspiration from EWC and the use of the Fisher information matrix to quantify parameter importance. However, instead of incrementing the loss function by a FIM-based regularisation penalty, the idea behind the method presented is to update each of the model parameters using a different learning rate, which is inversely proportional to the respective FIM value. Thus, parameters that correspond to a higher FIM value and are therefore more important, are updated at a slower rate than parameters with a lower Fisher value.

Lay Summary

This thesis focuses on optimisation methods. Optimisation in mathematics refers to the challenge of finding a minimum/maximum of a given function. When dealing with complex high-dimensional functions, finding an optimum analytically is in most cases impossible. Instead, the search for an optimum is performed computationally through the use of optimisation algorithms. Many of these algorithms require information about the gradient (slope) of the function being optimised. The basic idea behind the simplest methods is to then take sufficiently small steps in the direction of the gradient.

High-dimensional optimisation is a particularly challenging problem for multiple reasons. Gradient computation in high dimensions can be particularly expensive and ill-conditioned regions as well as multi-modality issues are usually much more prevalent. Some examples of applications where these problems may arise, are problems in molecular dynamics, such as particle cluster optimisation as well as neural network optimisation in machine learning. Neural networks have recently found wide applicability on various tasks, from machine translation, image recognition, autonomomous driving and image and video generation to protein folding and drug discovery. Neural networks are high-dimensional functions and their optimisation involves finding a set of parameters that makes them perform sufficiently well on a given task. In this work we develop methods for training neural networks and examine mechanisms that can speed up the training process of such models.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Ben Leimkuhler for his invaluable guidance and support throughout my PhD. I am profoundly grateful to him for being my mentor, for his encouragement and for all the opportunities he has given me. His insights, advice, and discussions have been instrumental in helping me progress in my research and find my way as an applied mathematician.

I would also like to thank Prof. Gabriel Stoltz for all his guidance and advice both during our remote collaboration, as well as during the time he hosted me at Ecole des Ponts. I want to thank him for being a great co-mentor and for introducing me to Lyapunov convergence proofs.

I also want to thank Dr Toni Mey and Dr James McDonagh for being great co-supervisors and sharing their knowledge and expertise on molecular dynamics problems.

Finally, I would like to thank my friends and office mates Aidan, Andrew, Martin, Mike, Pete, and Yasmin for making my time in the office and in Edinburgh so enjoyable.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Aikaterini Karoni

Overview

Optimisation is a pervasive field with applications in disciplines ranging from statistics, machine learning and artificial intelligence, to engineering, finance and operational research. This thesis introduces a family of new optimisation methods and demonstrates their potential on various machine learning benchmarks as well as on applications such as cluster optimisation in molecular dynamics. The dissertation is organised into six chapters.

- **Chapter 1** presents the basic ideas and techniques used in gradient-based optimisation. We start with gradient descent, the simplest gradient-based optimisation scheme and also introduce the concept of momentum as a mechanism to speed up and stabilise the dynamics of gradient descent. We then proceed to talk about discretisation methods, discuss the notion of Lyapunov stability and convergence of continuous and discrete dynamics as well as the key assumptions commonly made in the process of proving convergence. We demonstrate how to use Lyapunov functions to show convergence of the continuous and discretised dynamics of gradient descent and momentum gradient descent.

- **Chapter 2** discusses the use and applications of optimisation methods in the context of machine learning and neural networks. We discuss how "big data" necessitates the use of *stochastic gradients* and how stochastic gradients affect the convergence proof process.

- **Chapter 3** builds on a paper that was written during the PhD [58]. We introduce a family of optimisation algorithms designed from a dynamical systems perspective. Our family of methods uses a momentum mechanism, similarly to momentum gradient descent. Unlike momentum gradient descent however, the momentum coefficient in our methods is not fixed but adaptive and regulated based on the kinetic energy and/or gradients of the system. We demonstrate that our methods outperform momentum gradient descent and are very efficient at controlling oscillations in challenging optimisation landscapes as well as on cluster optimisation problems arising in molecular dynamics.

- **Chapter 4** is based on some work in progress which will soon be submitted to a journal. We introduce three new optimisers based on the ideas presented in Chapter 3. We examine their dynamical properties and demonstrate their potential on a variety of standard machine learning benchmarks.

- **Chapter 5** discusses Continual Learning methods. Continual learning refers to a learning setting where a model needs to be continually updated as new data comes in. In such a scenario, the challenge lies in managing to update the model to fit new data, whilst maintaining model performance on past datasets. We propose a preconditioned learning rates approach where each of the model parameters is trained with a different learning rate based on an importance metric that quantifies how important each parameter is for a model pretrained on a given dataset. We show convergence of our algorithm and test it on two standard continual learning benchmarks.

- **Chapter 6** contains a summary of the main results of the thesis as well as a discussion of possible future directions of research.

A main categorisation of optimisation methods is based on whether the variables we are optimising over are discrete or continuous. In discrete optimisation problems [102], constraints on the variables x_i could have the form $x_i \in \mathbb{Z}$ or could be binary constraints of the form $x_i \in \{0, 1\}$. These types of problems are called "integer programming" problems. The distinguishing trait of discrete optimisation problems is that the variables x_i are drawn from a finite set [96]. In continuous optimisation the variables x_i are usually drawn from a set that is uncountably infinite.

Another important distinction to be made is between local and global [48] optimisation methods. The goal of local optimisation methods is to find a local minimum, which is a point where the function being optimised has the lowest value locally, i.e. in a neighbourhood around the local minimum. Global optimisation methods on the other hand seek to find a global solution, which is a point with the lowest possible function value among all other points [96]. Finally, optimisation methods can also be categorised as stochastic or deterministic. Stochastic optimisation [124] uses randomness to explore the solution space. An example and subcategory of stochastic optimisation methods are genetic algorithms, a subcategory inspired from biological evolution that involves steps such as selection, crossover and mutation [37]. Another way that randomness can be brought into the optimisation process is in the form of stochastic gradients, where a gradient sum is appproximated by random sampling rather than full computation. This type of approach is commonly used in a Bayesian 'big data' analytics setting where the size of the data set precludes full calculation of the likelihood function and the gradient of its logarithm (see Section 2).

In this thesis we focus on continuous local optimisation problems arising in machine learning, molecular dynamics and standard optimisation scenarios, including the optimisation of benchmark functions such as the Rosenbrock function. In the context of machine learning applications we operate within a supervised learning setting, where datasets consist of input-label pairs. In contrast, unsupervised learning deals with datasets that lack labeled outputs,

and the task is to discover patterns within the data without supervision (labels). The models we will use are neural networks, which are compositional functions with trainable parameters that take an input (training features) and produce an output (prediction). The goal of training is to find the set of parameters for the model that lead to the “best fit” on the data.

The present work combines mathematical analysis with numerical study of optimisation methods, while trying to present concepts which are relevant for the deep learning community. Although we purely focus on optimization and do not directly condider sampling applications, sampling and optimisation have been shown to be closely related [80, 146] and acceleration mechanisms from optimization can also enhance sampling methods [80], and vice versa. Exploring this interplay could lead to the development of fast, reliable, and interpretable optimisers and samplers for machine learning problems. Although not within the scope of this thesis, examining the connection between sampling and optimisation methods would be an interesting future direction and could lead to the development of better samplers and optimisers.

Contents

Abstract iii

Lay Summary v

Acknowledgements vi

Declaration vii

Overview viii

1 Preliminaries 1

1.1 Optimization Methods: A dynamical systems perspective 1

1.1.1 First-Order Optimisation Methods 2

Gradient Descent 2

Momentum Gradient Descent 3

AdaGrad 4

RMSProp 4

Adam 5

Nesterov Accelerated Gradient 5

1.1.2 Second-Order Optimisation Methods 6

Newton's Method 6

Quasi-Newton Methods 7

BFGS 7

1.1.3 Difference and Differential Equations 9

Forward Euler method 10

Backward Euler method 10

Splitting schemes 11

A continuous formulation of Momentum Gradient Descent 11

Continuous dynamics formulation of Adam 13

1.2 Lyapunov Functions and Convergence of Optimization Algorithms 14

Common assumptions for proving convergence 15

1.2.1 Convergence of Gradient Descent 16

1.2.2 Convergence of momentum Gradient Descent 18

2 Applications of Optimisation in Machine Learning 21

2.1 Machine Learning 21

2.1.1 Supervised Learning 21

2.1.2 Neural Networks . . . 22
Activation functions . . . 24
Loss functions . . . 26
Backpropagation . . . 28
Convolutional Neural Networks . . . 29
Residual Neural Networks . . . 30
Transformers . . . 31
Stochastic gradients . . . 31

3 Friction-adaptive Descent: A Family of Dynamics-based Optimisation Methods 33
3.1 Introduction . . . 33
3.2 Friction-adaptive descent (FAD) . . . 39
3.2.1 Kinetic friction-adaptive descent (KFAD) . . . 40
3.2.2 Force-based friction-adaptive descent (FFAD) . . . 40
3.2.3 Projective mixture coupling . . . 43
3.3 Dynamical systems analysis of friction-adaptive descent . . . 44
3.3.1 Elementary properties of friction-adaptive dynamics . . . 44
3.3.2 Exponential convergence of the FAD dynamics . . . 46
3.4 Numerical methods . . . 48
3.4.1 Splitting scheme for Linearly Dissipated Hamiltonian Dynamics (LDHD) 48
3.4.2 A splitting scheme for friction-adaptive descent . . . 49
The “C” step for KFAD/FFAD/mixture coupling . . . 50
The “C” step for general matrices $A(x)$ . . . 51
3.4.3 Alternative solution for higher accuracy . . . 51
3.5 Convergence analysis . . . 52
3.6 Numerical experiments . . . 58
3.6.1 Parameter selection in the Rosenbrock problem . . . 58
3.6.2 Lennard–Jones and Morse Clusters . . . 64
Optimal choice of α and μ . . . 65
Exploration of LJ75 . . . 66
Approximate global minimization of a Morse cluster . . . 68

4 Higher-Order Damping Mechanisms for Neural Network Training 72
4.1 Introduction . . . 72
4.2 Algorithms . . . 75
4.2.1 Individual Kinetic Friction Adaptive Descent (iKFAD) . . . 75
Dynamical properties of iKFAD . . . 80
Exponential convergence of the continuous iKFAD dynamics . . . 81
Convergence analysis of discrete iKFAD dynamics . . . 83
4.2.2 Cubically Damped Momentum Gradient Descent (CD) . . . 84

Dynamical properties of CD . . . 85
Numerical Discretization . . . 87
Convergence analysis of discrete dynamics . . . 88
Convergence of discretized CD dynamics with stochastic gradients . 91
4.2.3 CADAM . . . 92
Numerical discretization of CADAM . . . 92
4.3 Numerical experiments . . . 92
4.3.1 FMNIST . . . 93
4.3.2 CIFAR-10 . . . 96
4.3.3 CIFAR-100 . . . 98
4.3.4 Training DistilBERT on QNLI dataset . . . 100
4.3.5 Training DistilBERT on SST-2 dataset . . . 101

5 Preconditioned Learning Rates for Continual Learning 104
5.1 Introduction . . . 104
5.2 Natural gradient descent and adaptive gradient methods . . . 105
5.3 Continual learning . . . 107
5.3.1 Elastic Weight Consolidation . . . 108
5.4 Evaluation of continual learning benchmarks . . . 109
5.5 Multirate Methods . . . 110
5.6 Experiments . . . 111
5.6.1 Permuted MNIST . . . 111
5.6.2 Single-headed split MNIST . . . 113
5.7 Convergence Analysis . . . 114
5.7.1 Multirate SGD based on Fisher Information Matrix . . . 114
5.8 Next steps . . . 117

6 Conclusions 118

Appendices

A Additional Numerics from Chapter 4 120
A.1 CIFAR-10 . . . 120
A.2 CIFAR-100 . . . 122
A.3 QNLI . . . 125

Bibliography 126

Chapter 1

Preliminaries

1.1 Optimization Methods: A dynamical systems perspective

Consider a function $f : \mathbb{R}^d \to \mathbb{R}$. Optimisation methods, in their simplest form seek to solve the following problem

$$x^* = \arg\min f(x), \qquad x \in \mathbb{R}^d,$$

The goal of an optimisation method is therefore to find the or a minimum of a function. In most real world applications, the function f is non-convex and multimodal, meaning it has multiple minima. A minimum x^* is called a *global* minimum of f if $f(x^*) \leq f(x) \quad \forall x \in \mathbb{R}^d$. Although global optimisation techniques exist, that aim to find the minimum of a function [48], in this thesis we mainly concern ourselves with *local* optimisation. We want to find a good quality minimum without necessarily caring whether the minimum is a global one. In fact, finding the global minimum in applications such as neural network optimisation can be undesirable as it may lead to overfitting [15]. The authors of [15] also show that for large networks, multiple high-quality local minima can be found around the global minimum. Even though we mainly study the methods as local optimisers, we do actually also use and study them in the context of stochastic gradients, which can be seen as a kind of global optimisation mechanism that can lead to improved generalisation. The stochastic gradients allow the paths to leave the vicinity of a single local minimum and explore more widely. When it comes to finding the minima of high-dimensional multimodal optimisation landscapes, the use of analytical techniques is in most cases impractical. When navigating such landscapes, we can only rely on local information about the function f and its derivatives in order to navigate the space. One main categorisation criterion of optimisation methods is the highest derivative order used by the optimiser during the minimisation process. Thus, zero-order optimisation methods only use the values of f during the search for the minimum. First order methods use the gradient $\nabla f(x)$ as a means to navigate the optimisation landscape and second order methods use both the gradient and Hessian $\nabla^2 f(x)$ to minimise f. The focus of this thesis is on the development of first order optimisation methods.

1.1.1 First-Order Optimisation Methods

First-order optimisation methods update the argument x of the function f being minimised by taking small steps in the direction of the gradient. The gradient $\nabla f(x) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d}\right]^T$ of a function points to the direction of steepest ascent / increase of f. To see why that is the case, we consider the directional derivative of f along an arbitrary unit vector $v \in \mathbb{R}^d$ defined as

$$\nabla_v f(x) = \nabla f(x) \cdot v = \|\nabla f(x)\|\|v\| \cos(\theta) = \|\nabla f(x)\| \cos(\theta),$$

where θ is the angle between the two vectors $\nabla f(x)$ and v, "$\cdot$" denotes the inner product between two vectors and "$\|\cdot\|$" the Euclidian norm, unless stated otherwise. Note: throughout the document we will use the notation $u^T v$ and $u \cdot v$ interchangeably to denote the inner product between vectors u and v. The directional derivative gives us the rate of change of $f(x)$ in the direction of v. It is trivial to see that the directional derivative is maximum when $\theta = 0$, or $\theta = \pi$, i.e. when we move along a vector v that has the same direction as or opposite direction to the gradient. Thus, the gradient points to the direction of fastest rate of change of the function, i.e.

$$\arg\max_{\theta} \left[\|\nabla f(x)\| \cos(\theta)\right] = 0.$$

The gradient vector itself points in the direction of steepest ascent (this can be deduced from its definition as the vector of partial derivatives of f), whereas the direction opposite to that of the gradient points to the direction of steepest descent.

Gradient Descent

This previous observation motivates the introduction of the first and simplest first-order optimisation method called Gradient Descent (GD). Gradient descent iteratively updates x_n by taking a small step in the direction opposite to the gradient, thus ensuring that the iterates are always moving in the direction of fastest decrease of f.

The update rule for gradient descent therefore reads

$$x_{n+1} = x_n - \delta t \nabla f(x_n), \tag{1.1}$$

where δt is called the step size and determines the magnitude of the update in x. Note that we can rewrite equation (1.1) as $(x_{n+1} - x_n)/\delta t = -\nabla f(x_n)$. Taking the limit of the above equation as $\delta t \to 0$, yields

$$\dot{x} = -\nabla f(x). \tag{1.2}$$

The continuous dynamics of gradient descent therefore correspond to the gradient flow (1.2), where the trajectories of x follow the direction of steepest descent of f. Notice that along any trajectory following the dynamics of equation 1.2, we have $\dot{f}(x(t)) = -\|\nabla f(x(t))\|^2 \leq 0$, meaning that the value of f is non-increasing.

Momentum Gradient Descent

Although for a sufficiently small step size, gradient descent is guaranteed to lead to a monotonically decreasing sequence of $f(x_n)$, it can suffer in the presence of pathological curvature, for example when the Hessian is ill-conditioned or if there are regions with very low gradients or saddle points.

One way to overcome this issue, is to introduce a short-term memory of previous gradients, otherwise called "momentum" to the dynamics of gradient descent. Momentum gradient descent (mGD) introduces an auxiliary variable called "momentum" which encodes history of previous gradients. This introduces some "inertia" to the optimiser dynamics, which can help the dynamics to overcome energy barriers, advance along flat regions or saddle points and can smooth out oscillations due to high curvature. The equations for momentum gradient descent are

$$p_{n+1} = \rho p_n + (1-\rho)\nabla f(x_n). \tag{1.3}$$

$$x_{n+1} = x_n - \delta t p_{n+1}, \tag{1.4}$$

where $\rho \in [0,1)$ is called the momentum coefficient. Appropriately choosing the momentum coefficient is of primary importance as momentum can both suppress oscillations in directions of high curvature and advance convergence in low curvature regions, but it can also introduce oscillations depending on the step size, due to the dynamics of the optimizer not being a gradient flow.

It is useful here to comment on the nature of the momentum update rule given by equation (1.3). More specifically, assuming $p_0 = 0$ the series of momentum updates we obtain from (1.3) is

$$\begin{aligned}
p_1 &= \rho p_0 + (1-\rho)\nabla f(x_0) = (1-\rho)\nabla f(x_0),\\
p_2 &= \rho p_1 + (1-\rho)\nabla f(x_1) = \rho(1-\rho)\nabla f(x_0)) + (1-\rho)\nabla f(x_1),\\
p_3 &= \rho p_2 + (1-\rho)\nabla f(x_2) = \rho^2(1-\rho)\nabla f(x_0) + \rho(1-\rho)\nabla f(x_1) + (1-\rho)\nabla f(x_2),\\
&\vdots\\
p_n &= \sum_{i=1}^{n} \rho^{n-i}(1-\rho)\nabla f(x_{i-1}).
\end{aligned} \tag{1.5}$$

We can see that p_n can be expressed as an exponentially weighted average of past gradients. The value of ρ determines how important "newer" gradients are, i.e. how quickly they are taken into account. In the limit case where $\rho = 0$, we have $p_n = \nabla f(x_{n-1})$, which means we only take the current gradient into account without looking at past information. This corresponds to the dynamics of gradient descent. As $\rho \to 1$ on the other hand, the inertia of the system increases with the most recent gradients being taken less and less into account.

AdaGrad

The methods mentioned so far, solely make use of the gradient information at each point to update the function parameters. Thus, if the gradient at point x is high, a method like gradient descent will take a big step in the direction opposite to that of the gradient. This update mechanism however can lead to the appearance of oscillations in regions with high gradients and high curvature. Similarly, in flat regions, with low gradients, using methods that only take the gradient information into account can lead to slow convergence. When navigating a loss landscape, it is therefore desirable to also take the local curvature (Hessian) into account. However, computing the Hessian and its inverse at each optimisation step can be computationally prohibitive.

The authors of [26], proposed an optimiser, called AdaGrad, for individually adapting the step size for each parameter based on the sum of the squares of past gradients. The dynamics of the optimiser, are as follows

$$x_{n+1} = x_n - \delta t G_n^{-1/2} \nabla f(x_n), \tag{1.6}$$

where $G_n = \sum_{i=1}^{n} \nabla f(x_i) \nabla f(x_i)^T$. AdaGrad therefore adapts the learning rate of each parameter based on the magnitude of previous gradients. This can be particularly useful in cases where input data is sparse (corresponding parameters have low gradients that would normally lead to negligible updates), or curvature and gradients are low. In such cases, AdaGrad would boost the learning rate for parameters with low gradients and prevent the dynamics from getting stuck in flat regions. The main drawback of this optimiser is that for parameters with higher gradients, learning rates can eventually become very small, potentially stalling training.

RMSProp

RMSProp is an unpublished optimiser introduced by Geoffrey Hinton in one of his classes [1] with the aim of dealing with the vanishing learning rate issue in AdaGrad. Instead of accumulating the squares of all previous gradients, RMSProp proposes the use of an exponentially weighted average (short-term history) of the squares of previous gradients instead of the sum of all previous squared gradients. The dynamics of the optimiser are as follows

$$x_{n+1} = x_n - \delta t \frac{\nabla f(x_n)}{\sqrt{\zeta_n} + \varepsilon}, \tag{1.7}$$

$$\zeta_{n+1} = \beta \zeta_n + (1-\beta)[\nabla f(x_{n+1})]^2, \tag{1.8}$$

where $\beta \in (0,1)$ is a parameter whose value determines the balance between taking old and new gradients into account in computing ζ. The notation "$[\cdot]^n$" is used to denote element-wise exponentiation to the power of n.

1. https://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam

Both AdaGrad and RMSProp utilise individual adaptive step sizes, but neither of them incorporates the momentum mechanism of mGD. Adam [61] is an optimiser that combines the benefits of individual adaptive sizes with the benefits of momentum. The discrete dynamics of Adam are the following

$$x_{n+1} = x_n - \delta t \frac{p}{\sqrt{\zeta_n} + \varepsilon}, \tag{1.9}$$

$$p_{n+1} = \beta_1 p_n + (1-\beta_1)\nabla f(x_{n+1}), \tag{1.10}$$

$$\zeta_{n+1} = \beta \zeta_n + (1-\beta)[\nabla f(x_{n+1})]^2, \tag{1.11}$$

The momentum variable p is an exponentially weighted average of all past gradients, whereas ζ is an exponentially weighted average of the squares of previous gradients. An explanation for the improved convergence of Adam in many applications is that it approximates the first and second moment of the gradients, with the second moment being used to precondition the dynamics and speed up convergence.

Nesterov Accelerated Gradient

Nesterov's accelerated gradient (NAG) method [92, 93, 131] is very similar to momentum gradient descent but incorporates a "look-ahead" mechanism whose aim is to reduce oscillations often appearing in standard mGD.

As a reminder, the momentum variable in mGD represents, an exponentially weighted average of previous gradients. Hence, the momentum mechanism can have a smoothing effect on the dynamics in the presence of oscillating gradients (at the same time, mGD can potentially introduce oscillations of its own, due to the corresponding continuous dynamics not being a gradient flow). To motivate the Nesterov acceleration mechanism, let us look at the x update step in momentum gradient descent. Using equation (1.3), the x update step of equation (1.4) can be written as

$$x_{n+1} = x_n - \underbrace{\delta t \rho p_n}_{T_1} - \underbrace{\delta t (1-\rho)\nabla f(x_n)}_{T_2}.$$

We see that the mGD update in x involves two terms/steps, T_1 and T_2. Momentum gradient descent performs a first partial update step (T_1) on x_n based on the previous momentum value p_n and then a second and final update step on x_n based on the gradient at the starting point x_n. Nesterov's accelerated gradient method also performs the same first update (T_1) on x_n based on the momentum from the previous step. This step leads the discrete dynamics to the intermediate point $x_{n+1/2} = x_n - \delta t \rho p_n$. Nesterov's method takes advantage of that intermediate position and computes the gradient there before performing the final update T_2

to compute x_{n+1}. The Nesterov accelerated gradient method equations are given as

$$p_{n+1} = \rho p_n + (1-\rho)\nabla f(x_n - \delta t \rho p_n), \tag{1.12}$$

$$x_{n+1} = x_n - \delta t p_{n+1}. \tag{1.13}$$

We see that the gradient is computed at the intermediate point $\nabla f(x_n - \delta t \rho p_n)$ rather than at x_n. This allows the algorithm to "see" slightly ahead and adjust more quickly to changes of gradient sign or magnitude.

1.1.2 Second-Order Optimisation Methods

So far we have examined first order optimisation methods, where the only information used by the optimiser is the gradient of the loss function at each iterate. Second order optimisation schemes further take the curvature/Hessian into account when performing an optimisation step. While second order methods can be more accurate and converge faster than their first order counterparts [78], computing the Hessian matrix at every step of the optimisation process can be very costly $O(d^2)$. In the following subsections we will present some of the most popular second order optimisation methods.

Newton's Method

Consider the following second-order Taylor series approximation $\tilde{f}$ of a loss function f

$$f(x_n + h) \approx \tilde{f}_n(h) = f(x_n) + \nabla f(x_n) \cdot h + \frac{1}{2} h^T \nabla^2 f(x_n) h, \tag{1.14}$$

where $h = x_{n+1} - x_n$. The idea behind Newton's method is to minimise the above quadratic approximation of f at each optimisation step [7, 41]. Differentiating equation (1.14), setting the gradient of $\tilde{f}_n(n)$ to zero, solving for x and scaling the update by a step size δt_n we obtain

$$x_{n+1} = x_n - \delta t_n \nabla^2 f(x_n)^{-1} \nabla f(x_n), \tag{1.15}$$

which is the update equation used in Newton's method. One of the downsides of Newton's method is that it requires computation and inversion of the Hessian matrix of the loss function, which can make the method unsuited for high-dimensional machine learning problems. Besides the increased computational cost, analysis of Newton's method typically also assumes that the Hessian is positive definite [41], (section 8.6). In deep learning application, the loss landsacpe is usually non-convex, which also limits the applicability of the method on machine learning loss landscapes [41].

Quasi-Newton Methods

Newton's method has an accelerated convergence rate compared to gradient descent for convex problems [93], but computing and inverting the Hessian can render the use of the method prohibitive for high-dimensional deep learning applications. Quasi Newton methods were developed as an attempt to deal with the computational burden imposed by the need to compute and invert the Hessian. The update rule for Quasi-Newton methods has the following general form

$$x_{n+1} = x_n - \delta t_n B_n^{-1} \nabla f(x_n), \tag{1.16}$$

which is similar to the update used in Newton's method, the difference being that instead of using the inverse of the Hessian, a matrix B_n is used instead. The goal is to use a B_n that leads to faster convergence than gradient descent but also is less expensive to compute than the inverse of the Hessian.

As an example, one potential choice would to set B_n to be a diagonal approximation of the Hessian. This would reduce the computational cost compared to Newton's method and potentially improve convergence speed in some scenarios, but a diagonal approximation of the Hessian might not always be useful, depending on the geometry of the loss landscape.

BFGS

The BFGS method, named after its inventors, Broyden, Fletcher, Goldfarb and Shanno, is the most popular quasi-Newton method [12, 30, 38, 96, 118]. Similarly to Newton's method, the idea is to write down a quadratic approximation for the objective function $f(x)$ and minimise that approximation at each optimisation step. Unlike Newton's method where we use a second-order Taylor approximation of f, in BFGS we have the following approximation for f around point x_n

$$\tilde{f}_n(h) = f(x_n) + \nabla f(x_n)^T h + \frac{1}{2} h^T B_n h, \tag{1.17}$$

where B_n is a symmetric positive definite matrix. B_n must be easier to compute and invert than the Hessian and must also give an improved convergence rate over gradient descent. Setting $\nabla \tilde{f}(h) = 0$ we obtain $h_n = -B_n^{-1} \nabla f(x_n)$. We therefore have the following update rule for BFGS

$$x_{n+1} = x_n - \delta t_n B_n^{-1} \nabla f(x_n). \tag{1.18}$$

As mentioned, our goal is to formulate a less expensive quadratic approximation to $f(x)$, therefore we want B_n to encode curvature along the trajectory of the iterates. If two non-linear functions have the same gradient at two points close to each other, then they have similar local curvature. With that in mind, one can demand that the gradient of $\tilde{f}_{n+1}$ be the same as the gradient of f at two points x_n and x_{n+1}, thus forcing the curvatures of the two functions to

be similar. The following analysis can be found in [96]. We demand

$$\nabla \tilde{f}_{n+1}(0) = \nabla f(x_{n+1}) \qquad \text{and} \qquad \nabla \tilde{f}_{n+1}(-\delta t_n h_n) = \nabla f(x_n). \tag{1.19}$$

The gradient of the approximation $\tilde{f}$ of the loss function f at iteration $n+1$ is

$$\nabla \tilde{f}_{n+1}(h) = \nabla f(x_{n+1}) + B_{n+1} h.$$

We see that the first condition in (1.19) is automatically satisfied. From the second condition and using that fact that $x_{n+1} = x_n + \delta t_n h_n$, we have

$$B_{n+1}(x_{n+1} - x_n) = \nabla f(x_{n+1}) - \nabla f(x_n).$$

For simplicity of notation we define $s_n = x_{n+1} - x_n$ and $y_n = \nabla f(x_{n+1}) - \nabla f(x_n)$, upon which, the condition for B_{n+1} becomes

$$B_{n+1} s_n = y_n. \tag{1.20}$$

This is known as the secant equation [96]. Given that B is positive definite, the following condition must hold $s_n^T y_n > 0$. Finally, we further demand that B be symmetric. The secant equation (1.20), combined with the conditions $s_n^T y_n > 0$ and $B^T = B$, are not sufficient to uniquely determine B_{n+1} . To uniquely determine B_{n+1} we also require that B_{n+1} is as close to B_n as possible. This ensures that there are not any abrupt changes in the Hessian. Therefore B_{n+1} is obtained by solving the following optimisation problem [96]

$$\min_B \|B - B_n\|, \tag{1.21}$$

subject to

$$B s_n = y_n \qquad \text{and} \qquad B^T = B, \tag{1.22}$$

where $s_n^T y_N > 0$. After an appropriate choice of norm (we refer the reader to [96] for more details), the solution to the minimisation problem (1.21) is

$$B_{n+1} = (I - \rho_n y_n s_n^T) B_n (I - \rho_n s_n y_n^T) + \rho_n y_n y_n^T, \tag{1.23}$$

where $\rho_n = \frac{1}{y_n^T s_n}$. In order to apply the update rule (1.18) the matrix B_{n+1} needs to be inverted. The Sherman–Morrison formula [119] is a formula used to compute the inverse of a rank-one update to a matrix whose inverse has previously been computed. Using that, and defining $H_n = B_n^{-1}$, we obtain

$$H_{n+1} = H_n - \frac{H_n y_n y_n^T B_n}{y_n^T H_n y_n} + \frac{s_n s_n^T}{y_n^T s_n}. \tag{1.24}$$

This formula is called the DFP (Davidon–Fletcher–Powell) updating formula [19, 31]. If instead of imposing conditions (1.22) on B, we impose them directly on the inverse H and solve the optimisation problem (1.21) for H, we obtain the BFGS update rule

$$H_{n+1} = (I - \rho_n s_n y_n^T) H_n (I - \rho_n y_n s) n^T) + \rho_n s_n s_n^T. \tag{1.25}$$

The BFGS update rule has replaced the DFP update as a more effective formula [96].

BFGS might be more cost-effective than Newton's method, but it still involves computing and storing an approximation of the inverse of the Hessian. This can make the use of BFGS impractical for some high-dimensional deep learning problems [40]. Momentum gradient descent is simpler and has a lower computational cost, as well as lower memory requirements, making it more suitable for large-scale problems. However, BFGS often converges faster than momentum gradient descent, especially for well-conditioned problems.

1.1.3 Difference and Differential Equations

So far, we have looked at the dynamics of various optimisers from a discrete perspective. Considering a state z_n, the next state z_{n+1} is obtained by solving a set of difference equations of the form $z_{n+1} = \phi_{\delta t}(z_n)$, where δt is the step size of the discrete dynamics and regulates the magnitute of the update in z. Dynamics of this form is called discrete or discretised as the series of z_n consists of a set of discrete values. Difference and differential equations are often closely related. In many cases, a difference equation can be shown to be a discrete approximation of a differential equation and conversely, the process of studying and solving differential equations often necessitates the study of their discrete counterparts (difference equations).

For some discrete dynamics such as those corresponding to gradient descent (1.1), deriving the associated continuous dynamics is straightforward and can be done simply by rearranging the terms in the discrete equations and taking the limit as $\delta t \to 0$, as shown in the case of gradient descent (see section 1.1.1).

Going from a set of continuous ODEs to a set of discretised equations is more straightforward and involves converting derivative terms into difference terms. Two common discretisation schemes are the forward and backward Euler method discussed below.

Forward Euler method

Consider the following ODE

$$\dot{x}(t) = g(x(t)). \tag{1.26}$$

The Euler method assumes that a curve around a point can be locally approximated by its tangent line at that point, i.e. it uses a local linear approximation of the function. The forward Euler method approximates the slope at current point t using point t and a nearby point $t+\delta t$, where δt is a sufficiently small step size. The derivative $\dot{x}(t)$ can therefore be written as

$$\begin{aligned}\dot{x}(t) = g(x(t)) &\approx \frac{x(t+\delta t)-x(t)}{\delta t} \Rightarrow \\ x(t+\delta t) &\approx x(t) + \delta t g(x(t)).\end{aligned}$$

We can see that using the forward Euler method to discretise equation (1.26) leads to a discrete equation where the right hand side contains only terms at the current point t, the values of which, are known. Hence, this scheme allows us to easily solve for $x(t+\delta t)$ given $x(t)$.

Backward Euler method

The backward Euler method also uses finite differences to approximate the derivative at point t, but instead of using points t and $t+\delta t$ to compute an approximation of the slope at t, it uses points t and $t-\delta t$. We therefore have

$$\begin{aligned}\dot{x}(t) = g(x(t)) &\approx \frac{x(t)-x(t-\delta t)}{\delta t} \Rightarrow \\ x(t) &\approx x(t-\delta t) + \delta t g(x(t)),\end{aligned}$$

or written for point $t+\delta t$

$$x(t+\delta t) \approx x(t) + \delta t g(x(t+\delta t)).$$

We see that for this discretisation, the unknown quantity $x(t+\delta t)$ appears on both the left and right hand side of the discretised equation. For this reason, the backward Euler method is called an implicit scheme. Depending on the exact form of g, for example whether it is a linear on non-linear function, solving for $x(t+\delta t)$ can be straightforward or could require the solution of an equation.

Splitting schemes

Consider the following initial value problem

$$\dot{x}(t) = f(x(t)), \tag{1.27}$$

where $x(0) = x_0$. Let us also assume that $f(x)$ can be decomposed into a sum of simpler functions, i.e. $f(x) = f_1(x) + f_2(x) + \ldots + f_N(x)$. We define a mapping $\Phi_t(x)$ that maps a point in phase space to its position in phase space after some time t under the dynamics of the system (1.27). The flow map therefore describes the evolution of a point in time. We call this mapping the flow map. The flow map is therefore a function of time and is equal to the solution $x(t)$ of the initial value problem 1.27, i.e. $\Phi_t(x) = x(t)$. We then consider the systems of subdynamics

$$\begin{aligned}
\dot{x} &= f_1(x), \\
\dot{x} &= f_2(x), \\
&\vdots \\
\dot{x} &= f_N(x)
\end{aligned}$$

and their corresponding flow maps $\Phi_{1,t}(x), \Phi_{2,t}(x), \ldots \Phi_{N,t}(x)$. The idea behind splitting methods is to solve each of the subdynamics of the original system separately and approximate the flow map of the discretised equations as a composition of the individual flow maps of each subdynamics, i.e. $\Phi_{\delta t} = \Phi_{1,\delta t} \circ \Phi_{2,\delta t} \circ \ldots \circ \Phi_{N,\delta t}$. For a comprehensive view of splitting methods we refer the reader to the following resources [10, 72, 87].

A continuous formulation of Momentum Gradient Descent

In section 1.1.1, we presented the dynamics of momentum gradient descent in the form of the discrete equations (1.4), (1.3). However, expressing the dynamics in the form of a system of ODEs can often prove useful in developing understanding and intuition about the methods and can facilitate the design of new optimisers. In this section we will show how the mGD discrete equations (1.4), (1.3) can be obtained as an Euler discretisation of dissipative Hamiltonian dynamics (DHD). We start with the following system of ODEs corresponding to DHD

$$\dot{x} = p, \tag{1.28}$$

$$\dot{p} = -\nabla f(x) - \gamma p. \tag{1.29}$$

It has been shown in [122], [82], that the above formulation of mGD is equivalent to the discrete mGD equations commonly appearing in the literature [110].

As demonstrated in [122], if we take the Euler discretization of the system of equations (1.28), (1.29), with time step δt, we obtain

$$\begin{aligned} p_{n+1} &= p_n - \delta t(\nabla f(x_n) - \gamma p_n) \\ x_{n+1} &= x_n + \delta t\, p_n \end{aligned} \quad \Longrightarrow \quad \begin{aligned} p_{n+1} &= (1-\delta t\gamma)p_n - \delta t \nabla f(x_n) \\ x_{n+1} &= x_n + \delta t\, p_n \end{aligned} \tag{1.30}$$

Multiplying the momentum equation by the step size δt and setting $\tilde{p}_n = \delta t\, p_n$, we obtain

$$\begin{aligned} \delta t\, p_{n+1} &= (1-\delta t\gamma)\delta t\, p_n - \delta t^2(\nabla f(x_n), \\ x_{n+1} &= x_n + \delta t\, p_n, \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \tilde{p}_{n+1} &= (1-\delta t\gamma)\tilde{p}_n - \delta t^2 \nabla f(x_n), \\ x_{n+1} &= x_n + \delta t\, p_n. \end{aligned}$$

Finally, by making the change of variables $\tilde{\gamma} = 1 - \delta t\gamma$ and $\tilde{\delta} t = \delta t^2$, the discretized equations can be rewritten as

$$\begin{aligned} \tilde{p}_{n+1} &= \tilde{\gamma}\tilde{p}_n - \tilde{\delta} t \nabla f(x_n), \\ x_{n+1} &= x_n + \tilde{p}_n, \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} \tilde{p}_{n+1} &= \tilde{\gamma}\tilde{p}_n + \tilde{\delta} t \nabla f(x_n), \\ x_{n+1} &= x_n - \tilde{p}_n, \end{aligned} \tag{1.31}$$

which is the mSGD scheme [110]. This continuous formulation of momentum gradient descent will later be useful in motivating the design of our proposed adaptive friction optimisers in Chapter 3.

One more thing to note is that we can solve the continuous form of the momentum equation (1.29) for $p(t)$ as follows

$$\begin{aligned} &\dot{p}(t) = -\nabla f(x(t)) - \gamma p(t) \Rightarrow \int_0^t \frac{d}{dt}(p(t)e^{\gamma u})du = -\int_0^t \nabla f(x(u))e^{\gamma u}du \Rightarrow \\ &p(t) = e^{-\gamma t}p(0) - \int_0^t \nabla f(x(t))e^{-\gamma(t-u)}du \quad \overset{\tilde{p}=-p}{\Rightarrow} \\ &\tilde{p}(t) = e^{-\gamma t}\tilde{p}(0) + \int_0^t \nabla f(x(t))e^{-\gamma(t-u)}du, \end{aligned} \tag{1.32}$$

where we did a variable substitution at the end to remove the negative sign in front of the integral of the gradients. From equation (1.32) we see that the momentum variable encodes the history of past gradients in the form of an exponentially weighted average. This is in accordance with the result for the discrete dynamics of momentum gradient descent as derived in equation (1.5) .

Continuous dynamics formulation of Adam

The discrete equations for Adam also have a corresponding continuous dynamics counterpart [17]. Similarly to [17], we show that Adam can be expressed as a discretization of the following dynamical system

$$\dot{x} = \frac{p}{\sqrt{\zeta + \varepsilon}}, \tag{1.33}$$

$$\dot{p} = -\nabla f(x) - \gamma p, \tag{1.34}$$

$$\dot{\zeta} = [\nabla f(x)]^2 - \alpha\zeta. \tag{1.35}$$

In order to be consistent with the notation used for Adam in the literature, we introduce the change of variables $\bar{p} = -p$, upon which the above system of equations becomes

$$\dot{x} = -\frac{\bar{p}}{\sqrt{\zeta + \varepsilon}}, \tag{1.36}$$

$$\dot{\bar{p}} = \nabla f(x) - \gamma\bar{p} \tag{1.37}$$

$$\dot{\zeta} = [\nabla f(x)]^2 - \alpha\zeta. \tag{1.38}$$

By applying an Euler discretization to the above system of equations, we get

$$\begin{aligned} \bar{p}_{n+1} &= \bar{p}_n + \delta t(\nabla f(x_n) - \gamma\bar{p}_n), \\ \zeta_{n+1} &= \zeta_n + \delta t([\nabla f(x_n)]^2 - \alpha\zeta_n), \\ x_{n+1} &= x_n - \delta t\left(\frac{\bar{p}_{n+1}}{\sqrt{\zeta_{n+1} + \varepsilon}}\right), \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \bar{p}_{n+1} &= (1 - \gamma\delta t)\bar{p}_n + \delta t\nabla f(x_n), \\ \zeta_{n+1} &= (1 - \alpha\delta t)\zeta_n + \delta t[\nabla f(x_n)]^2, \\ x_{n+1} &= x_n - \delta t\left(\frac{\bar{p}_{n+1}}{\sqrt{\zeta_{n+1} + \varepsilon}}\right). \end{aligned}$$

Setting $\beta_1 = 1 - \gamma\delta t \Rightarrow \delta t = (1 - \beta_1)/\gamma$ and $\beta_2 = 1 - \alpha\delta t \Rightarrow \delta t = (1 - \beta_2)/\alpha$, we can rewrite the discretized system as

$$\begin{aligned} \bar{p}_{n+1} &= \beta_1\bar{p}_n + \frac{1 - \beta_1}{\gamma}\nabla f(x_n), \\ \zeta_{n+1} &= \beta_2\zeta_n + \frac{1 - \beta_2}{\alpha}[\nabla f(x_n)]^2, \\ x_{n+1} &= x_n - \delta t\left(\frac{\bar{p}_{n+1}}{\sqrt{\zeta_{n+1} + \varepsilon}}\right). \end{aligned}$$

Multiplying the $\bar{p}$ equation by γ and the ζ equation by α, setting $\tilde{p} = \gamma\bar{p}$ and $\tilde{\zeta} = \alpha\zeta$ we obtain

$$\tilde{p}_{n+1} = \beta_1\tilde{p}_n + (1 - \beta_1)\nabla f(x_n), \tag{1.39}$$

$$\tilde{\zeta}_{n+1} = \beta_2\tilde{\zeta}_n + (1 - \beta_2)[\nabla f(x_n)]^2, \tag{1.40}$$

$$x_{n+1} = x_n - \tilde{\delta}t\left(\frac{\tilde{p}_{n+1}}{\sqrt{\tilde{\zeta}_{n+1}+\tilde{\varepsilon}}}\right), \tag{1.41}$$

where we have made the changes of variable $\tilde{\delta}t = \frac{\delta t\sqrt{\alpha}}{\gamma}$ and $\tilde{\varepsilon} = \alpha\varepsilon$.

We can see that, discretization of the system of equations (1.33), (1.34) and (1.35) results in the system of equations (1.39), (1.40), (1.41), which are the equations for Adam found in the literature ([61]).

1.2 Lyapunov Functions and Convergence of Optimization Algorithms

Lyapunov functions are a powerful tool in proving stability and convergence of ODEs [56, 57]. When it comes to analysing convergence properties of optimisation algorithms, i.e. proving that the dynamics will reach and remain in the vicinity of a minimum, most convergence proofs look for what is called a "Lyapunov function", otherwise called a "potential function" [7]. The idea behind the construction of Lyapunov functions is to find a function that decreases along the trajectories of the dynamics. Similarly to how the potential energy of a particle rolling down a hill decreases as it moves down the slope, a Lyapunov function must decrease along trajectories of the system.

Definition 1.2.1. A Lyapunov function is a continuous function $\mathscr{W} : \mathbb{R}^d \to \mathbb{R}$ which satisfies the properties below

1. $\mathscr{W}(z) \geq 0 \quad \forall z$.
2. $\mathscr{W}(z) = 0 \Leftrightarrow z = z^*$, where z^* is an equilibrium point of the dynamics we are applying the Lyapunov function to.
3. $\mathscr{W}(z) \to \infty$ as $\|z\| \to \infty$.
4. $\exists c > 0 : \dot{\mathscr{W}}(z) \leq -c\mathscr{W}(z)$

In the discrete setting we replace Condition 4 with the following condition

4. $\exists c \in (0,1) : \mathscr{W}(z_{n+1}) \leq c\mathscr{W}(z_n) \quad \forall n \in \mathbb{N}$.

We state below a lemma known as Grönwall's inequality [42], which can be useful in the process of showing exponential convergence.

Lemma 1.2.2. *Consider an interval I of any of the forms $[a,b], [a,b), [a,+\infty)$, where $a,b \in \mathbb{R}$ and $a < b$. Also consider two real-valued continuous functions $w : I \to \mathbb{R}$, $r : I \to \mathbb{R}$, where w is differentiable on the interior I^o of I. If w satisfies an inequality of the form*

$$\dot{w}(t) \leq r(t)w(t), \quad \textit{where } t \in I^o \textit{ and } r : I \to \mathbb{R},$$

then $w(t)$ is bounded by

$$w(t) \leq w(a)e^{\int_a^t r(s)ds}.$$

Common assumptions for proving convergence

We present some definitions and inequalities below which will be useful for the convergence proofs contained in this thesis.

Definition 1.2.3 (Strong convexity)**.** A differentiable function f is $m-$strongly convex if and only if

$$f(y) \geq f(x) + \nabla f(x) \cdot (y-x) + \frac{m}{2}\|y-x\|^2, \quad \forall x, y \in \mathbb{R}^d.$$

If f is twice differentiable, Definition 1.2.3 is equivalent to $\nabla^2 f(x) \geq m > 0$ for all $x \in \mathbb{R}^d$, where we use the notation $\nabla^2 f(x) \geq m$ to indicate that the smallest eigenvalue of the Hessian matrix is bounded below by m. A more precise way of expressing this would be by writing $\nabla^2 f(x) - mI \geq 0 \Rightarrow \lambda_{\min} - m \geq 0$. In other words, the matrix $\nabla^2 f(x) - mI$ is positive semidefinite. If that is the case, one can derive the inequality in Definition 1.2.3 from the fundamental theorem of calculus using the upper bound on the Hessian.

Definition 1.2.4 (Smoothness)**.** A differentiable function f is $M-$smooth if and only if

$$\|\nabla f(y) - \nabla f(x)\| \leq M\|y-x\|, \quad \forall x, y \in \mathbb{R}^d,$$

that is if its gradient is Lipschitz continuous. As previously mentioned, "$\|\cdot\|$" denotes the Euclidean norm.

Similarly if f is twice differentiable, Definition 1.2.4 is equivalent to $\nabla^2 f(x) \leq M$, i.e. the largest eigenvalue of the Hessian is upper bounded by M, for all $x \in \mathbb{R}^d$.

Lemma 1.2.5. *The condition in Definition 1.2.4 implies the following inequality*

$$f(y) \leq f(x) + \nabla f(x) \cdot (y-x) + \frac{M}{2}\|y-x\|^2, \quad \forall x, y \in \mathbb{R}^d. \tag{1.42}$$

Proof. Following [93][Lemma 1.2.3], we have from the fundamental theorem of calculus we have for all $x, y \in \mathbb{R}^d$ that

$$\begin{aligned} f(y) &= f(x) + \int_0^1 \nabla f(x + t(y-x)) \cdot (y-x)\, dt \\ &= f(x) + \nabla f(x) \cdot (y-x) + \int_0^1 (\nabla f(x + t(y-x)) - \nabla f(x)) \cdot (y-x)\, dt, \end{aligned}$$

and therefore we have

$$f(y) - f(x) - \nabla f(x) \cdot (y-x) \leq \int_0^1 tM\|y-x\|^2 dt = \frac{M}{2}\|y-x\|^2,$$

as required. □

1.2.1 Convergence of Gradient Descent

Below, we present convergence proofs for the continuous and discrete dynamics of gradient descent and momentum gradient descent using Lyapunov functions, following methods found in the literature [65, 75, 113]. To help illustrate the main ideas behind convergence proofs, let us examine the dynamics of gradient descent. The continuous dynamics for gradient descent are given by

$$\dot{x} = -\nabla f(x), \tag{1.43}$$

where we assume that f is a twice differentiable function such that $m \leq \nabla^2 f(x)$.

Let us define $\mathscr{G}(t) = f(x(t)) - f(x^*)$, (where x^* is the equilibrium point of the dynamics) and differentiate it with respect to time

$$\dot{\mathscr{G}}(t) = \dot{f}(x) = \nabla f(x) \cdot \dot{x} = -\|\nabla f(x)\|^2 \leq 0.$$

This means that the value of $\mathscr{G}(t)$ and therefore $f(t)$ is non-increasing along the trajectories of 1.43. This tells us that the dynamics are stable, but does not provide a rate of convergence. In order to obtain a rate of convergence, we define an appropriate Lyapunov function, which allows us to show decay with a rate.

Theorem 1.2.6 (Convergence of continuous gradient descent)**.** *Assume that $f \in C^2(\mathbb{R}^d)$ has minimiser $x^* \in \mathbb{R}$, and $0 < m \leq \nabla^2 f(x)$ for all $x \in \mathbb{R}^d$. Then considering $x(t)$ to be the continuous solution to* (1.43) *in $\mathbb{R}^d$ we have that*

$$\|x(t) - x^*\| \leq e^{-mt}\|x(0) - x^*\|.$$

Proof. We define the following Lyapunov function

$$\mathscr{W}(x(t)) = \frac{1}{2}\|x(t) - x^*\|^2,$$

where, $\|\cdot\|$ denotes the Euclidean norm. Below, we use the notation $\mathscr{W}(t)$ to denote $\mathscr{W}(x(t), p(t), \xi(t))$. Upon differentiating $\mathscr{W}(t)$ and using the fact that $\nabla f(x^*) = 0$, we have

$$\begin{aligned}
\dot{\mathscr{W}}(t) &= (x(t) - x^*) \cdot \dot{x} = -(x(t) - x^*) \cdot \nabla f(x(t)) \\
&= -(x(t) - x^*) \cdot (\nabla f(x(t)) - \nabla f(x^*)) \\
&\leq -[f(x(t)) - f(x^*)] + \frac{m}{2}\|x(t) - x^*\|^2 \\
&\leq m\|x(t) - x^*\|^2 \\
&= -2m\mathscr{W}(t),
\end{aligned}$$

where we used the strong convexity inequality from definition 1.2.3 to upper bound the term $-(x(t)-x^*)\cdot\nabla f(x(t))$ as well as the term $-[f(x(t))-f(x^*)]$ (upon choosing x and y appropriately).

Finally, using Gronwall's inequality (Lemma 1.2.2) we obtain

$$\mathscr{W}(t) \leq e^{-2mt}\mathscr{W}(0),$$

which proves exponential convergence of the dynamics. □

Having shown exponential convergence of the continuous gradient descent dynamics, we will now show convergence for the discretised GD dynamics. Using an Euler discretisation with step size δt we have

$$x_{n+1} = x_n - \delta t \nabla f(x_n). \tag{1.44}$$

Theorem 1.2.7 (Convergence of discrete gradient descent dynamics)**.** *Making the same assumptions of Theorem 1.2.6 and further assuming an upper bound on the largest eigenvalue of the Hessian, i.e.* $\nabla^2 f(x) \leq M$ *as well as imposing the step size restriction* $\delta t \leq 2/M$*, we have for* $(x_n)_{n\in\mathbb{N}}$ *iterates of* (1.44) *that*

$$\|x_n - x^*\| \leq (1 - m\delta t(2-\delta t M))^{n/2}\|x_0 - x^*\|,$$

i.e. under the aforementioned assumptions, we can show convergence in the Euclidean norm.

Proof. We define the following discrete Lyapunov function

$$\mathscr{W}(x_n) = \frac{1}{2}\|x_n - x^*\|^2.$$

We start by upper bounding each of the terms in $\mathscr{W}(x_{n+1})$ as follows

$$\begin{aligned}
\mathscr{W}(x_{n+1}) &= \frac{1}{2}\|x_n - x^* - \delta t\left(\nabla f(x_n) - \nabla f(x^*)\right)\|^2 \\
&= \frac{1}{2}\|x_n - x^*\|^2 - \delta t(x_n - x^*)\cdot(\nabla f(x_n) - \nabla f(x^*)) + \frac{1}{2}\delta t^2\|\nabla f(x_n) - \nabla f(x^*)\|^2 \\
&\leq (1 - m\delta t(2-\delta t M))\mathscr{W}(x_n),
\end{aligned}$$

where we have assumed that $\delta t \leq \frac{2}{M}$. Finally, we have exponential convergence, in particular

$$\mathscr{W}(x_n) \leq (1 - m\delta t(2-\delta t M))^n \mathscr{W}(x_0).$$

Note that, in order to ensure convergence, we must have $1 - m\delta t(2-\delta t M) < 1$, which is why we demanded $\delta t \leq \frac{2}{M}$. □

1.2.2 Convergence of momentum Gradient Descent

The continuous dynamics for momentum gradient descent are given as

$$\dot{x} = p, \tag{1.45}$$
$$\dot{p} = -\nabla f(x) - \gamma p. \tag{1.46}$$

We will again assume that f is a twice differentiable function, with $0 < m \leq \nabla^2 f(x) \leq M$. We define $\mathscr{G}(t) = f(x(t)) - f(x^*) + \frac{1}{2}\|p(t)\|^2$. Taking the time derivative of $\mathscr{G}$, we have

$$\dot{\mathscr{G}}(t) = \dot{f}(x) + p \cdot \dot{p} = \nabla f(x) \cdot \dot{x} + p \cdot (-\nabla f(x) - \gamma p) = -\gamma \|p\|^2 \leq 0.$$

Theorem 1.2.8 (Convergence of continuous mGD)**.** *Consider $(x(t), p(t))_{t\geq 0}$ to be the solution of* (1.45)-(1.46) *and assume that $0 < m \leq \nabla^2 f(x) \leq M$ for all $x \in \mathbb{R}^d$, then we have for $\varepsilon \in [0, 1/2]$ and*

$$\mathscr{W}(t) = f(x(t)) - f(x^*) + \frac{1}{2}\|p(t)\|^2 + \varepsilon(x(t) - x^*) \cdot p(t) + \varepsilon\|x(t) - x^*\|^2,$$

that

$$\mathscr{W}(t) \leq e^{-rt}\mathscr{W}(0),$$

where $r = \min\left\{\varepsilon, \frac{4\left[\gamma - 5\varepsilon/4\right]}{3}, \frac{1}{3}\right\}$. Further, we have that $\mathscr{W}$ is equivalent to the Euclidean norm, in the sense that it can be upper and lower bounded by it.

Proof. In order to obtain a rate of convergence, we define the following Lyapunov function

$$\mathscr{W}(t) = f(x(t)) - f(x^*) + \frac{1}{2}\|p(t)\|^2 + \varepsilon(x(t) - x^*) \cdot p(t) + \varepsilon\|x(t) - x^*\|^2.$$

We first wish to derive a lower bound for $\mathscr{W}(t)$ to ensure positivity of the Lyapunov function. For ease of notation, we shall use x and p to mean $x(t)$ and $p(t)$. We then have that

$$\begin{aligned}\mathscr{W}(t) &= f(x) - f(x^*) + \frac{1}{4}\|p\|^2 + \frac{1}{4}\|p\|^2 + 2\varepsilon(x - x^*) \cdot \frac{p}{2} + \varepsilon^2\|x - x^*\|^2 - \varepsilon^2\|x - x^*\|^2 \\ &\quad + \frac{\varepsilon}{2}\|x - x^*\|^2 + \frac{\varepsilon}{2}\|x - x^*\|^2 \\ &= f(x) - f(x^*) + \frac{1}{4}\|p\|^2 + \left\|\frac{p}{2} + \varepsilon(x - x^*)\right\|^2 + \left(\frac{\varepsilon}{2} - \varepsilon^2\right)\|x - x^*\|^2 + \frac{\varepsilon}{2}\|x - x^*\|^2.\end{aligned}$$

Assuming $\varepsilon \in [0, 1/2]$ we have

$$\mathscr{W}(t) \geq f(x) - f(x^*) + \frac{1}{4}\|p\|^2 + \frac{\varepsilon}{2}\|x - x^*\|^2. \tag{1.47}$$

We can similarly obtain an upper bound as follows

$$\begin{aligned}\mathscr{W}(t) &= f(x) - f(x^*) + \frac{3}{4}\|p\|^2 - \frac{1}{4}\|p\|^2 + 2\varepsilon(x-x^*)\cdot\frac{p}{2} - \varepsilon^2\|x-x^*\|^2 \\ &\quad + \varepsilon^2\|x-x^*\|^2 + \frac{3\varepsilon}{2}\|x-x^*\|^2 - \frac{\varepsilon}{2}\|x-x^*\|^2 \\ &= f(x) - f(x^*) + \frac{3}{4}\|p\|^2 - \left(\left[\frac{p}{4} - \varepsilon(x-x^*)\right] + \left[\frac{\varepsilon}{2} - \varepsilon^2\right]\|x-x^*\|^2\right) \\ &\quad + \frac{3\varepsilon}{2}\|x-x^*\|^2.\end{aligned}$$

Assuming again that $\varepsilon \in [0, 1/2]$ we obtain the following upper bound

$$\mathscr{W}(t) \leq f(x) - f(x^*) + \frac{3}{4}\|p\|^2 + \frac{3\varepsilon}{2}\|x-x^*\|^2. \tag{1.48}$$

Upon differentiating $\mathscr{W}(t)$, we obtain

$$\begin{aligned}\dot{\mathscr{W}}(t) &= \nabla f(x)\cdot\dot{x} + p\cdot\dot{p} + \varepsilon\dot{x}\cdot p + \varepsilon(x-x^*)\cdot\dot{p} + 2\varepsilon(x-x^*)\cdot\dot{x} \\ &= -(\gamma-\varepsilon)\|p\|^2 - \varepsilon\nabla f(x)\cdot(x-x^*) + \varepsilon(2-\gamma)(x-x^*)\cdot p.\end{aligned}$$

From the strong convexity assumption $-\nabla f(x)\cdot(x-x^*) \leq -(f(x)-f(x^*)) - \frac{m}{2}\|x-x^*\|^2$. We therefore have that

$$\begin{aligned}\dot{\mathscr{W}}(t) &\leq -(\gamma-\varepsilon)\|p\|^2 - \varepsilon\big[(f(x)-f(x^*)\big] - \varepsilon\frac{m}{2}\|x-x^*\|^2 + \varepsilon(2-\gamma)(x-x^*)\cdot p \\ &\leq -(\gamma-\varepsilon)\|p\|^2 - \varepsilon\big[(f(x)-f(x^*)\big] - \varepsilon\frac{m}{2}\|x-x^*\|^2 \\ &\quad + \varepsilon(2+\gamma)\left(\frac{1}{4\delta}\|p\|^2 + \delta\|x-x^*\|^2\right),\end{aligned}$$

and hence

$$\dot{\mathscr{W}}(t) \leq -\left(\gamma - \varepsilon\big[1 + (2+\gamma)\frac{1}{4\delta}\big]\right)\|p\|^2 - \varepsilon\big[(f(x)-f(x^*)\big] - \varepsilon\big[1-(2+\gamma)\delta\big]\|x-x^*\|^2.$$

Using the upper bound on $\mathscr{W}(t)$ we obtain

$$\dot{\mathscr{W}}(t) \leq -\min\left\{\varepsilon, \frac{4\big[\gamma - \varepsilon\big(1+(2+\gamma)\frac{1}{4\delta}\big)\big]}{3}, \frac{2\big[1-(2+\gamma)\delta\big]}{3}\right\}\mathscr{W}(t),$$

and setting $\delta = \frac{1}{2(2+\gamma)}$, we have

$$\dot{\mathscr{W}}(t) \leq -\min\left\{\varepsilon, \frac{4\big[\gamma - 5\varepsilon/4\big]}{3}, \frac{1}{3}\right\}\mathscr{W}(t).$$

We have therefore shown that $\exists r > 0$ such that

$$\mathscr{W}(t) \leq e^{-rt}\mathscr{W}(0).$$

Using the lower bound on the Lyapunov function, this leads to the following inequality

$$f(x(t)) - f(x^*) + \frac{1}{4}\|p(t)\|^2 + \frac{\varepsilon}{2}\|x(t) - x^*\|^2 \leq e^{-rt}\mathscr{W}(0),$$

which directly implies convergence in x and p given that $f(x(t)) - f(x^*) \geq 0 \quad \forall t \geq 0$. □

The same Lyapunov function can be used to prove convergence of discretizations of these dynamics (see [114]) for a sufficiently small time step, using similar techniques as we will use in the later chapters.

Chapter 2

Applications of Optimisation in Machine Learning

2.1 Machine Learning

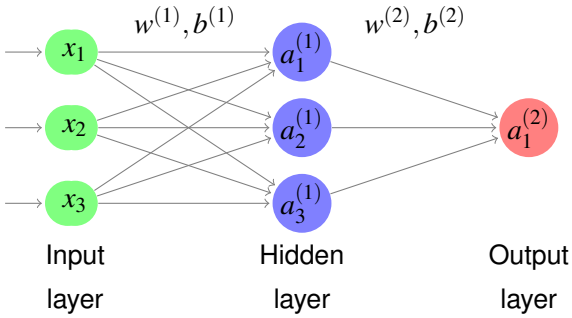
In this section we provide an introduction to the main concepts in the field of Machine Learning, including Neural Networks (NNs) and popular NN architectures, loss functions, back-propagation, which is a gradient estimation method used in NN training and optimisation algorithms. Machine Learning (ML) can be defined as a group of methods for detecting patterns in data in an automated fashion. These patterns are then used to make predictions and decisions under uncertainty [91]. Machine learning methods have been gaining more and more traction as more and more digital data becomes available and their widespread use has been facilitated by efficient programming frameworks and the advancement of computing devices.

2.1.1 Supervised Learning

One of the main branches of machine learning is what is called Supervised Learning. Supervised learning refers to a series of methods that involve "training" a model on a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ consisting of pairs of inputs/features $x_i \in \mathbb{R}^{d_0}$ and corresponding labels $y_i \in \mathbb{R}$, where d_0 is the number of input features to the network and N is the number of data points. Supervised learning tasks themselves can be divided into two main categories: Regression and Classification tasks. In regression problems, the model is designed to produce a continuous valued output, for instance predicting the energy of a molecular configuration given its atomic coordinates or predicting future stock prices given past price data. In classification problems, the model is asked to place an object in a category, for example classify a molecule as being a potential drug or not (binary classification) or a protein as being an enzyme, signalling or transport protein (multiclass classification problem). In classification tasks, the model outputs a discrete value, typically 0 or 1 for binary classification. Unsupervised learning on the other hand involves tasks where the algorithm is not presented with the "right" answers and is instead required to discover patterns in data and perform tasks such as clustering the data in categories (for example news clustering) or dimensionality reduction.

2.1.2 Neural Networks

Neural networks [7, 40, 89] are compositional functions used by machine learning algorithms for the purpose of function approximation. The goal of the learning algorithm is to tune the coefficients (parameters) of the function (model) in order to best fit the given data. An artificial neural network can be visualised as a graph that consists of nodes called "neurons", each of which is connected to other nodes of the model, similarly to how neurons in biological systems form synapses with each other. The following figure shows an example of a simple three-layer fully-connected, feed-forward neural network.



The network consists of an input layer with input feature vector $a^{(0)} = x = [x_1, x_2, x_3]^T$. Each of the connections between neurons, represented as lines in the graph, carries what is called the "weight" and the "bias" of the corresponding connection. The weight is a learnable coefficient that multiplies the input coming in from each connection to a node and the bias is a learnable constant added to the product of the weight and node input. Let us denote the number of nodes/neurons in layer l of the network, as s_l. For a layer $l > 1$ and node $i \in \{1, \ldots, s_l\}$, the node value $a_i^{(l)}$, is computed as the linear combination of all inputs to the node (outputs of the previous layer) $a_j^{(l-1)}$ for $j \in \{1, \ldots, s_{l-1}\}$, each weighted by the corresponding weight $w_{ij}^{(l)}$ of the connection between node i and j plus an offset term (bias) $b_i^{(l)}$. The final node value $a_i^{(l)}$ is then computed by applying a non-linear transformation (called an "activation function") on the linear combination of the inputs.

For the above example, we thus compute the hidden node values as follows

$$
\begin{aligned}
a_1^{(1)} &= \sigma(w_{11}^{(1)} a_1^{(0)} + w_{12}^{(1)} a_2^{(0)} + w_{13}^{(1)} a_3^{(0)} + b_1^{(1)}), \\
a_2^{(1)} &= \sigma(w_{21}^{(1)} a_1^{(0)} + w_{22}^{(1)} a_2^{(0)} + w_{23}^{(1)} a_3^{(0)} + b_2^{(1)}), \\
a_3^{(1)} &= \sigma(w_{31}^{(1)} a_1^{(0)} + w_{32}^{(1)} a_2^{(0)} + w_{33}^{(1)} a_3^{(0)} + b_3^{(1)}),
\end{aligned}
$$

where σ is an activation function that, as mentioned, performs a non-linear operation on the linear combination of the node inputs. In this example we have assumed that the same activation function σ is applied to all layers of the network but that need not necessarily be the case. Finally, the value of the output node is computed as

$$
a_1^{(2)} = \sigma(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + w_{13}^{(2)} a_3^{(1)} + b_1^{(2)}).
$$

Next, we define the weight and bias matrices for the hidden and output layer as follows

$$w^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix} \text{ and } b^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix},$$

$$w^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{pmatrix} \text{ and } b^{(2)} = \begin{pmatrix} b_1^{(2)} \end{pmatrix}.$$

The purpose of the bias is to shift the activation function along the input axes and regulate the activation threshold of each neuron. To demonstrate this, consider the sigmoid function $\sigma(x)$ and the "shifted" sigmoid $\sigma(x+3)$ with a bias equal to three, shown in Fig. 2.1. We see that the shifted sigmoid outputs values closer to one, i.e. "activates" for a different range of inputs. Having a separate bias for each neuron therefore gives greater flexibility to the model.

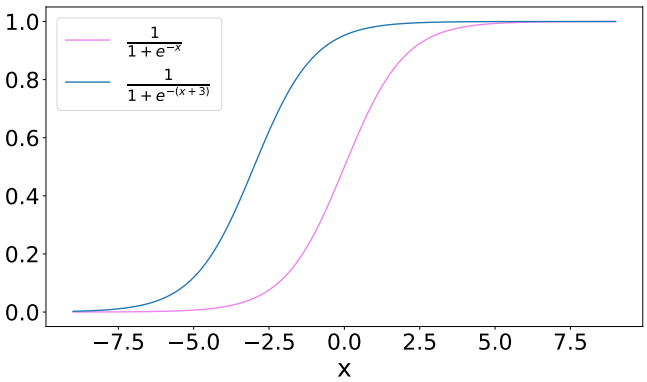


Figure 2.1: Sigmoid versus shifted sigmoid function.

In vectorised form and after defining $z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$, the above computations for layer l can be summarised as

$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}, \tag{2.1}$$

$$a^{(l)} = \sigma(z^{(l)}), \tag{2.2}$$

where $a^{(l)} = [a_1^{(l)}, \ldots, a_{s_l}^{(l)}]$. As demonstrated in the above example, neural networks are compositional functions. A simple fully-connected feed-forward neural network consisting of L layers, with input x and parameters (weights and biases) θ can be expressed as

$$f(x;\theta) = (f_{\theta^{(L)}} \circ \ldots \circ f_{\theta^{(2)}} \circ f_{\theta^{(1)}})(x),$$

where $f_{\theta^{(1)}}, f_{\theta^{(2)}}, \ldots, f_{\theta^{(L)}}$ are the functions corresponding to the $1^{\text{st}}, 2^{\text{nd}}, \ldots,$ and L^{th} layer of the network respectively and $\theta^{(n)}$ represents the n^{th} layer parameters (weights and biases). Assuming each of the nodes in layer $l-1$ is connected to each of the nodes in layer l, the weight matrix $w^{(l)}$ will consist of s_l rows and s_{l-1} columns, i.e. $w^{(l)} \in \mathbb{R}^{s_l \times s_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{s_l}$.

Activation functions

As mentioned in subsection 2.1.2, activation functions are used to introduce non-linearity to the model. The reason for choosing a non-linear activation function is to account for the non-linearity of most datasets. In the example in 2.1.2, if g was chosen to be linear, for example $g(x) = x$, we would have

$$\begin{aligned} a^{(1)} &= f_{\theta^{(1)}}(a^{(0)}) = g(w^{(1)}x + b^{(1)}) = w^{(1)}x + b^{(1)} \\ a^{(2)} &= f_{\theta^{(2)}}(a^{(1)}) = g(w^{(2)}a^{(1)} + b^{(2)}) = w^{(2)}(w^{(1)}x + b^{(1)}) + b^{(2)} \\ &= (w^{(2)}w^{(1)})x + (w^{(2)}b^{(1)} + b^{(2)}) \\ &= wx + b. \end{aligned}$$

This means that if we do not introduce non-linearity into the model through the use of the activation functions, the output of the model will always have a linear dependence on the inputs, which is not desirable, unless we want to approximate a linear function.

Some commonly used activation functions include the

- Sigmoid (or logistic) activation, defined as

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

  The output of the sigmoid activation is restricted to the interval $(0,1)$. This activation is therefore often used on the final layer of a model when doing binary classification tasks, as it "forces" the model to predict a value between zero and one which can be interpreted as the probability of the input belonging to class A or class B respectively. The main issue with the sigmoid function is that for large positive or small negative inputs x, the gradient of $\sigma(x)$ tends to zero (as $\sigma'(x) = \sigma(x)[1-\sigma(x)]$). In neural network training, gradients of the model parameters are computed through a process called "backpropagation", which is essentially an application of the chain rule. For deep neural networks (networks with many layers), very small / very large gradients can get amplified through the "backpropagation" process, which can lead to what is referred to as the "vanishing / exploding gradients" problem respectively. This issue is often addressed by techniques such as "gradient clipping" [103] and "batch normalisation"[51].

- Hyperbolic Tangent (tanh), which is given by the following formula

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

  Similarly to the sigmoid function, the hyperbolic tangent output is also limited to a bounded range, namely the interval $(-1, 1)$.
- and Rectified Linear Unit (ReLU), defined as

$$\sigma(x) = \max(0, x).$$

  The ReLU activation is a popular activation choice for convolutional layers and deep learning models. One issue with the ReLU function is what is called the "dying ReLU" problem [1], where the ReLU neurons fall into inactivity due to their input becoming negative. This can lead to loss of expressive capacity of the network. A solution to this problem can be to use an alternative like a "leaky ReLU" [81], which is a tweaked ReLU with a small but non-zero gradient when the input is negative, which prevents the neuron from becoming inactive. Another issue with the ReLU function is its non-differentiability at zero. The softplus function defined as $\sigma(x) = \log(1 + e^x)$, is a differentiable alternative to the ReLU activation.

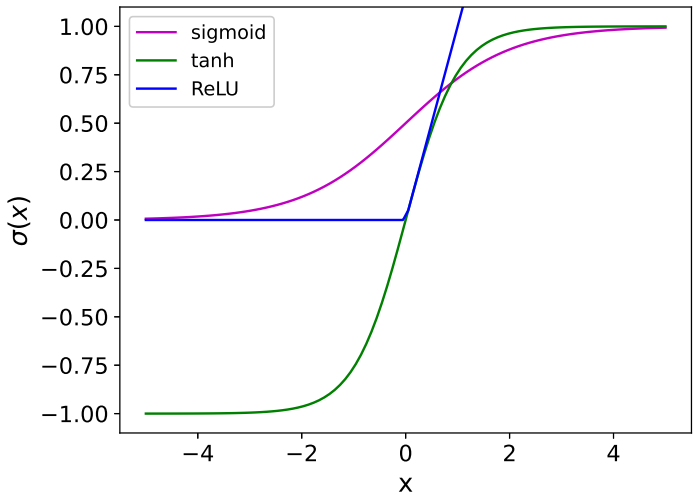


Figure 2.2

The plots of the sigmoid, hyperbolic tangent and ReLU activations can be seen in Fig. 2.2.

Loss functions

In the context of supervised learning, neural network training refers to the process of finding a set of network parameters (weights and biases) that best fit the given data. The question then arises of how to measure the quality of a fit. The metric used depends on the task at hand. For regression tasks, the most common metric is the "Mean Squared Error" (MSE) loss defined as

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2,$$

where $f(x_i)$ is the model output given input x_i and y_i is the corresponding label (desired output). The MSE loss therefore measures the expectation of the difference between the model output and the desired value (label). For classification tasks with N_c classes, the potential labels are discrete ($y \in \{0, 1, \ldots, N_c\}$), rather than continuous as was the case in regression, where $y \in \mathbb{R}$. If we define $\hat{y} = f_\theta(x)$, the goal in a binary classification task is for the model to output $\hat{y} = p(y = 1|x)$.

Would the MSE loss still be an appropriate choice for classification tasks? To address this question, we look at what assumptions underlie the MSE loss function. Let us consider the scenario where we have the following non-linear relationship between the features x_i and labels y_i of the dataset, i.e.

$$y_i = \phi(x_i) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is a zero-mean Gaussian random variable with variance σ^2. We can also equivalently write $y_i \sim \mathcal{N}(\phi(x_i), \sigma^2)$, which implies that

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\phi(x))^2}{2\sigma^2}},$$

that is, each label y_i is sampled from a normal distribution with mean $\phi(x_i)$ and variance σ^2.

The likelihood of y is therefore given by the product

$$p(y|x) = p(y_1|x_1)p(y_2|x_2)\ldots p(y_N|x_N) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\phi(x_i))^2}{2\sigma^2}} = \frac{1}{(\sqrt{2\pi\sigma^2})^N} e^{-\Sigma_{i=1}^{N}\frac{(y_i-\phi(x_i))^2}{2\sigma^2}}.$$

Thus, we have

$$\log p(y|x) = -\left[\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \phi(x_i))^2 + \frac{N}{2}\log(2\pi\sigma^2)\right],$$

from which we can see

$$\arg\max \log p(y|x) = \arg\min \sum_{i=1}^{N}(y_i - \phi(x_i))^2.$$

We therefore see that minimisation of the MSE loss is equivalent to maximum likelihood estimation under a Gaussian prior [7]. This explains why the MSE loss is not suitable for classification. In classification tasks, the label distribution is not normally distributed around a mean value, rather, the labels follow a discrete distribution. In the case of binary classification for instance, the possible label values are zero and one.

When designing a model to perform classification, we would like the outputs of our model to correspond to probabilities. For example, if we have ten classes, we would design a model with ten output nodes and we would want each node output to be a positive number between zero and one, expressing the probability of the input of the model belonging to the corresponding class. To convert the outputs to probabilities, we pass them through what is called a softmax function. The softmax is related to the sigmoid function presented earlier and in fact for binary classification tasks, using the softmax and the sigmoid are equivalent (if we assume we have two output nodes). In the case of binary classification though, it can be shown that one output node is sufficient.

Denoting the number of classes by K, the softmax function is defined as

$$p(y=k|x) = \frac{\exp(f_{\theta,k}(x))}{\sum_{i=1}^{K}\exp(f_{\theta,k}(x))},$$

where $f_\theta(x) \in \mathbb{R}^K$ and $f_{\theta,k}(x)$ the output of the k^{th} node.

Once the model outputs have been converted into probabilities, the loss function most commonly used for classification tasks is what is called the cross-entropy loss given as

$$L_{CE}(\theta) = -\sum_{i=1}^{N} p^*(y_i|x_i)\log p(\hat{y}_i|x_i,\theta),$$

where $p^*(y_i|x_i)$ is the true class distribution (given by the labels for training example x_i) and $p(\hat{y}_i|x_i,\theta)$ the predicted class distribution for input x_i (obtained through taking the softmax of model output $\hat{y}_i = f_\theta(x_i)$).

To motivate the use of the cross entropy loss, we will show that minimising it, is equivalent to minimising the Kullback-Leibler (KL) divergence between the target (label) distribution and the output distribution. The KL divergence between p^* and p is defined as

$$\begin{aligned} D_{KL}(p^*,p) &= \sum_{i=1}^{N} p^*(y_i|x_i)\log\frac{p^*(y_i|x_i)}{p(y_i|x_i,\theta)} \\ &= \sum_{i=1}^{N} p^*(y_i|x_i)\log p^*(y_i|x_i) - \sum_{i=1}^{N} p^*(y_i|x_i)\log p(y_i|x_i,\theta). \end{aligned}$$

It can therefore be concluded that

$$\arg\min_{\theta} D_{KL}(p^*, p) = \arg\min_{\theta} L_{CE}(\theta),$$

which motivates the use of the cross-entropy loss function for classification tasks.

Backpropagation

The goal of the backpropagation algorithm is to compute the derivatives of the loss function with respect to the model parameters. Once we have these derivatives, we can use them to update the model parameters with the goal of minimising the loss function. Let us start with the forward pass/propagation equations presented in section 2.1. We recall that the forward pass equations used to map the input $a^{(0)}$ to the output $a^{(L)}$ are given by

$$\begin{aligned} z^{(l)} &= w^{(l)} a^{(l-1)} + b^{(l)}, \\ a^{(l)} &= \sigma(z^{(l)}), \end{aligned}$$

for $l = 1, \ldots, L$. Given that the loss $L(\theta)$ directly depends on the model output $a^{(L)} = f_\theta(a^{(0)})$, which in turns directly depends on the last layers parameters, we need to start the gradient computation from the final layer, and progressively work our way back to the first layer, hence the name "backpropagation". The idea is to successively compute

$$\frac{\partial L}{\partial a^{(L)}}, \frac{\partial L}{\partial z^{(L)}}, \frac{\partial L}{\partial w^{(L)}}, \frac{\partial L}{\partial b^{(L)}}, \ldots, \frac{\partial L}{\partial a^{(1)}}, \frac{\partial L}{\partial z^{(1)}}, \frac{\partial L}{\partial w^{(1)}} \text{ and } \frac{\partial L}{\partial b^{(1)}}.$$

Consider a fully-connected feed-forward neural network with L layers. Assuming the same activation function σ is used for all the layers and choosing $L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2 = (a^{(L)} - y_i)^2$, we have

$$\begin{aligned} \frac{\partial L}{\partial a^{(L)}} &= \frac{2}{N}\sum_{i=1}^{N}(a^{(L)} - y_i), \\ \frac{\partial L}{\partial z^{(L)}} &= \frac{\partial L}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}} = \frac{\partial L}{\partial a^{(L)}} * \sigma'(z^{(L)}), \\ \frac{\partial L}{\partial w^{(L)}} &= \frac{\partial L}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}} = \frac{\partial L}{\partial z^{(L)}} a^{(L-1)}, \\ \frac{\partial L}{\partial b^{(L)}} &= \frac{\partial L}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial b^{(L)}} = \frac{\partial L}{\partial z^{(L)}}, \\ \frac{\partial L}{\partial a^{(L-1)}} &= \frac{\partial L}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = \frac{\partial L}{\partial z^{(L)}} w^{(L)}, \end{aligned}$$

The equations for backpropagation are therefore

$$\frac{\partial L}{\partial z^{(l)}} = \frac{\partial L}{\partial a^{(l)}} * \sigma'(z^{(l)}),$$

$$\frac{\partial L}{\partial w^{(l)}} = \frac{\partial L}{\partial z^{(l)}} a^{(l-1)},$$
$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial z^{(l)}},$$
$$\frac{\partial L}{\partial a^{(l-1)}} = \frac{\partial L}{\partial z^{(l)}} w^{(l)}.$$

Once the derivatives of the loss with respect to the parameters (weights and biases) of each layer have been computed, we can use this information to update the model parameters. The update rule for the parameters depends on the optimiser we are using. We presented some of the most popular optimisers in Chapter 1.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [66] are a type of neural network architecture primarily used for computer vision problems. In order to pass an image as an input to a fully-connected neural network, the image would have to be "flattened", i.e. turned into a one-dimensional vector. This would lead to the loss of important information regarding spatial relationships (connectivity) between pixels, while also increasing the number of parameters of the model. As an example, consider a 1000×1000 grayscale image that we wish to pass into a fully-connected architecture. After flattening the image, the length of the input vector to the network will be equal to 10^6, that is we would have one million input nodes. If we further assume that the first layer of the network has 1000 neurons, the number of weights/connections between the input and first layer alone would be $10^6 \times 1000 = 1$ billion parameters.

CNNs address this issue of overparameterisation, while at the same time preserving the two-dimensional structure of the image. Let us assume the input image is a greyscale image and therefore only has one channel (for an RGB image, the number of channels would be $C = 3$). Let us also consider a convolutional layer with a learnable filter/kernel (the size of the kernel has to be smaller than that of the input image). The convolution operation between the filter and input channel involves the element-wise multiplication between elements in the filter and the corresponding elements in the image and then summation of the elements of the resulting matrix. After this summation is completed, the filter is subsequently shifted along the image with a predetermined stride/step. At each position, the result of the convolution is stored in the corresponding entry of the output feature map.

The number of channels of the filter must match the number of channels of the input to the convolutional layer. A convolution operation between a filter and an input of any depth/number of channels, results in an output feature map with a single channel. If a convolutional layer has f filters, the number of channels of the output of the convolutional layer will also be f.

CNNs also make use of what are called “pooling” layers. Pooling can be used inside a CNN for dimensionality reduction of the representation. The most commonly used types of pooling are average and max pooling. When the average pooling filter is applied on an image region the result is a single value, corresponding to the average of the values of the filtered image elements in that region. In the case of max pooling, only the maximum element of the corresponding image region is kept.

Residual Neural Networks

Several theoretical studies [43, 60, 106, 133–135] have demonstrated the benefit of deep neural networks over their shallow counterparts. Intuitively, having more layers, allows the network to distribute feature extraction tasks across layers [151], with earlier layers often learning more general features such as learning how to detect edges and later layers learning more high level, abstract features. A deep architecture allows each layer to “build” on information from previous layers, while a shallow one does not have that flexibility. Adding another layer to a network adds another composition. Keeping the number of layers fixed and increasing the width of the network can also increase model expressivity but is not as powerful as increasing the number of layers/compositions.

Deep convolutional neural networks [64, 67] have be shown to yield significant performance improvements on image classification tasks [45, 117, 121, 132, 151] over shallower architectures. Despite the advantage of increased expressivity, training deep neural networks can present a significant challenge [95, 104]. Two of the most common issues encountered during training of deep neural networks are the problem of “vanishing” and “exploding” gradients [8, 104]. The backpropagation algorithm uses the gradients in layer l to compute gradients in each layer $l-1$. This means that there is a series of multiplications (due to the chain rule) involved in gradient computations. If the numbers being multiplied are small, this can lead to smaller and smaller (vanishing) gradients the further back we go into the network. Similarly, if the numbers multiplied are quite large, consecutive multiplications can lead to very large (exploding) gradients as we go back deeper and deeper into the layers. The issue of vanishing and exploding gradients can be addressed through appropriate weight initialisation as well as batch normalisation [36, 68, 115]. Another issue observed during the training of deep networks [45], was that despite increased flexibility and number of parameters deep neural networks seemed to have higher train as well as test error than their shallow counterparts [44, 126].

Residual neural networks (RNNs) [45] were introduced to address this issue in training deep neural networks. The main idea behind RNNs is the introduction of what are called “skip connections”. Skip connection allow for the input to a “block” of layers to bypass that block. The input is then directly added to the blocks output. The reason for that is that without the skip connctions, information / signal tends to get distorted within deep networks due to the multiple layers of operations performed on it. For a shallow network, the transformation

applied to the input to get to the output is relatively straightforward, whereas for deep NNs, the input signal goes through multiple transformations that distort the information contained in the input. Residual networks are convolutional neural networks, but their layers are grouped into “blocks”. The input to each block is both propagated through the block but also a copy of it “skips” the block through the skip connection and gets added directly to the output of the block.

Transformers

Transformers are a neural network architecture originally introduced in [139] that is particularly suitable for working with sequential data (where the order of the elements is crucial for understanding the information the data conveys). Although originally introduced to deal with Natural Language Processing (NLP) tasks, such as machine translation and sequence transduction (conversion of an input sequence into an output sequence) in general, transformers have also found applicability in other types of machine learning problems, such as computer vision tasks [22]. Transformers employ a mechanism called “Self-attention”. This mechanism is used to weigh the importance of different parts of the input when processing information. When dealing with text input for example, self-attention allows the model to understand the importance of different words in a sentence.

Transformers consist of an encoder and decoder part. The encoder processes input data and the decoder is responsible for generating the output. BERT [21], is a transformer-based language model developed by Google as a text-understanding tool. DistilBERT [112], is a distilled (with fewer parameters) version of BERT which makes training significantly faster at a small the cost in terms of accuracy. We will be using DistilBERT for all NLP experiments presented in Chapter 4.

Stochastic gradients

As we saw in Chapter 1, optimisation algorithms such as momentum gradient descent and Adam, require access to the gradients of the loss function at each optimisation step. Let us assume we have a dataset of size N and want to minimise a loss function defined as

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l_i(\theta),$$

where $l_i(\theta)$ the individual loss for each data-point (x_i, y_i). Differentiating the loss function L, we obtain

$$\nabla L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla l_i(\theta).$$

We see that computing the full gradient $\nabla L(\theta)$, requires the computation of the gradient $\nabla l_i(\theta)$ at each data point. Given that we have N data points, this means that computing $\nabla L(\theta)$ would involve N forward passes as well as N backpropagation steps (one forward and one backward pass for each datapoint). The cost of this can become prohibitive when N is very large, i.e. for large datasets.

An approach towards reducing the cost of computing the gradients, is to use an unbiased stochastic approximation of the full gradient [108], instead of $\nabla L(\theta)$ itself. Rather than use the full dataset of size N to compute the gradient, we use a subset of the data of size $b \in \mathbb{N}$, where $b << N$. The quantity b is called the "batch size" and represents the number of samples/datapoints we use to compute the stochastic approximation of the gradient. The first step towards this goal is to randomly sample a set of datapoints to use for gradient computation. We consider a set ω, of independent, identically distributed (i.i.d.) random variables, such that each ω_i is a randomly selected element from the set of indices $S = \{1, .., N\}$. We draw b elements from S and therefore have $\omega = \{\omega_1, \omega_2, \ldots, \omega_b\}$. Each ω_i is therefore a number selected from S to be used as an index for the selection of data points. We proceed to define an unbiased estimator $G : \mathbb{R}^d \times \{1, ..., N\}^b \to \mathbb{R}^d$ of the full-gradient as

$$G(x, \omega) := \frac{1}{b} \sum_{i \in \omega} \nabla l_i(x)$$

is constructed such that $\mathbb{E}[G(\cdot, \omega)] = \nabla L(\cdot)$.

Apart from often reducing computational time (see [14, 53, 109]), using stochastic gradients can lead to enhanced exploration of the loss landscape and can make it easier to escape saddle points [18, 35] and local minima, due to the introduction of noise into the training process. This enhanced exploration can ultimately lead to the discovery of better local minima, which in turn results in improved generalisation [153].

Chapter 3

Friction-adaptive Descent: A Family of Dynamics-based Optimisation Methods

3.1 Introduction

The work presented in this chapter is based on the following published paper [58]. We describe a family of descent algorithms which generalizes common existing schemes used in applications such as neural network training and more broadly for optimisation of smooth functions–potentially for global optimisation, or as a local optimisation method to be deployed within global optimisation schemes like basin hopping. By introducing an auxiliary degree of freedom we create a dynamical system with improved stability, reducing oscillatory modes and accelerating convergence to minima. The resulting algorithms are simple to implement and control, and convergence can be shown directly by Lyapunov's second method.

Although this framework, which we refer to as friction-adaptive descent (FAD), is fairly general, we focus most of our attention here on a specific variant: kinetic energy stabilization (which can be viewed as a zero-temperature Nosé–Hoover scheme but with added dissipation in both physical and auxiliary variables), termed KFAD (kinetic FAD). To illustrate the flexibility of the FAD framework we consider several other methods. In certain asymptotic limits, these methods can be viewed as introducing cubic damping in various forms; they can be more efficient than linearly dissipated Hamiltonian dynamics in common optimisation settings.

We present details of the numerical methods and show convergence for both the continuous and discretized dynamics in the convex setting by constructing Lyapunov functions. The methods are tested using a toy model (the Rosenbrock function). We also demonstrate the methods for structural optimisation for atomic clusters in Lennard–Jones and Morse potentials. The experiments show the relative efficiency and robustness of FAD in comparison to linearly dissipated Hamiltonian dynamics.

We study the design of extended systems for exploring energy (or loss) landscapes in high dimensions. We suppose that we are given some objective function $f : \mathbb{R}^d \to \mathbb{R}$, differentiable and bounded below, and the goal is to find a local (or possibly global) minimum of f. To facilitate analytical study, we will assume that the function f is convex, so that the minimizer is unique. Ultimately we expect such local minimization strategies to be deployed as part of more complicated global optimisation procedures or as part of stochastic optimisation methods.

There is a wide body of literature available on optimisation methods (see e.g. [11, 96]), but we focus on the narrower class of iterative methods which are constructed by discretization of continuous dynamical systems. In this work we assume that the gradient of f is tractable, even if costly to evaluate. Recall from Chapter 1 that a standard iterative scheme for finding local minima of f is gradient descent [93], which generates the sequence of iterates $x_1, x_2, \ldots$, beginning from a given initial condition x_0, via

$$x_{n+1} = x_n - \delta t \nabla f(x_n).$$

This method can be viewed as the Euler discretization of the continuous dynamics $\dot{x} = -\nabla f(x)$. Gradient descent is an effective optimisation strategy in many situations. Combined with the concept of stochastic gradients from Bayesian data analysis, it is widely used in deep learning (see for instance [40, Chapter 8]). However, it has limitations including slow convergence and instability (for large values of δt). One way to increase the flexibility of iterative methods is to add a vector of momenta ($p \in \mathbb{R}^d$), that is to treat the extended system with state described by (x, p) in which successive iterates are generated from

$$(x_{n+1}, p_{n+1}) = T_{\delta t}(x_n, p_n),$$

where $T_{\delta t}$ represents an approximation of the time δt flow map of some suitable system. For example $T_{\delta t}$ might be obtained as a numerical discretization of

$$\frac{\mathrm{d}x}{\mathrm{d}t} = p, \tag{3.1}$$

$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\nabla f(x) - \gamma p. \tag{3.2}$$

Here $\gamma > 0$ is a parameter which ensures that $p \to 0$ over time and which ultimately guarantees the approach to a local minimum of f. This idea of extending the system is very natural by reference to Newtonian mechanics. Define the energy $H(x, p) = \|p\|^2/2 + f(x)$ (where $\|p\|^2/2$ represents the kinetic energy and $f(x)$ the potential energy), then (3.1)-(3.2) represents a physical system subject to linearly dissipative forces, i.e. mechanical damping. Upon introducing finite differences into (3.1)-(3.2) one arrives at various discrete procedures.

In the optimisation community, this is typically referred to as Polyak's heavy ball method [107], although we here use the term linearly dissipated Hamiltonian descent (LDHD). A variant of this approach, corresponding to a particular discretization of (3.1)-(3.2), is termed Nesterov accelerated gradient [94].

Since the goal of extending the gradient dynamics $\dot{x} = -\nabla f(x)$ by introducing new variables is to solve the optimisation problem (or obtain new candidate optimisation schemes), there is nothing that anchors us to physically motivated dynamics. It is natural to further generalize the framework in various ways. The only essential criteria are the following:

1. correctness – the extended system should have accessible mathematical properties and allow for a thorough convergence analysis;
2. efficiency – the extended system should be easy to implement and computationally tractable in the sense of adding minimal overhead in the computation of a timestep compared to, say, gradient descent, and not less robust in practice;
3. value – at the end of the day, there should be some tangible benefit from the added complexity of the extension.

A family of optimisation methods of this type was recently studied in [82], including the use of more general kinetic energies $k(p)$ and models for dissipation $D(p)$, resulting from discretization of

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \nabla k(p), \tag{3.3}$$

$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\nabla f(x) - D(p). \tag{3.4}$$

One of the key outputs of the study of (3.3)-(3.4) is that, by carefully choosing k as a symmetrization of the convex conjugate of f while keeping the usual friction $D(p) = p$, a geometric convergence rate can be obtained for strongly convex and smooth functions f, with a convergence rate which does not depend on the upper/lower bounds on the Hessian. The convergence result can in fact be extended to non-smooth or non-strongly convex functions, and to discretizations of (3.3)-(3.4). Motivation for general kinetic energies comes from works like [128] which have shown potential for acceleration in the sampling context.

The reasons for using the straightforward generalization (3.3)-(3.4) are related to the criteria 1-3. It is easy to construct Lyapunov functions for (3.3)-(3.4), numerical implementation can be performed easily, and there are some clear wins–examples where this framework proves beneficial. However, the choice of $D(p)$ is tied to $D(p) = \gamma p$ in [82] in order to guarantee a convergence rate independent of the strong convexity and smoothness parameters of f (see in particular Remark 1 there). The linear dissipation case corresponds to the class of models

known as “conformal Hamiltonian systems" which implies a geometric principle underpinning the flow of the system [9, 86]. While this property is not extensively utilized in optimization treatments, the reference [32] explores it in that context. All numerical experiments in [82] focused on $D(p) = \gamma p$.

Another way of generalizing LDHD is to consider the damping coefficient as a variable parameter. An example of this is the method of Su, Boyd and Candes [129], which generalizes Nesterov descent by replacing γ with $\gamma(t) \propto t^{-1}$. A recent work of Moucer, Taylor and Bach [90] demonstrated convergence for such methods using Lyapunov functions. There have also been a few approaches introduced in the machine learning literature (see, e.g., [130]) to determine what they term the ‘momentum parameter’, akin to adjusting γ during the descent process. From our perspective, these schemes are heuristic and sacrifice the intuitive foundation of the Polyak model.

In this chapter, we consider generalizations that go beyond (3.3)-(3.4) by introducing additional variables along with nonlinear negative feedback loop control laws. This framework is suggested in work of Maxwell [84] and Wiener [147], but the methods studied here are more directly related to the Nosé–Hoover sampling methods [47, 97, 98] that are widely used in molecular modelling. These are ‘thermostats’ that regulate the kinetic energy of system. In the case of a standard Hamiltonian system, the Nosé–Hoover thermostat introduces an adaptive friction coefficient

$$\frac{\mathrm{d}x}{\mathrm{d}t} = p, \tag{3.5}$$

$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\nabla f(x) - \xi p, \tag{3.6}$$

$$\frac{\mathrm{d}\xi}{\mathrm{d}t} = \|p\|^2 - \beta^{-1}, \tag{3.7}$$

where β is a parameter (which can be viewed as a “reciprocal temperature”). In contrast to some other frequently encountered extended systems (e.g. generalized Langevin equation [127]) the coupling between auxiliary variables and physical ones in (3.5)-(3.7) is *nonlinear* which potentially complicates both numerics and analysis.

It can be shown that (3.5)-(3.7) preserves the probability measure with density

$$\rho(x,p,\xi) = Z^{-1} \exp\left(-\beta\left[\|p\|^2/2 + f(x) + \xi^2/2\right]\right),$$

(see [70]), where Z is a normalization constant so that $\int_{\mathbb{R}^{2d+1}} \rho(x,p,\xi)\mathrm{d}x\,\mathrm{d}p\,\mathrm{d}\xi = 1$. This probability measure can be interpreted as an extended canonical ensemble in statistical physics. In the case where the system is ergodic, the invariant probability measure is unique and the law of the process converges to it in the long time limit. The configurational marginal

of ρ is proportional to $\exp(-\beta f(x))$, thus the extended system allows sampling of a target canonical distribution. It is intuitively apparent that when β is large ("low temperature") the system will be driven to the vicinity of a local minimum. One may also see the direct control of the kinetic energy as a potentially desirable stabilizing feature during optimisation.

When (3.7) is augmented by linear dissipation and stochastic forcing, the resulting Nosé–Hoover–Langevin system [71] also can be shown to converge to the extended canonical state. The thermostatting idea can be generalized in a variety of ways, for example incorporating coordinate-dependent projections in the coupling between auxiliary variables and physical ones [52], an idea that we exploit later in this chapter. It can also be shown that adding noise and linear dissipation in the physical variables (x, p) to the Nosé–Hoover thermostat maintains the canonical state [54]. Despite the nonlinearity, numerical integration of the Nosé-Hoover system and its generalizations can be performed using straightforward splitting methods. Moreover, theoretical analyses are now available for various forms of Nosé dynamics (see, for example, [73] for a hypercoercivity analysis of the Adaptive Langevin method, or [46] for a Lyapunov analysis). Drawing on these works, we explore models that incorporate thermostat-type control laws which can help to restrict or guide the approach to minima. In contrast to existing works on thermostatting, we include linear dissipation in both physical and auxiliary variables.

While there are many ways to embed a given optimisation problem in a higher dimensional dynamical framework, our aim here is to keep the iteration steps as simple as possible, involving only explicit calculations and simplifying convergence analysis. This leads us to consider methods which can be viewed as simple modifications of Hamiltonian dynamics:

$$\begin{aligned}
\dot{x} &= p, \\
\dot{p} &= -\nabla f(x) - \xi \Phi(x,p) - \gamma p, \\
\dot{\xi} &= p \cdot \Phi(x,p).
\end{aligned}$$

Computing the orbital derivative of the extended Hamiltonian

$$\tilde{H}(x,p,\xi) = \|p\|^2/2 + f(x) + \xi^2/2$$

with respect to this system results in

$$\frac{d}{dt}\left[\tilde{H}(x,p,\xi)\right] = -\gamma\|p\|^2 - \xi\, p\cdot\Phi + \xi\, p\cdot\Phi = -\gamma\|p\|^2.$$

Thus, if $\gamma > 0$,

$$\frac{d}{dt}\left[\tilde{H}(x,p,\xi)\right] \leq 0,$$

with equality only when $p = 0$. The function $\tilde{H}$ is thus a weak Lyapunov function. If Φ is such that $\Phi(x,0) = 0$, then any state of the form $(x,0,\xi)$ with $\nabla f(x) = 0$ is a critical point of the dynamical system.

The convergence here appears to rely on the linear dissipation in the physical momentum variable. However, in many cases the method will be convergent even with $\gamma = 0$. To see this, consider the choice $\Phi(x,p) = p$, so with $\gamma = 0$ we have standard Nosé–Hoover dynamics at zero temperature. Assuming $\xi(0) > 0$, we have that ξ is monotone increasing and remains positive, since $\dot{\xi} = \|p\|^2 \geq 0$. The monotonicity is undesirable (it adds "stiffness", for example) and motivates the addition of friction in the equation for the auxiliary variable. The addition of linear dissipation in p ($\gamma > 0$) is not always essential, but it helps to regularize the method in case of a degeneracy in the control law, as we shall see later; it also makes possible the Lyapunov stability analysis we use here to show exponential convergence of both the dynamics and discretization.

As a numerical illustration, we show the result of applying the kinetic energy-based method (termed KFAD) in the case of a system with harmonic potential $f(x) = x^T Cx/2$ where $C = \mathrm{diag}(1,10)$. (We defer presenting the details of the numerical methods in order to focus on the dynamical aspects; numerics will be addressed in Section 3.4. For these graphs, we initialized the system at $(x_1(0),x_2(0)) = (1,2)$ and have used a timestep $\delta t = 0.01$ which was small enough to ensure that the figures are illustrative of the properties of the dynamics, not the numerical method, per se.) A friction value of $\gamma = 1$ was used in both methods.

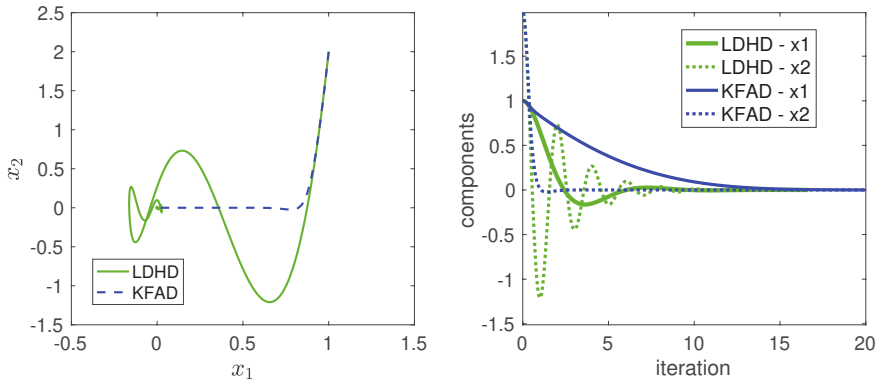


Figure 3.1: The convergence of the trajectories of the linearly dissipated Hamiltonian dynamics (green) compared with that of KFAD (blue). On the right, the graphs of the individual components are shown as functions of time.

As we can see from Fig. 3.1, the extended system has largely removed the oscillatory "pre-equilibrium" dynamics, in particular the strongest part of the oscillation which is due to the second component. In these tests the stopping criterion was a simple tolerance test based on the distance to the (known) minimum $\|x - x_{\min}\| \leq 10^{-4}$. The KFAD scheme used here took

around 25% more timesteps to reach the minimum (2085 vs. 1663), but as we explain later, the situation is often reversed in nonlinear systems, and more powerful methods designed using thermostats with position-dependent coupling can help to damp strong oscillations in stiff optimisation problems while also providing excellent efficiency gains.

The rest of this chapter is organised as follows. In Section 3.2 we present a family of extended systems, motivate various choices in its definition, and discuss its basic properties. Section 3.3 studies convergence of the dynamical system through construction of Lyapunov functions. Section 3.4 provides numerical discretisations, and the convergence of discretisation schemes is taken up in Section 3.5. Finally, we present numerical experiments with several variants of the FAD scheme in Section 3.6.

3.2 Friction-adaptive descent (FAD)

We restrict to methods that have dynamical equations that are easy to solve by common discretization methods such as splitting. A useful class is defined by

$$\Phi(x,p) = A(x)p, \tag{3.8}$$

for some symmetric matrix function $A = A(x)$. We also include linear damping in the equation for ξ and we introduce a coefficient μ to control the coupling between momenta and auxiliary variable. The equations then become

$$\dot{x} = p, \tag{3.9}$$

$$\dot{p} = -\nabla f(x) - \xi A(x)p - \gamma p, \tag{3.10}$$

$$\dot{\xi} = \mu^{-1} p^T A(x) p - \alpha\xi. \tag{3.11}$$

The inclusion of friction in the equation for ξ is reminiscent of Nosé–Hoover–Langevin [71], but we do not currently propose to incorporate additive noise (or consider gradient noise here), thus (3.9)-(3.11) is a deterministic system of ODEs. Without the dissipation term $-\alpha\xi$, the control law for ξ would be $\dot{\xi} = \mu^{-1} p^T A(x) p$. Assuming a symmetric positive semidefinite matrix $A(x)$, we have $\dot{\xi}(t) \geq 0$ for all $t \geq 0$, leading to a non-decreasing adaptive friction ξ. Adding the dissipation term allows the friction to both dynamically increase and decrease. A dissipation term also adds a "short-term memory" to the coupling between ξ and p, similarly to how the dissipation term $-\gamma p$ in (3.2) introduces momentum ("memory" of recent states) in Hamiltonian dynamics, leading to the dynamics of momentum gradient descent. This notion of taking history into account when designing control laws was introduced in the work of Maxwell [84] on governors, where he observed that in order to achieve speed regulation with

zero steady state error, the integral (history) of the speed error must be minimized. Finally, adding the dissipation term $-\alpha\xi$, ensures rigorous convergence to the equilibrium. Without this term there would be an infinite number of equilibrium points, each with a different value of ξ.

We are interested in methods defined by various choices of A. We describe some examples below. There are certainly many more options than these but we leave their study for future work.

3.2.1 Kinetic friction-adaptive descent (KFAD)

In case $A = I$ we have the Nosé–Hoover dynamics at zero temperature, supplemented by linear damping in both p and ξ. The distinguishing feature here is that the FAD coupling matrix $A = I$ is itself non-degenerate.

If $\alpha > 0$, we can see that asymptotically, near equilibrium, $\xi \sim \alpha^{-1}\mu^{-1}p^T Ap = \alpha^{-1}\mu^{-1}\|p\|^2$. Inserting this into (3.10), we have

$$\dot{p} = -\nabla f(x) - \xi p - \gamma p \sim -\nabla f(x) - \frac{1}{\alpha\mu}\|p\|^2 p - \gamma p.$$

This can be viewed as a combination of cubic damping and linear damping and is described by a generalized (quartic) Rayleigh dissipation function [39, 140]:

$$R(p) = \frac{1}{4\alpha\mu}\|p\|^4 + \frac{\gamma}{2}\|p\|^2.$$

In the case of KFAD, the nonlinear damping is isotropic, depending only on $\|p\|$. Cubic damping is commonly used as a vibration suppression mechanism in mechanical systems [100, 155] and was studied extensively in the scalar case by A. Babister [6]. We stress though that we use the freedom of the auxiliary variable to create a more flexible dynamics. The friction parameter α cannot be assumed to be large, so the relaxation to the equilibrium of the ξ equation is likely to be relevant to the overall behavior; as we shall see in the numerical experiments we generally found modest values of α ($\alpha \approx 1$) to be most useful in practice.

3.2.2 Force-based friction-adaptive descent (FFAD)

A further generalization is to allow A in (3.8) to be indefinite. For example one could use $A = F(x)F(x)^T$ where $F(x) = -\nabla f(x)$ is the force vector.

Assuming α to be sufficiently large, the auxiliary variable will relax rapidly to its equilibrium obtained by setting the right hand side of (3.11) to zero, that is

$$\xi \approx \frac{1}{\alpha\mu}(p \cdot F(x))^2.$$

The system (3.9)-(3.10) then becomes

$$\dot{x} = p, \tag{3.12}$$

$$\dot{p} = F(x) - \frac{1}{\alpha\mu}(p \cdot F(x))^2 F(x)F(x)^T p - \gamma p. \tag{3.13}$$

We can derive the damping forces from the framework of generalized Rayleigh dissipation [140]. Specifically, define

$$\begin{aligned} R(x,p) &= \frac{1}{4\alpha\mu}(p \cdot F(x))^4 + \frac{\gamma}{2}\|p\|^2 \\ &= \frac{1}{4\alpha\mu}\left(p^T F(x)F(x)^T p\right)^2 + \frac{\gamma}{2}\|p\|^2. \end{aligned}$$

Then

$$\nabla_p R(x,p) = \frac{1}{\alpha\mu}\left(p^T F(x)F(x)^T p\right) F(x)F(x)^T p + \gamma p,$$

so the damping forces in (3.13) are of the required form, i.e. $-\nabla_p R(x,p)$. It follows that, with $H(x,p) = \|p\|^2/2 + f(x)$ the Hamiltonian of the system,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}H &= -p \cdot \nabla_p R(x,p) \\ &= -\frac{1}{\alpha\mu}(p^T F(x)F(x)^T p)^2 - \gamma\|p\|^2. \end{aligned}$$

Note that (3.13) can be rewritten as

$$\dot{p} = F_{\text{eff}}(x) - \gamma p,$$

where the "effective" force is given by

$$F_{\text{eff}}(x) = \left(1 - \frac{1}{\alpha\mu}(p \cdot F(x))^3\right) F(x). \tag{3.14}$$

When $p \cdot F(x) > 0$, i.e. when we are moving in the direction of $F(x)$, for example when we are descending towards a minimum, then $F_{\text{eff}}(x) = aF(x)$ with $a < 1$. When $p \cdot F(x) < 0$, then $F_{\text{eff}}(x) = aF(x)$ with $a > 1$, so in the case where we are moving in a direction opposite to the force, the effective force experienced by the particle increases, pulling the particle back in the direction of the force. This works as an oscillation suppression mechanism. We see that the same mechanism that suppresses vibrations also slows down descent towards the minima. Examining the dynamics (3.12)-(3.13) can therefore help us explain the behaviour of FFAD.

The cubic damping in (3.13) acts to rapidly diminish $p \cdot F$, that is, the component of the momentum in the steepest descent direction. To demonstrate this in practice, we investigate the FFAD method for the Rosenbrock function given by

$$f_R(x_1, x_2) = (a - x_1)^2 + b(x_2 - x_1^2)^2, \qquad a = 1,\ b = 100.$$

This function has a unique global minimum $f(x^*) = 0$ at $x^* = (1, 1)$. The Hessian matrix of this potential has both steep and moderate components, thus we could call it a stiff optimisation problem. It is known that it presents challenges for most standard optimisation schemes.

We show below the results for a typical experiment comparing KFAD and FFAD with LDHD. For KFAD/FFAD we used coefficients $\mu = 1$ and $\alpha = 0.1$. For linear dissipation, present in all three methods, we used $\gamma = 1$. We started the integrator at $(1, 2)$, just above the minimum. The stepsize was $\delta t = 0.01$. For all methods we used splitting schemes in which various parts of the ODEs are integrated exactly; all details of these methods are presented in Section 4.

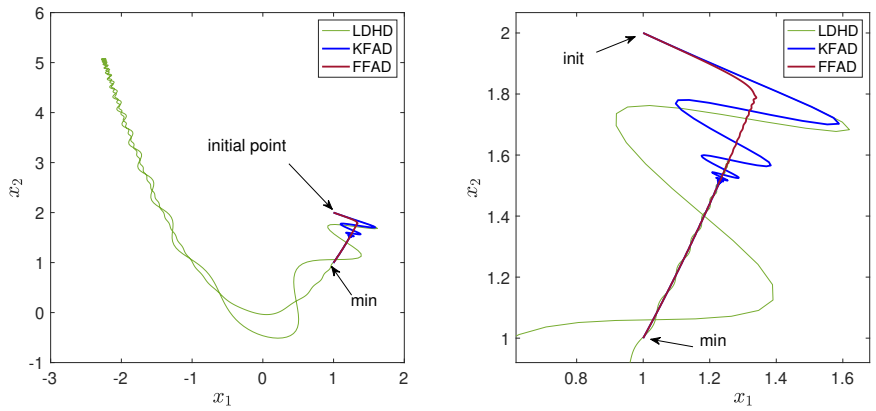


Figure 3.2: Comparison of performance of LDHD, FFAD and KFAD on the 2d Rosenbrock model, for initial condition $(1, 2)$. The second graph shows a close-up of motion in the vicinity of the minimum.

In this example, LDHD makes a rather unhelpful excursion into an extreme end of the valley (as far as the vicinity of the point $(-2, 5)$), before gradually wobbling back toward the minimum. KFAD moves rapidly into the narrow channel, and then, steadily approaches the minimum. The number of timesteps needed to reach the minimum (using the same criterion as before, $\|x - x_{\min}\| \leq 10^{-4}$) were as follows for the three methods: LDHD,1803; KFAD,1119; FFAD: 1447.

We re-ran the example with several different initial conditions. The results were qualitatively similar. Although the number of steps to reach the minimum varied considerably, FFAD removed nearly all oscillation outside the channel. KFAD on the other hand was typically as or more efficient than FFAD, and reduced but did not eliminate oscillation outside the channel. The result for a different initial condition, $(4, 2)$, is shown in Fig. 3.3.

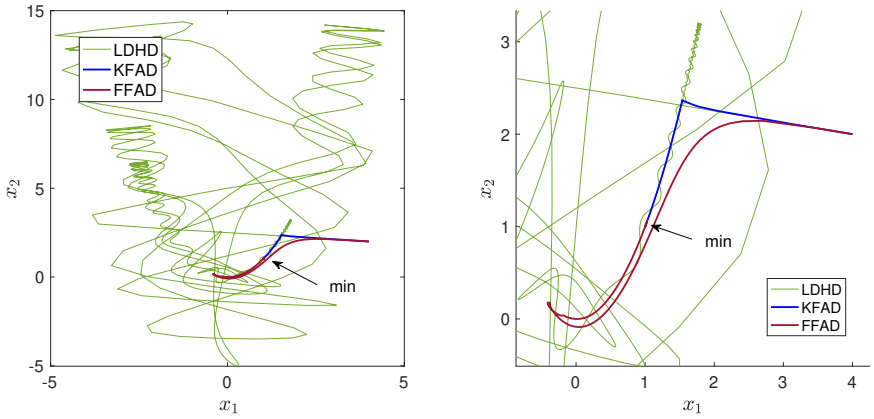


Figure 3.3: A second illustration of the relative performance of LDHD, KFAD and FFAD on the 2d Rosenbrock model, with initial condition $(4, 2)$. A close-up of the vicinity of the minimum is shown in the right panel.

In Fig. 3.3, the timesteps needed to reach the minimum were as follows: LDHD: 2010, KFAD: 1604, and FFAD: 3658. FFAD showed a tendency to overshoot the channel and then oscillate during the approach, or to slightly miss the channel and then follow it. The performance of KFAD here is remarkable – it completely removes the oscillatory initial phase. We will provide more detailed numerical studies on the Rosenbrock model in Section 3.4, but the results observed here are characteristic of the differences in the methods. KFAD outperforms LDHD for almost all initial conditions.

3.2.3 Projective mixture coupling

We attribute the somewhat erratic performance of FFAD (in terms of efficiency) to the degenerate nature of the friction coupling. We also recognize the potential for $\|F\|$ to be very large, which can lead to excessive variation in the friction coupling force. We therefore considered alternatives based on

$$A(x) = \lambda_1 I + \lambda_2 \Pi^F(x),$$

where $\Pi^F(x) = \|F(x)\|^{-2} F(x)F(x)^T$ is a projector onto the range of $F(x)$. The use of such a projector can be seen as a zero temperature version of "projective thermostatting" [52], albeit with a novel choice of the projection which has not been previously considered in the thermostat setting. We explore different choices of the parameters λ_1, λ_2 in the numerical experiments.

We illustrate the potential for mixture coupling (McFAD) on the Rosenbrock example, see Fig. 3.4. The first figure shows comparisons of the minimization curves; the second graphs the distance from the minimum.

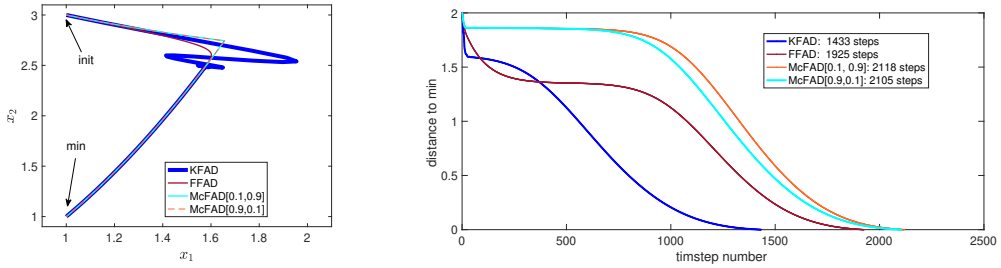


Figure 3.4: Left: convergence for mixture coupling with parameters $[\lambda_1, \lambda_2] = [0.1, 0.9]$ and with parameters $[0.9, 0.1]$ for the Rosenbrock function. Comparing to KFAD we see that the convergence is more direct, with essentially no oscillation. Right: we illustrate the convergence in terms of decay of the distance with time.

There are wide plateaus in the convergence of FFAD and mixture coupling which slow convergence. KFAD makes more rapid progress to the minimum, despite oscillations in the trajectories themselves.

3.3 Dynamical systems analysis of friction-adaptive descent

In this section, we briefly discuss the properties of the dynamics and introduce a Lyapunov function to show exponential convergence under certain assumptions on f and A, in particular that f is strongly convex and that A is symmetric positive semidefinite.

Before turning to the convergence theorem, we present some computed "phase portraits" of KFAD and FFAD (See Fig. 3.5). By phase portraits we mean projections of the solution curves of each system into some suitable plane (in this case, we consider the x_1x_2 projection of FAD in the case of the Rosenbrock function, where we assumed initial $\xi_0 = 0$). The parameter γ was fixed at 1 and α and μ were also taken to be 1. The stepsize for integration was fixed at 0.01. Points are generated on a grid with unit spacing in $[-4, 4] \times [-4, 4]$. In all cases every solution shown converges to $(1, 1)$.

Convergence to the minimum is achieved for all the initial conditions, although the way that convergence is achieved depends strongly on the initial point.

3.3.1 Elementary properties of friction-adaptive dynamics

With ξ defined as in (3.9)-(3.11), we have, from $\dot{\xi} = p^T A(x) p/\mu - \alpha\xi$, that

$$\xi(t) = \mathrm{e}^{-\alpha t}\xi(0) + \frac{1}{\mu}\int_0^t \mathrm{e}^{-\alpha(t-s)} p(s)^T A(x(s)) p(s)\,\mathrm{d}s. \tag{3.15}$$

In particular this shows that, when $\xi(0) \geq 0$ and A is everywhere symmetric positive semidefinite, then $\xi(t) \geq 0$ for all $t \geq 0$, thus ξ retains the interpretation of a friction coefficient.

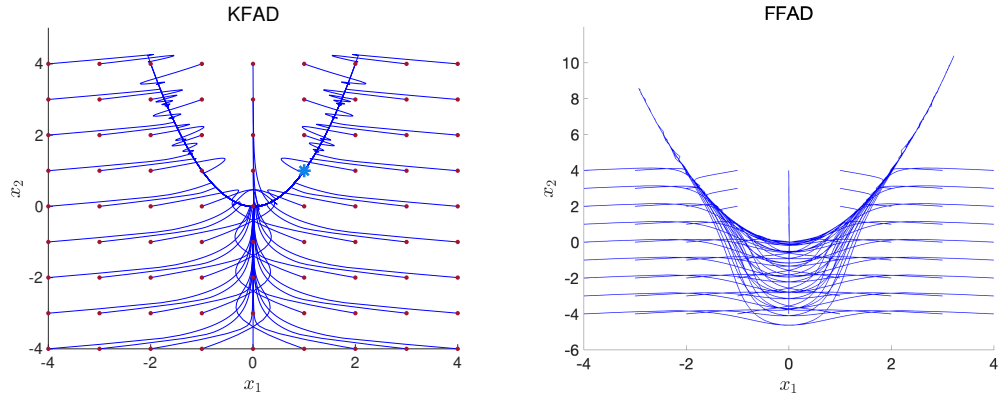


Figure 3.5: Phase portraits, as described in the text, show the convergence of each method from a number of different initial points (red points) in the x_1x_2 plane to the minimizer at $(1,1)$, marked as a blue star. Left: KFAD. Right: FFAD Note the different scale in the FFAD figure due to the long excursions along the narrow channel.

A first result is that the dynamics is well posed on infinite time horizons under some conditions on f and A.

Lemma 3.3.1. *Assume that $A(x)$ is symmetric positive semidefinite for all $x \in \mathbb{R}^d$, the function f is smooth and $f(x) \to +\infty$ as $\|x\| \to +\infty$. Then, for any initial condition $(x_0, p_0, \xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}_+$, the solution of* (3.9)-(3.11) *is well defined for all times $t \geq 0$, and there exists $R \in \mathbb{R}_+$ such that*

$$\forall t \geq 0, \qquad \|x(t)\| \leq R, \quad \|p(t)\| \leq R, \quad 0 \leq \xi(t) \leq R.$$

Proof. The Cauchy–Lipschitz theorem ensures the existence and uniqueness of a solution for a positive time. In order to show that the solution is global in time, we introduce the Lyapunov function

$$\mathscr{G}(x,p,\xi) = f(x) - f(x^*) + \frac{1}{2}\|p\|^2 + \frac{\mu}{2}\xi^2.$$

A simple computation shows that the function $\mathscr{G}(t) = \mathscr{G}(x(t), p(t), \xi(t))$ satisfies

$$\begin{aligned}\dot{\mathscr{G}}(t) &= \nabla f(x(t)) \cdot \dot{x}(t) + p(t)\dot{p}(t) + \mu\xi(t)\dot{\xi}(t) \\ &= -\gamma\|p(t)\|^2 - \alpha\mu\xi(t)^2 \leq 0,\end{aligned}$$

where we used that $\xi(t) \geq 0$. This shows that $\mathscr{G}(t) \leq \mathscr{G}(0)$, from which the result easily follows since $f - f(x^*)$ is nonnegative. □

The next result, whose proof is immediate, characterizes equilibria of the dynamics.

Proposition 3.3.2. *The equilibria of the system (3.9)-(3.11) correspond to*

$$(\dot{x}, \dot{p}, \dot{\xi}) = 0 \Rightarrow (p^*, F(x^*), \xi^*) = 0,$$

i.e. they coincide with the physical equilibria.

The last result relates LDHD and FAD.

Proposition 3.3.3. *Fix a time $\tau > 0$, and assume that the conditions of Lemma 3.3.1 are satisfied, and that A is bounded over compact sets. Then, in the limit $\alpha \to \infty$ and $\alpha\mu \to +\infty$, the solution to the system (3.9)-(3.11) converges uniformly over the time interval $[0,\tau]$ to the solution of linearly dissipated Hamiltonian dynamics.*

Proof. The equality (3.15) combined with the bound from Lemma 3.3.1 implies that there exists $K \in \mathbb{R}_+$ (depending on the initial condition) such that

$$0 \leq |\xi(t)| \leq \mathrm{e}^{-\alpha t}|\xi(0)| + \frac{K}{\mu}\int_0^t \mathrm{e}^{-\alpha(t-s)}\,\mathrm{d}s \leq \mathrm{e}^{-\alpha t}|\xi(0)| + \frac{K}{\alpha\mu}.$$

The two terms converge to 0 uniformly on bounded time intervals of the form $[t_{\min},\tau]$ for any $t_{\min} \in (0,\tau)$, respectively as $\alpha \to +\infty$ for the first one, and as $\alpha\mu \to +\infty$ for the second one. This bound is next combined with the integral formulation of the solution

$$p(t) = p(0) - \gamma\int_0^t p(s)\,ds - \int_0^t \xi(s)A(x(s))p(s)\,ds.$$

The last term converges to 0, which leads to the claimed convergence result since the system (3.9)-(3.10), with $\xi(t)$ replaced by 0, is precisely linearly dissipated Hamiltonian dynamics. □

The fact that FAD methods eventually inherit the dissipative property of LDHD is reassuring, but in practice this behavior may not be observed until very late in the descent process. As we hinted at earlier, the FAD dynamics, even without friction, can be convergent. We explore this property in the numerical experiments of Section 3.6, in the context of the Morse cluster.

3.3.2 Exponential convergence of the FAD dynamics

Under additional conditions on f, for instance f strongly convex, one can obtain the exponential convergence to the equilibria of Proposition 3.3.2.

Theorem 3.3.4. *Assume that $\gamma > 0$, that $A(x)$ is symmetric positive semidefinite for all $x \in \mathbb{R}^d$ and bounded over compact sets, and that there exist $a,b > 0$ such that*

$$a\left[f(x) - f(x^*)\right] + b\|x - x^*\|^2 \leq (x - x^*)\cdot\left(\nabla f(x) - \nabla f(x^*)\right). \tag{3.16}$$

Then, for any initial condition $(x_0,p_0,\xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}_+$, there exist $\kappa > 0$ and $C \in \mathbb{R}_+$ such that the solution of (3.9)-(3.11) *satisfies*

$$f(x(t)) - f(x^*) + \|p(t)\| + \xi(t) \leq C\mathrm{e}^{-\kappa t}.$$

The condition (3.16) is satisfied for $f \in C^2$ with $0 < m \leq \nabla^2 f(x) \leq M < +\infty$. A similar condition is considered in [83].

Proof. Consider $\varepsilon > 0$ and introduce the following Lyapunov function (which is, up to the additional term ξ^2, a common choice for stochastic Langevin dynamics [83], also considered for dissipated Hamiltonian dynamics [90]):

$$\mathscr{W}_\varepsilon(x,p,\xi) = f(x) - f(x^*) + \frac{1}{2}\|p\|^2 + \frac{\mu}{2}\xi^2 + \varepsilon(x-x^*)\cdot p + \varepsilon\|x(t)-x^*\|^2. \tag{3.17}$$

A discrete Cauchy–Schwarz inequality implies, for $\varepsilon \in [0,1/2]$, the lower bound

$$\mathscr{W}_\varepsilon(x,p,\xi) \geq f(x) - f(x^*) + \frac{1}{4}\|p\|^2 + \frac{\mu}{2}\xi^2 + \frac{\varepsilon}{2}\|x(t)-x^*\|^2, \tag{3.18}$$

as well as the upper bound

$$\mathscr{W}_\varepsilon(x,p,\xi) \leq f(x) - f(x^*) + \frac{3}{4}\|p\|^2 + \frac{\mu}{2}\xi^2 + \frac{3\varepsilon}{2}\|x(t)-x^*\|^2. \tag{3.19}$$

A simple computation shows that the function $\mathscr{W}_\varepsilon(t) = \mathscr{W}_\varepsilon(x(t),p(t),\xi(t))$ satisfies

$$\begin{aligned}
\dot{\mathscr{W}}_\varepsilon(t) &= \nabla f(x(t))\cdot\dot{x}(t) + p(t)\dot{p}(t) + \mu\xi(t)\dot{\xi}(t) + \varepsilon p(t)\cdot\dot{x}(t) \\
&\quad + \varepsilon\dot{p}(t)\cdot(x(t)-x^*) + 2\varepsilon\dot{x}(t)\cdot(x(t)-x^*) \\
&= -(\gamma-\varepsilon)\|p(t)\|^2 - \alpha\mu\xi(t)^2 \\
&\quad - \varepsilon(x(t)-x^*)\cdot\nabla f(x(t)) + \varepsilon(x(t)-x^*)\cdot[2-\gamma-\xi(t)A(x(t))]\,p(t) \\
&\leq -(\gamma-\varepsilon)\|p(t)\|^2 - \alpha\mu\xi(t)^2 - \varepsilon(x(t)-x^*)\cdot\nabla f(x(t)) \\
&\quad + \varepsilon(2+\gamma+R\|A(x(t))\|)\left(\delta\|x(t)-x^*\|^2 + \frac{1}{4\delta}\|p(t)\|^2\right),
\end{aligned}$$

where we used a Cauchy–Schwarz inequality and Lemma 3.3.1. Note that $\|A(x(t))\| \leq \mathscr{A} < +\infty$ since A is bounded on compact sets and $x(t)$ is uniformly bounded. We then set $\delta = b/[2(2+\gamma+R\mathscr{A})]$ in order to use (3.16). In view of the upper bound (3.19), this leads to

$$\begin{aligned}
\dot{\mathscr{W}}_\varepsilon(t) &\leq -\left(\gamma - \varepsilon\left[1 + \frac{(2+\gamma+R\mathscr{A})^2}{2b}\right]\right)\|p(t)\|^2 - \alpha\mu\xi(t)^2 - \frac{b\varepsilon}{2}\|x(t)-x^*\|^2 \\
&\quad - a\varepsilon(f(x(t)) - f(x^*)) \\
&\leq -\min\left\{2\alpha, a\varepsilon, \frac{b}{3}, \frac{4}{3}\left(\gamma - \varepsilon\left[1 + \frac{(2+\gamma+R\mathscr{A})^2}{2b}\right]\right)\right\}\mathscr{W}_\varepsilon(t),
\end{aligned}$$

from which one can conclude exponential convergence of $\mathscr{W}_\varepsilon(t)$ to 0 by a Grönwall inequality, upon choosing $\varepsilon > 0$ small enough (which can be optimised in order to get an explicit lower bound on the convergence rate). This implies the desired inequality. □

Note that the convergence rate a priori depends on the chosen initial condition. This is due to a technical limitation in the way we prove exponential convergence, since the dynamics is compared to a family of dissipated Hamiltonian dynamics, of which we consider the worse case scenario. More importantly, our proof currently only holds for $\gamma > 0$. We believe that the convergence result can be improved, and extended to more general situations. This is however left for future work.

3.4 Numerical methods

The dynamical system (3.9)-(3.11) may be unfamiliar unless one has prior experience of Nosé dynamics. Since a reliable numerical method is essential in situations where many thousands of steps may be taken to determine a minimum, we discuss the integration method in detail here. We first mention that in our experiments, we adopted a simple splitting integrator to propagate linearly dissipated Hamiltonian dynamics. Following the standard method for such systems, we break the system into three parts: (i) $\dot{x} = p$, $\dot{p} = 0$, (ii) $\dot{x} = 0$, $\dot{p} = -\nabla f(x)$, and (iii) $\dot{x} = 0$, $\dot{p} = -\gamma p$. Each of these systems is easily integrated so we compose the flows to obtain a simple (and reliable) numerical scheme. The same general procedure can be used to integrate the dynamical equations of friction-adaptive descent.

3.4.1 Splitting scheme for Linearly Dissipated Hamiltonian Dynamics (LDHD)

We discretize the system of equations (3.1)-(3.2) using the splitting scheme presented in (3.20). Splitting methods are applicable when the dynamics we are trying to solve can be split into a sum of parts each of which is easier to integrate than the original dynamics. For an introduction to splitting methods we refer the reader to [87]. Here, the dynamics is decomposed as

$$\begin{pmatrix} \dot{x} \\ \dot{p} \end{pmatrix} = \underbrace{\begin{pmatrix} p \\ 0 \end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix} 0 \\ -\nabla f(x) \end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix} 0 \\ -\gamma p \end{pmatrix}}_{\text{D}}. \tag{3.20}$$

We use a symmetric integrator which we denote as “BADAB”. For each full step of the numerical integrator, we want to solve each part (A,B,D) of the dynamics over a time step δt. Since the letter “B” appears twice in the integrator, each of the “B” steps is integrated with a stepsize of $\delta t/2$. The same holds for the “A” step, but not for the “D” step, which will be integrated over with a full time step δt. This is the integrator that we use for all numerical experiments involving LDHD. More details on solving A,B,D are given below.

3.4.2 A splitting scheme for friction-adaptive descent

One possible additive decomposition of the FAD dynamics is the following:

$$\begin{pmatrix} \dot{x} \\ \dot{p} \\ \dot{\xi} \end{pmatrix} = \underbrace{\begin{pmatrix} p \\ 0 \\ 0 \end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix} 0 \\ -\nabla f(x) \\ 0 \end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix} 0 \\ -\xi A(x)p \\ \frac{p^T A(x) p}{\mu} - \alpha \xi \end{pmatrix}}_{\text{C}} + \underbrace{\begin{pmatrix} 0 \\ -\gamma p \\ 0 \end{pmatrix}}_{\text{D}}. \tag{3.21}$$

To design an algorithm, we combine the steps defined above to create an integration scheme for our method. We discuss another decomposition in Section 3.4.3 below.

In general ABCD represents an integrator where A is solved for time δt then B for time δt, etc. Since the steps do not commute, we get many different numerical methods on the alphabet A,B,C,D. Guided by past experience we favor schemes which are symmetric in form (palindromic letter sequences).

There is an important efficiency constraint on the arrangement of steps. Since both B and C involve the gradient of f, if we separate these by any step that changes x we will need to perform a second gradient calculation. Since the gradient is very likely the most significant cost in the integrator, it is important to avoid this extra calculation. This suggests to favor schemes that keep B and C together. It makes sense to combine them at the middle of the step, particularly where gradients are likely to be based on subsampling, thus we suggest to use the integrator DABCBAD, where each evaluation of A,B, and D would be performed for a half timestep. With the exception of part C, each of the terms can be integrated exactly. The part C itself can be solved by a further round of splitting.

Let us first consider the A, B and D parts. The equations of the A part are

$$\begin{aligned} \dot{x} &= p, \\ \dot{p} &= 0, \\ \dot{\xi} &= 0. \end{aligned}$$

The variables p and ξ are constant during this step and the solution at time t, given initial conditions (x_0, p_0, ξ_0), is just $x(t) = x_0 + t p_0$. In a similar way the positions and the auxiliary variable are constant during the B step, so the force F is constant, and the update of momenta is therefore $p(t) = p_0 + tF(x_0)$. Because of the simple choice of linear dissipation, the D update boils down to $p(t) = \exp(-t\gamma)p_0$. This leaves just the C part of the splitting to discuss.

The equations for C are

$$\dot{p} = -\xi A(x) p, \tag{3.22}$$

$$\dot{\xi} = \frac{p^T A(x) p}{\mu} - \alpha\xi. \tag{3.23}$$

During this step $\dot{x} = 0$ so x is constant and we can write $A(x) \equiv A$ (constant). We propose to solve this system by one of the following techniques, depending on the method choice.

The "C" step for KFAD/FFAD/mixture coupling

The solution is greatly simplified in certain cases, e.g. $A = I$ (KFAD) or $A = FF^T$ (FFAD). More generally, we can solve it readily for any mixture coupling of the form

$$A = \lambda_1 I + \lambda_2 FF^T,$$

which addresses both of the two special cases as well as more general couplings.

We rely on the following leapfrog scheme:

$$p_{n+1/2} = \exp(-(\delta t/2)\xi_n A) p_n, \tag{3.24}$$

$$\xi_{n+1} = \exp(-\delta t\alpha)\xi_n + \frac{1}{\alpha\mu}(1 - \exp(-\delta t\alpha)) p_{n+1/2}^T A p_{n+1/2}, \tag{3.25}$$

$$p_{n+1} = \exp(-(\delta t/2)\xi_{n+1} A) p_{n+1/2}, \tag{3.26}$$

To calculate the matrix exponentials we use the fact that A can be written as

$$A = \lambda_1 I + \lambda_2 \|F\|^2 \Pi,$$

where

$$\Pi = \|F\|^{-2} FF^T$$

is an orthogonal projector. Rodrigues' formula [72] gives, for $\tau \in \mathbb{R}$,

$$\exp(\tau\Pi) = I + (\mathrm{e}^\tau - 1)\Pi.$$

Thus, for any $p \in \mathbb{R}^d$,

$$\exp(\tau\Pi) p = p + (\mathrm{e}^\tau - 1)\Pi p = p + (\mathrm{e}^\tau - 1)\|F\|^{-2}(p \cdot F)F.$$

This gives, for $\tau_n = -\delta t \lambda_2 \|F\|^2 \xi_n / 2$,

$$p_{n+1/2} = \exp(-(\delta t/2)\xi_n \lambda_1) \left(p_n + (\mathrm{e}^{\tau_n} - 1)\|F\|^{-2}(p_n \cdot F)F \right).$$

After computing ξ_{n+1} using (3.25), we then follow by computing $\tau_{n+1} = -\delta t \lambda_2 \|F\|^2 \xi_{n+1}/2$ and setting

$$p_{n+1} = \exp(-(\delta t/2)\xi_{n+1}\lambda_1)\left(p_{n+1/2} + (\mathrm{e}^{\tau_{n+1}} - 1)\|F\|^{-2}(p_{n+1/2}\cdot F)F\right).$$

The "C" step for general matrices $A(x)$

In the general case we cannot simplify the matrix exponentials as above. Instead we suggest to use the following linearly implicit scheme (here $A \equiv A(x)$ is the matrix fixed at the start of the C step):

$$\begin{aligned} p_{n+1/2} &= p_n - (\delta t/2)\xi_n A p_{n+1/2}, \\ \xi_{n+1} &= \mathrm{e}^{-\alpha\delta t/2}\xi_n + (\mu\alpha)^{-1}(1-\exp(-\alpha\delta t))p_{n+1/2}^T A p_{n+1/2}, \\ p_{n+1} &= p_{n+1/2} - (\delta t/2)\xi_{n+1} A p_{n+1/2}. \end{aligned}$$

3.4.3 Alternative solution for higher accuracy

Another possible splitting scheme for the FAD dynamics is based on the following decomposition of the dynamics, which differs from the one considered in Section 3.4.2 in the way the C,D parts are decomposed:

$$\begin{pmatrix} \dot{x} \\ \dot{p} \\ \dot{\xi} \end{pmatrix} = \underbrace{\begin{pmatrix} p \\ 0 \\ 0 \end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix} 0 \\ -\nabla f(x) \\ 0 \end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix} 0 \\ -\xi A(x)p \\ \frac{p^T A(x) p}{\mu} \end{pmatrix}}_{\text{C'}} + \underbrace{\begin{pmatrix} 0 \\ -\gamma p \\ -\alpha\xi \end{pmatrix}}_{\text{D'}}. \tag{3.27}$$

We derive the integrator for C' here as the other parts of the splitting scheme can be integrated analytically, as discussed in Section 3.4.2. As for the previous splitting, during the evolution of the C' system, the matrix $A(x) \equiv A$ (i.e., is fixed). There are several cases to consider.

Case I. If A is a projector, $A^2 = A$, we proceed as follows. Taking the dot product of (3.22) by p, we find equations for ξ and $\omega = p^T A p$ of the form

$$\dot{\omega} = -2\omega\xi, \tag{3.28}$$

$$\dot{\xi} = \omega/\mu. \tag{3.29}$$

Once ξ has been found, the solution of (3.22) requires solving

$$\dot{p}(t) = -\xi(t) A p.$$

In the case that A is the identity (KFAD) the calculation is trivial. It also easily solved in other settings since the time-dependency in the right hand side is scalar, in other words $\xi(t)A$ commutes with $\xi(t')A$ for any $t,t' \geq 0$. In this case the solution is just

$$p(t) = \exp(\eta(t)A)\, p(0), \tag{3.30}$$

where $\eta(t) = -\int_0^t \xi(s)\mathrm{d}s$. We can compute the exponential in (3.30) using Rodrigues' formula.

The equations (3.28)-(3.29) are completely integrable, but their exact solution requires the integration of transcendental functions and their subsequent inversion. Instead, we solved the two-dimensional system by further splitting and integration with smaller timesteps, in a multiple time-stepping approach [136].

This scheme can be implemented with any desired accuracy. We used this integrator with good results in our tests of atomic clusters in Section 6.

Case II. We next consider $A = \lambda_1 I + \lambda_2 \Pi$, where Π is a projector. Then we can define $\theta = \|p\|^2$ and $\omega = p^T \Pi p$. After some calculation, this results in a closed system of three equations in the three variables θ, ω, and ξ. These can be solved to high accuracy using splitting.

Another special case arises if A is either diagonal or easily diagonalized by some orthogonal matrix, that is $A = XDX^T$ where D is diagonal. Then splitting and multiple timestepping gives a high accuracy solution for little additional work.

3.5 Convergence analysis

To discuss convergence to the minimizer in the case of the discretization scheme, we first observe that FAD methods can be viewed as dissipated versions of Hamiltonian dynamics. In the simplest case we consider the system as being evolved under a Verlet discretization in combination with steps that introduce linear damping and nonlinear controlled damping (through auxiliary equations). The analysis we provide in this section proceeds in two steps: we first check whether stationary configurations of discretized dynamics indeed correspond to critical points of the continuous dynamics; and then proceed to prove longtime convergence of the discrete dynamics to the equilibria specified in Proposition 3.3.2.

The analysis is conducted for a simple first order splitting scheme. A similar analysis can be conducted for other schemes, such as the ones derived in Section 3.4. We however consider here yet another scheme, simpler in its final form than the schemes considered in Section 3.4, to simplify the presentation.

More precisely, we consider the C'D'BA splitting, where the C' part is integrated by analytically updating the p variable with $\xi A(x)$ fixed, and then adjusting the ξ variable to preserve the quadratic invariant $\|p\|^2 + \mu\xi^2$ (associated to the subdynamics C'). This scheme is slightly simpler to analyze than the schemes of the previous section. It corresponds to the following sequence of updates:

$$\widetilde{p}_{n+1/2} = \mathrm{e}^{-A(x_n)\xi_n\delta t} p_n, \tag{3.31}$$

$$\widetilde{\xi}_{n+1} = \xi_n \sqrt{1 + p_n^T \frac{I - \mathrm{e}^{-2A(x_n)\xi_n\delta t}}{\mu\xi_n^2} p_n}, \tag{3.32}$$

$$\widetilde{p}_{n+1} = \mathrm{e}^{-\gamma\delta t}\widetilde{p}_{n+1/2}, \tag{3.33}$$

$$\xi_{n+1} = \mathrm{e}^{-\alpha\delta t}\widetilde{\xi}_{n+1}, \tag{3.34}$$

$$p_{n+1} = \widetilde{p}_{n+1} - \delta t \nabla f(x_n), \tag{3.35}$$

$$x_{n+1} = x_n + \delta t p_{n+1}. \tag{3.36}$$

Upon introducing the matrix $\beta_{n,\delta t} = \mathrm{e}^{-(\gamma I + \xi_n A(x_n))\delta t} \in \mathbb{R}^{d\times d}$, this scheme can be rewritten as

$$p_{n+1} = \beta_{n,\delta t} p_n - \delta t \nabla f(x_n), \tag{3.37}$$

$$x_{n+1} = x_n + \delta t \beta_{n,\delta t} p_n - \delta t^2 \nabla f(x_n), \tag{3.38}$$

$$\xi_{n+1} = \mathrm{e}^{-\alpha\delta t}\xi_n \sqrt{1 + p_n^T \frac{I - \mathrm{e}^{-2A(x_n)\xi_n\delta t}}{\mu\xi_n^2} p_n}. \tag{3.39}$$

Before proceeding to a convergence analysis of the discrete dynamics, we examine the *regularity* of the numerical method, that is, whether the equilibria of the discretized dynamics coincide with those of the continuous dynamics. We wish to show that the equilibrium for the continuous dynamics (x^*, p^*, ξ^*) derived in Proposition 3.3.2 is also an equilibrium for the discretised system, and further to ensure that the proposed discrete dynamics does not introduce "artificial" stationary points which do not correspond to equilibria of the continuous system.

Proposition 3.5.1. *A state $z = (x, p, \xi)$ is an equilibrium point of the discretized dynamics C'D'BA as presented in equations (3.31)-(3.36) if and only if z is an equilibrium of the continuous dynamics given by equations (3.9)-(3.11).*

Proof. Let $z^* = (x^*, p^*, \xi^*)$ be an equilibrium point of the continuous dynamics (3.9)-(3.11). We have shown in Proposition 3.3.2 that such an equilibrium is of the form $z^* = (x^*, 0, 0)$, where $\nabla f(x^*) = 0$.

It is trivial to see that if we start with $z_n = (x^*, 0, 0)$ and take one step of the discretized dynamics, i.e. apply the scheme (3.31)-(3.36) , we obtain $z_{n+1} = (x^*, 0, 0)$.

We next show that any equilibrium point of the discrete dynamics is an equilibrium of the continuous dynamics. Consider an equilibrium point z_n of the system of equations (3.31)-(3.36). Then, starting from z_n and taking a step of the discretised dynamics, one remains at the equilibrium, i.e. $z_{n+1} = z_n$. Setting $x_{n+1} = x_n$ in (3.36) gives us $p_{n+1} = 0$. This implies $p_n = p_{n+1} = 0$ as z_n is an equilibrium point. Using $p_n = 0$, (3.37) yields $\nabla f(x_n) = 0$. Finally, we reformulate (3.39) as

$$\xi_{n+1} = \mathrm{e}^{-\alpha\delta t}\sqrt{\xi_n^2 + p_n^T \frac{I - \mathrm{e}^{-2A(x_n)\xi_n\delta t}}{\mu} p_n}.$$

Using that $\xi_{n+1} = \xi_n$ and $p_n = 0$, we obtain $\xi_n^2 = e^{-2a\delta t}\xi_n^2 \Rightarrow \xi_n = 0$ since $\alpha, \delta t \neq 0$. Therefore, we have that $z_n = z^*$. □

Proposition 3.5.2. *Assume that $f \in C^2$ with $0 < m \leq \nabla^2 f(x) \leq M < +\infty$, that A is a function with values in the space of symmetric positive semidefinite $d \times d$ matrices, bounded on compact sets, and that $\gamma > 0$. Fix $L \in \mathbb{R}_+$. Then, for any initial condition $(x_0, p_0, \xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}_+$ such that*

$$\|x_0\| + \|p_0\| + \xi_0 \leq L,$$

there exist $\delta t^\star > 0$, $\kappa > 0$ and $C \in \mathbb{R}_+$ (depending on L) such that

$$\forall \delta t \in (0, \delta t^\star), \quad \forall n \geq 0, \qquad f(x_n) - f(x^*) + \|p_n\|^2 + \xi_n^2 \leq C\mathrm{e}^{-\kappa n \delta t},$$

where (x_n, p_n, ξ_n) are the iterates of the discretized dynamics C'D'BA as presented in equations (3.31)-(3.36).

The upper bound on the Hessian could be replaced more generally by M-smoothness. The lower bound is equivalent to m-strong convexity.

Proof. Without loss of generality, and in order to lighten the notation, we assume that $x^* = 0$ and $f(x^*) = 0$. We compute the variations of the various terms appearing in the Lyapunov function (3.17), singling out terms of order δt and gathering terms of order δt^2 and higher. The constant C which allows us to bound the remainder term may change from line to line, but is independent of δt and ε. First, using the upper bound on the Hessian,

$$\begin{aligned} f(x_{n+1}) &\leq f(x_n) + \delta t \nabla f(x_n) \cdot p_{n+1} + \frac{\delta t^2 M}{2}\|p_{n+1}\|^2 \\ &= f(x_n) + \delta t \beta_{n,\delta t} p_n \cdot \nabla f(x_n) + \delta t^2 \left[\frac{M}{2}\|\beta_{n,\delta t} p_n - \delta t \nabla f(x_n)\|^2 - \|\nabla f(x_n)\|^2\right] \\ &\leq f(x_n) + \delta t \beta_{n,\delta t} p_n \cdot \nabla f(x_n) + C\delta t^2 \left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right). \end{aligned}$$

Moreover,

$$\begin{aligned}\|x_{n+1}\|^2 &= \|x_n\|^2 + 2\delta t\beta_{n,\delta t}p_n\cdot x_n + \delta t^2\left[-2x_n\cdot\nabla f(x_n) + p_n^T\beta_{n,2\delta t}p_n\right]\\ &\quad - 2\delta t^3\beta_{n,\delta t}p_n\cdot\nabla f(x_n) + \delta t^4\|\nabla f(x_n)\|^2\\ &\leq \|x_n\|^2 + 2\delta t\beta_{n,\delta t}p_n\cdot x_n - 2\delta t^2 x_n\cdot\nabla f(x_n) + C\delta t^2\left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right).\end{aligned}$$

Next,

$$\|p_{n+1}\|^2 = p_n^T\beta_{n,2\delta t}p_n - 2\delta t\beta_{n,\delta t}p_n\cdot\nabla f(x_n) + \delta t^2\|\nabla f(x_n)\|^2.$$

Additionally,

$$\begin{aligned}x_{n+1}\cdot p_{n+1} &= p_n\cdot x_n - (I-\beta_{n,\delta t})p_n\cdot x_n - \delta t x_n\cdot\nabla f(x_n) + \delta t p_n^T\beta_{n,2\delta t}p_n\\ &\quad - 2\delta t^2\beta_{n,\delta t}p_n\cdot\nabla f(x_n) + \delta t^3\|\nabla f(x_n)\|^2\\ &\leq p_n\cdot x_n - \delta t x_n\cdot\nabla f(x_n) - (I-\beta_{n,\delta t})p_n\cdot x_n + \delta t p_n^T\beta_{n,2\delta t}p_n\\ &\quad + C\delta t^2\left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right)\\ &\leq p_n\cdot x_n - \delta t x_n\cdot\nabla f(x_n) - (I-\beta_{n,\delta t})p_n\cdot x_n + \delta t\mathrm{e}^{-2\gamma\delta t}\|p_n\|^2\\ &\quad + C\delta t^2\left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right).\end{aligned}$$

Finally,

$$\mu\xi_{n+1}^2 = \mathrm{e}^{-2\alpha\delta t}\left[\mu\xi_n^2 + p_n^T\left(I - \mathrm{e}^{-2\xi_n A(x_n)\delta t}\right)p_n\right].$$

We next consider the discrete Lyapunov function

$$\mathscr{W}(x,p,\xi) = f(x) + \frac{\|p\|^2}{2} + a\mu\xi^2 + b\|x\|^2 + cx\cdot p, \tag{3.40}$$

with parameters a, b, c to be determined. We choose the factor $1/2$ for $\|p\|^2$ so that the terms $\beta_{n,\delta t} p_n \cdot \nabla f(x_n)$ coming from the bounds on $f(x_{n+1})$ and $\|p_{n+1}\|^2$ cancel. Then, for c small enough,

$$
\begin{aligned}
&\mathscr{W}(x_{n+1}, p_{n+1}, \xi_{n+1}) \\
&\quad \leq \mathscr{W}(x_n, p_n, \xi_n) + 2b\delta t \beta_{n,\delta t} p_n \cdot x_n - \frac{1}{2} p_n^T (I - \beta_{n,2\delta t}) p_n \\
&\qquad + c\left[-\delta t x_n \cdot \nabla f(x_n) - (I - \beta_{n,\delta t}) p_n \cdot x_n + \delta t \mathrm{e}^{-2\gamma\delta t} \|p_n\|^2\right] \\
&\qquad - a\left(1 - \mathrm{e}^{-2\alpha\delta t}\right) \mu \xi_n^2 + a\mathrm{e}^{-2\alpha\delta t} p_n^T \left(I - \mathrm{e}^{-2\xi_n A(x_n)\delta t}\right) p_n \\
&\qquad - 2b\delta t^2 x_n \cdot \nabla f(x_n) + C\delta t^2 \left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right) \\
&\quad = \mathscr{W}(x_n, p_n, \xi_n) - a\left(1 - \mathrm{e}^{-2\alpha\delta t}\right) \mu \xi_n^2 - (c + 2b\delta t)\delta t\, x_n \cdot \nabla f(x_n) \\
&\qquad - \left[\frac{1}{2} - a\mathrm{e}^{-2\alpha\delta t} - c\delta t \mathrm{e}^{-2\gamma\delta t}\right] \|p_n\|^2 + \left[\frac{\mathrm{e}^{-2\gamma\delta t}}{2} - a\mathrm{e}^{-2\alpha\delta t}\right] p_n^T \mathrm{e}^{-2\xi_n A(x_n)\delta t} p_n \\
&\qquad + \left(\left[2b\delta t \beta_{n,\delta t} - c(I - \beta_{n,\delta t})\right] p_n\right) \cdot x_n + C\delta t^2 \left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right).
\end{aligned}
$$

The last equality suggests to choose $a = \mathrm{e}^{2(\alpha-\gamma)\delta t}/2$. This allows us to remove the factor $p_n^T \mathrm{e}^{-2\xi_n A(x_n)\delta t} p_n$. The component proportional to c in the cross term $x_n \cdot p_n$ is however not so easy to treat since the crude bound $\|I - \beta_{n,\delta t}\| \|x_n\| \|p_n\|$ would potentially lead to a factor $\|x_n\| \|p_n\|$ if $\beta_{n,\delta t}$ is very small, i.e. if $\xi_n A(x_n)$ is large. We therefore need bounds on x_n, ξ_n over the iterations in order to make sure that $\xi_n A(x_n)$ remains uniformly bounded. Such bounds are, in turn, obtained from the estimates on the Lyapunov function.

More precisely, we set $a = \mathrm{e}^{2(\alpha-\gamma)\delta t}/2$ and $c = b = \varepsilon > 0$, so that, using $x \cdot \nabla f(x) \geq 0$, and upon choosing δt and ε small enough,

$$
\begin{aligned}
\mathscr{W}(x_{n+1}, p_{n+1}, \xi_{n+1}) \leq \mathscr{W}(x_n, p_n, \xi_n) &- \varepsilon \delta t\, x_n \cdot \nabla f(x_n) - \frac{\alpha\mu\delta t}{2} \xi_n^2 - \frac{\gamma\delta t}{2} \|p_n\|^2 \\
&+ \varepsilon \left[2\delta t \beta_{n,\delta t} - (I - \beta_{n,\delta t})\right] p_n \cdot x_n + C\delta t^2 \left(\|p_n\|^2 + \|\nabla f(x_n)\|^2\right). \qquad (3.41)
\end{aligned}
$$

The assumption $0 < m < \nabla^2 f(x) \leq M$ together with the equalities $f(0) = 0$ and $\nabla f(0) = 0$ implies through Taylor expansions with integral remainder that $f(x) \geq m\|x\|^2/2$ and $\|\nabla f(x)\| \leq M\|x\|$. In view of these two inequalities, there exists $K \in \mathbb{R}_+$ such that, uniformly in $\varepsilon \in (0, 1/2]$,

$$
\|p\|^2 + \|\nabla f(x)\|^2 \leq K \mathscr{W}(x, p, \xi).
$$

Let us fix $R > 0$. Since A is bounded on compact sets, there exists $\mathscr{W}_R \in \mathbb{R}_+$ such that, for any (x, ξ) with $f(x) + \mu\xi^2 \leq \mathscr{W}_R$,

$$
\xi A(x) \leq R,
$$

so that

$$\mathrm{e}^{-\xi A(x)\delta t} \geq \mathrm{e}^{-R\delta t}. \tag{3.42}$$

Therefore, if x and ξ are bounded through the inequality $f(x)+\mu\xi^2 \leq \mathscr{W}_R$, then $\beta_{n,\delta t}$ does not become too small. Indeed, whatever the choice of $\varepsilon \in (0,1/2]$, and upon choosing δt small enough so that $a \leq 1$, one obtains from (3.40) that $f(x)+\mu\xi^2 \leq \mathscr{W}(x,p,\xi)$. The bound $f(x)+\mu\xi^2 \leq \mathscr{W}_R$ therefore holds as long as $\mathscr{W}(x,p,\xi) \leq \mathscr{W}_R$, irrespectively of the value of $\varepsilon \in (0,1/2]$. In particular, any $(x,p,\xi) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}_+$ with $\mathscr{W}(x,p,\xi) \leq \mathscr{W}_R$ satisfies

$$0 \leq I - \mathrm{e}^{-(\gamma+\xi A(x))\delta t} \leq 1 - \mathrm{e}^{-(R+\gamma)\delta t}. \tag{3.43}$$

We now conclude the proof by induction. We assume that $\mathscr{W}(x_n,p_n,\xi_n) \leq \mathscr{W}_R$, with $R>0$ chosen such that the inequality holds for $n=0$ as well. We want to prove that a similar inequality holds for $n+1$. Note that the induction hypothesis implies that the condition (3.43) holds. Using (3.41) and applying (3.43), we have

$$\begin{aligned}
\mathscr{W}(x_{n+1},p_{n+1},\xi_{n+1}) &\leq (1+CK\delta t^2)\mathscr{W}(x_n,p_n,\xi_n) \\
&\quad - \varepsilon\delta t\, x_n\cdot\nabla f(x_n) - \frac{\alpha\mu\delta t}{2}\xi_n^2 - \frac{\gamma\delta t}{2}\|p_n\|^2 \\
&\quad + \varepsilon\left(2\delta t + \left\|I-\beta_{n,\delta t}\right\|\right)\|p_n\|\|x_n\| \\
&\leq (1+CK\delta t^2)\mathscr{W}(x_n,p_n,\xi_n) + \varepsilon\left(2\delta t + 1 - \mathrm{e}^{-(R+\gamma)\delta t}\right)\|p_n\|\|x_n\| \\
&\quad - \varepsilon\delta t\, x_n\cdot\nabla f(x_n) - \frac{\alpha\mu\delta t}{2}\xi_n^2 - \frac{\gamma\delta t}{2}\|p_n\|^2 \\
&\leq (1+CK\delta t^2)\mathscr{W}(x_n,p_n,\xi_n) \\
&\quad - \varepsilon\delta t\left(x_n\cdot\nabla f(x_n) - \eta\frac{\left|2\delta t+1-\mathrm{e}^{-(R+\gamma)\delta t}\right|}{\delta t}\|x_n\|^2\right) \\
&\quad - \frac{\alpha\mu\delta t}{2}\xi_n^2 - \frac{\delta t}{2}\left[\gamma - \frac{\varepsilon}{2\eta}\frac{\left|2\delta t-1+\mathrm{e}^{-(R+\gamma)\delta t}\right|}{\delta t}\right]\|p_n\|^2 \\
&\leq (1-\rho\delta t+CK\delta t^2)\mathscr{W}(x_n,p_n,\xi_n),
\end{aligned}$$

for some $\rho>0$, upon choosing $\eta,\varepsilon>0$ small enough (which is possible in view of the discussion following (3.42)). The latter right hand side remains smaller than $\mathscr{W}_R$ for δt small enough. This shows that the induction hypothesis holds for $n+1$ as well, and allows us to conclude that it holds for all $n \geq 0$.

In fact, the very same calculations also give the claimed exponential convergence. □

As mentioned, a similar analysis can be conducted for the schemes derived in Section 3.4, but we chose to perform convergence analysis for the discretisation scheme (3.37)-(3.39) for the sake of simplicity of presentation. Note that this analysis can potentially be improved, as we were aiming to show convergence but did not aim to optimise the obtained convergence rate. Therefore, there is no direct correspondence between the results of the convergence analysis in this section and the numerical results reported in Section 3.6.

3.6 Numerical experiments

We concentrate in Section 3.6.1 on exploring the roles of the parameters and their ranges in the case of the Rosenbrock problem, and on comparing the performance of FAD methods to linearly dissipated Hamiltonian descent. After this we turn our attention in Section 3.6.2 to the application of FAD methods to the minimization of atomic clusters.

3.6.1 Parameter selection in the Rosenbrock problem

In this section, we examine how different combinations of γ and δt affect convergence of LDHD as given by (3.1)-(3.2) and mixture couplings according to the dynamics (3.9)-(3.10)-(3.11) and also examine how the parameters μ and α affect convergence. All comparisons in this first subsection are performed on the Rosenbrock function.

When comparing LDHD to FAD schemes, we used, as mentioned, the symmetric integrator for LDHD denoted as BADAB presented in Section 3.4.1, where D refers to the linear damping. Contraction rates for Langevin dynamics under synchronous coupling [74] also establish exponential convergence for BADAB (as many other discretizations) of LDHD.

We start by examining three different specific initializations and comparing the $(\gamma, \delta t)$ plots for LDHD and four variants of mixture coupling, namely the projective mixture-coupling method of Section 3.2.3 with $[\lambda_1, \lambda_2] = [0.1, 0.9], [0.5, 0.5], [0.9, 0.1]$ and $[1, 0]$. The three initializations we examine correspond to $(x_0, y_0) = (1.5, -0.5)$, $(x_0, y_0) = (0, -2)$ and $(x_0, y_0) = (4, 4)$. We recall that the minimum of the two-dimensional Rosenbrock is at $(x^*, y^*) = (1, 1)$. The momenta, as well as the auxiliary variable ξ are initialized to zero. The graphs are colored according to the number of iterations it takes for each of the methods to reach the vicinity of the minimum (within a tolerance of 10^{-4}). Each row of Fig. 3.6 corresponds to a specific initialization and each column to one of the five optimisers (LDHD and the four variants of projective mixture coupling). The maximum number of iterations has been set to 3000. The parameters μ and α were set to $\mu = 1$ and $\alpha = 1$ for the runs in Fig. 3.6.

As we can see in Fig. 3.6, for the three specific initializations we examined, KFAD is much more robust than LDHD in the sense that it allows fast convergence for a much wider range of step sizes and friction parameters. This is especially noticeable in the low friction regime, where LDHD converges rapidly only for small stepsizes. The results also suggest that project-

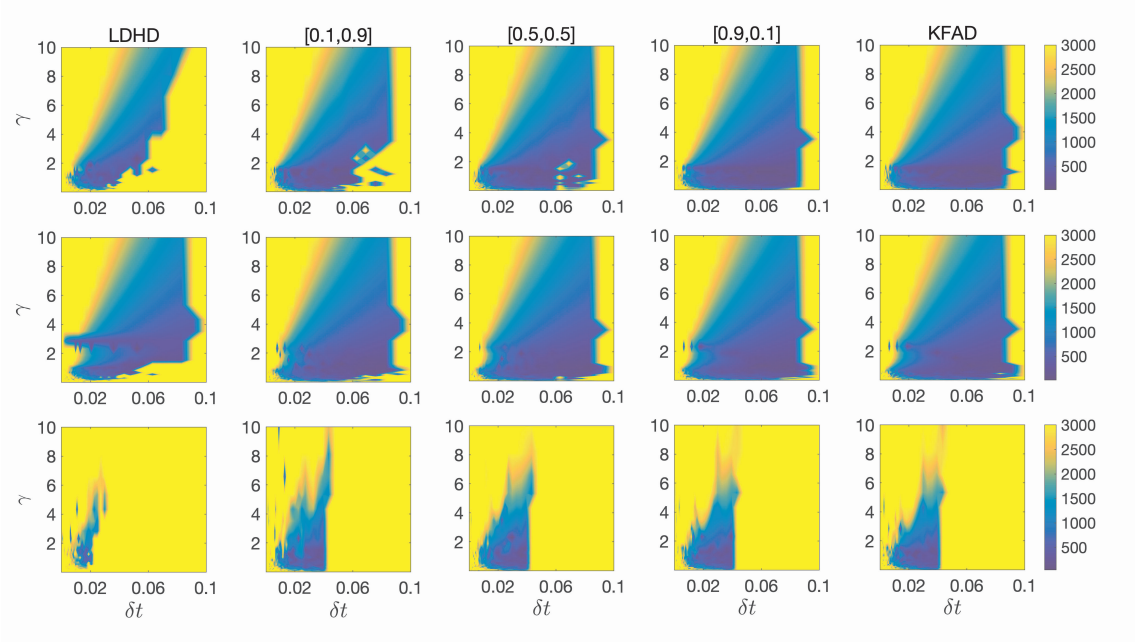


Figure 3.6: Comparison of number of iterations until convergence for LDHD versus projective mixture coupling. We used three different initial conditions: row 1, IC1: $(x_0, y_0) = (1.5, -0.5)$, row 2, IC2: $(0, -2)$, row 3, IC3: $(4, 4)$. The pair at the top of a column corresponds to $[\lambda_1, \lambda_2]$ in projective mixture coupling. Each column corresponds to a different method. In all FAD variants, $\mu = \alpha = 1$. The maximum number of iterations is set to 3000. The same axes and colormap range have been used for all graphs.

ive mixture coupling variants that are close to KFAD are faster and more robust than variants closer to FFAD. It should be noted that performance of all methods for the last initialization $(x_0, y_0) = (4, 4)$ is worse, as that initialization is further away from the minimum and the high gradients and steep potential make convergence more challenging.

Having looked at some specific initializations to get an initial comparison between LDHD and projective mixture coupling variants, we explore $(\gamma, \delta t)$ plots, but this time averaged over the choice of initial point in the (x, y) domain. In this way, we remove the effect that specific initializations may have on efficiency and can see which method performs better on average. We again set $\mu = 1$, $\alpha = 1$ for the runs in Fig. 3.7. We generate a grid of (x, y) pairs, where x ranges between $(-2, 2)$ and y between $(-1, 3)$. We create a (40×40) grid, and therefore average over 1600 different initializations. The plots are again coloured according to the number of iterations it takes the optimiser to reach the vicinity of the minimum. The maximum number of iterations is set to 1500 for better visualization purposes.

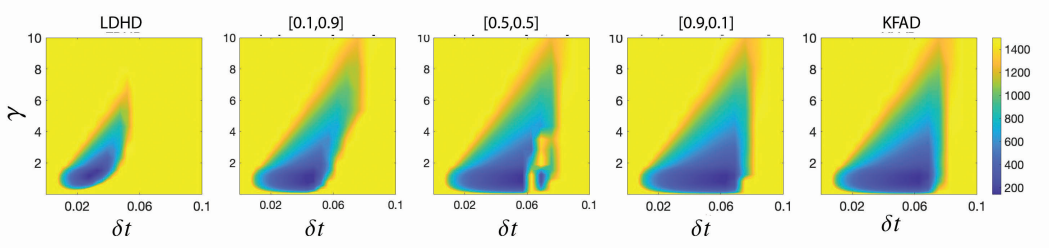


Figure 3.7: Comparison of number of iterations until convergence as a function of γ and δt, for LDHD versus four different variants of projective McFAD. Each of the plots is averaged over a number of different initializations in the (x, y) domain. The values of μ and α are fixed for all FAD plots to $\mu = 1$ and $\alpha = 1$. The maximum number of iterations is set to 1500.

Looking at Fig. 3.7, we see a big difference in the convergence speed of LDHD and the four variants of projective McFAD, the biggest contrast being observed between LDHD and KFAD. We can see that KFAD allows for fast convergence for a much wider gamut of stepsize and friction.

To get a clearer picture of what happens in the low γ regime, we replot the results in Fig. 3.7 using a logarithmic scale in the γ axis. We also allow the simulations to run for longer (5000 iterations as opposed to 1500 before), to make sure we capture convergence speed differences between the methods at low linear friction, which might not be apparent if we stop the runs too soon.

As we can see from the results of Fig. 3.8, KFAD can converge in the very low γ regime. This behavior can be connected to the interpretation of ξ as an “adaptive" friction coefficient, which can compensate for low γ [47, 54, 98]. LDHD has no such mechanism and therefore reduces, effectively, to Hamiltonian dynamics at low friction. Finally, we reproduce the $(\gamma, \delta t)$ graphs of

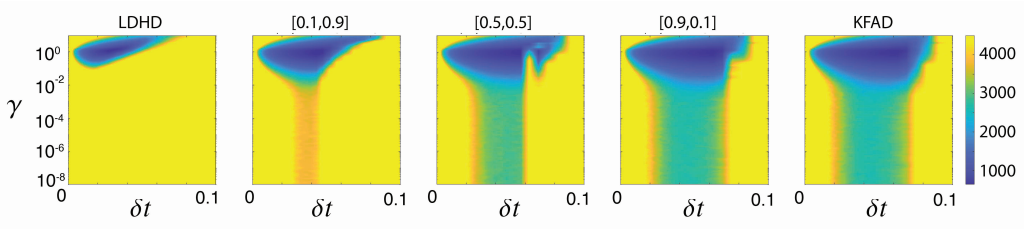


Figure 3.8: Number of iterations for convergence as a function of γ and δt. The pair "$[\lambda_1, \lambda_2]$" indicates the parameters used in mixture coupling. We use a logarithmic scale on the γ axis to focus on the low γ regime. As before, we use $\mu = 1$ and $\alpha = 1$.

Fig. 3.7 in Fig. 3.9 where, instead of coloring according to the number of iterations it takes to reach the vicinity of the minimum, we color according to the ratio of the number of iterations it takes each of the four projective mixture coupling variants to converge over the number of iterations it takes LDHD to converge.

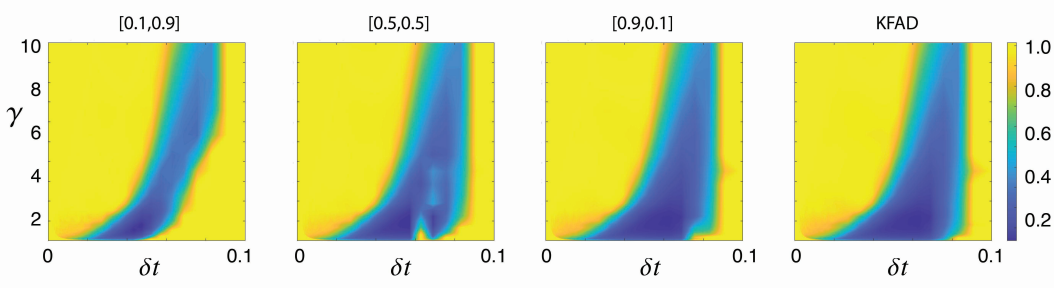


Figure 3.9: Ratio of number of iterations to reach vicinity of the minimum for projective mixture coupling ($\mu = 1, \alpha = 1$) compared to LDHD. The pair "$[\lambda_1, \lambda_2]$" indicates the parameters used in mixture coupling.

Having examined the relationship between γ and δt and how it affects convergence of LDHD and projective mixture coupling, we now proceed to examine the relationship between α and μ.

We choose a set of reasonable values for γ and δt, i.e. $\gamma = 1$, $\delta t = 0.05$ and then plot the number of iterations to reach the vicinity of the minimum as a function of μ and α for projective mixture coupling with various parameters. We again average over 1600 different initializations to eliminate the effect of initialization on convergence speed. The results are reported in Fig. 3.10.

As we can see in Fig. 3.10, there is a front beyond which performance deteriorates, but there is a wide range of (μ, α) pairs on the left of the front that we can choose, resulting in fast convergence. We therefore observe that the algorithm is robust regarding the choice of μ and α. The reason performance deteriorates for high μ is because the limit of friction-adaptive descent as $\mu \to \infty$ is just LDHD. On the other hand, at fixed δt, and for small μ, the tight coupling between kinetic energy and ξ will introduce instability in the numerical method. Thus

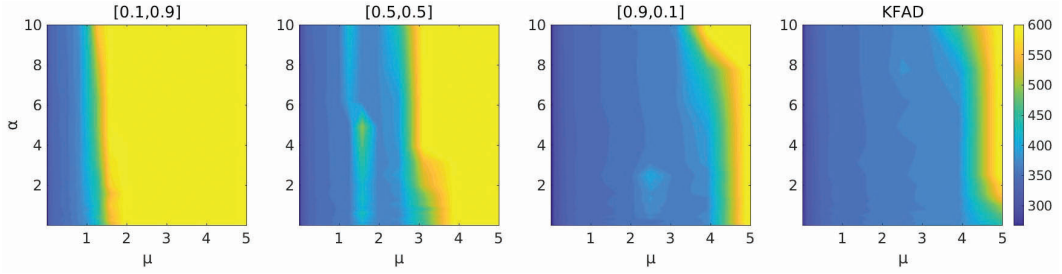


Figure 3.10: Number of iterations until convergence as a function of α and μ for friction $\gamma = 1$ and step size $\delta t = 0.05$. The contour plots are colored according to the number of iterations it takes for the four variants of mixture coupling to reach the vicinity of the minimum. The results are averaged over a range of different initializations in the (x, y) domain.

we recommend values of μ that are bounded in a moderate interval around unity. This is reminiscent of a similar recommendation for Adaptive Langevin dynamics [73]. Our primary objective so far has been to demonstrate that projective McFAD can lead to much faster convergence compared than LDHD. We are not necessarily aiming to show that projective McFAD outperforms other popular optimisation methods. It is however be interesting to see how KFAD compares to Adam [61]. We formulate Adam as a system of ODEs, similarly to [17]. The continuous equations we use for Adam are

$$\dot{x} = \frac{p}{\sqrt{\zeta} + \varepsilon}, \tag{3.44}$$

$$\dot{p} = -\nabla f(x) - \gamma p, \tag{3.45}$$

$$\dot{\zeta} = \nabla f(x)^2 - \alpha\zeta,, \tag{3.46}$$

where $\zeta \in \mathbb{R}^N$ and $\nabla f(x)^2$ denotes element-wise squares. We use the following splitting method to discretize the dynamics (3.44)-(3.46):

$$\begin{pmatrix} \dot{x} \\ \dot{p} \\ \dot{\zeta} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{p}{\sqrt{\zeta}+\varepsilon} \\ 0 \\ 0 \end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix} 0 \\ -\nabla f(x) \\ 0 \end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ \nabla f(x)^2 - \alpha\zeta \end{pmatrix}}_{\text{C}} + \underbrace{\begin{pmatrix} 0 \\ -\gamma p \\ 0 \end{pmatrix}}_{\text{D}}. \tag{3.47}$$

Parts B and D are discretized as explained in section 3.4. Step A can be solved exactly as $x_{n+1} = x_n + \delta t \frac{p_n}{\sqrt{\zeta_n}+\varepsilon}$. For step C we have $\zeta_{n+1} = \mathrm{e}^{-\alpha\delta t}\zeta_n + \frac{F^2}{\alpha}(1 - \mathrm{e}^{-\alpha\delta t})$.

We choose $\alpha = 1$ for Adam and $\alpha = 1$, $\mu = 1$ for KFAD. We set the maximum number of iterations to 2500 and average over 1600 different initializations. The plots in Figs. 3.11 and 3.12 are colored according to the number of iterations it takes for the optimiser to reach the vicinity of the minimum. The horizontal axis corresponds to the step size δt and the vertical

axis to the friction γ. Fig. 3.12 is the same as Fig. 3.11, but with a logarithmic scale on the γ axis. This allows us to see that for the 2D Rosenbrock function, KFAD is more efficient than both LDHD and Adam in the low γ regime. Adam on the other hand demonstrates faster convergence for the high γ/low step size regime.

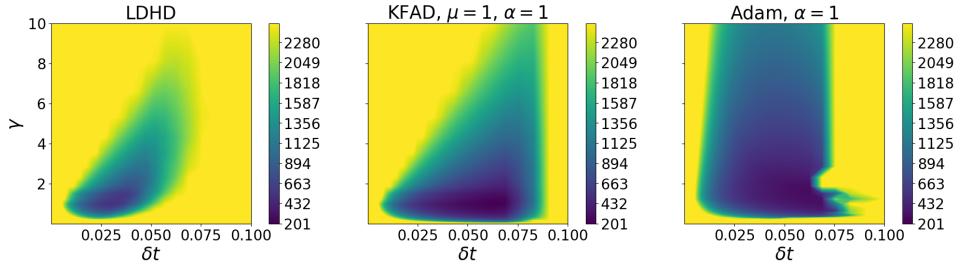


Figure 3.11: Number of iterations for LDHD, KFAD and Adam to reach the vicinity of the minimum of the 2D Rosenbrock function as a function of γ and δt. Results are averaged over 1600 different initializations.

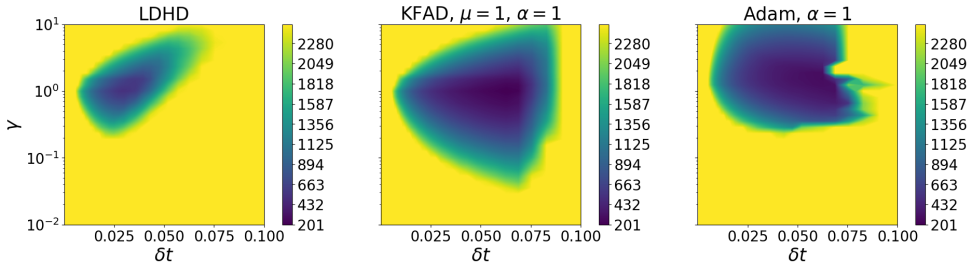


Figure 3.12: Number of iterations for LDHD, KFAD and Adam to reach the vicinity of the minimum function as a function of γ and step size δt. Results are averaged over 1600 different initializations. We use a logarithmic scale on the γ axis for better visualization of the low-γ regime. We can see that KFAD is faster than both LDHD and Adam in the low-γ region.

To make visual comparison of the three methods easier, we show the ratio of iterations it takes for KFAD to converge over those of Adam in Fig. 3.13. We see that there is a green and yellow region, where KFAD is between two to four times slower than Adam. There are also two regions, a horizontal band for low γ's and a vertical band for higher step sizes where KFAD is between three and ten times faster than Adam. We can see that KFAD allows the use of higher step sizes and allows for fast convergence in the high high step size regime for a wide range of γ's.

We would like to stress again that it was not our aim to necessarily compete with Adam, rather we wanted to show that FAD methods are competitive with momentum SGD and to demonstrate how cubic damping can be used as an effective mechanism to suppress oscillations and speed up convergence. In fact, it is possible to blend the ideas underpinning Adam with the ones of FAD in order to hopefully get the best features of both algorithms. We are currently working on this.

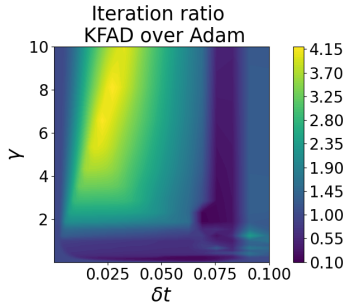


Figure 3.13: Ratio of number of iterations it takes to reach the vicinity of the minimum for KFAD ($\mu = 1, \alpha = 1$) compared to Adam ($\alpha = 1$).

3.6.2 Lennard–Jones and Morse Clusters

As an interesting and nontrivial test problem we have considered the optimisation of atomic clusters consisting of N atoms moving in $\mathbb{R}^3$ and interacting in a pairwise (distance-based) potential. The total potential energy is

$$U(q) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \varphi(\|q_i - q_j\|),$$

where q_i represents the three component vector of position coordinates of the ith atom. This problem has been extensively studied as a test case for global optimisation and many specialized methods have been developed [143]. A significant challenge is the high dimensionality of these problems and the proliferation of local minima, in some cases nearly at the same depth as the global one. As our first test case we used a system of 38 atoms in the Lennard–Jones pair potential defined by

$$\varphi_{\mathrm{LJ}}(r) = 4\left(r^{-12} - r^{-6}\right).$$

We consider here the optimisation for initial conditions taken in the vicinity of the (unique) global minimum which is unique up to rotations and translations.

In this exploration, we attempted again to use FFAD and various mixture coupling methods, but found them, as for the Rosenbrock function, to be of little benefit in terms of efficiency. It should be clarified here that while FFAD was effective in damping oscillations, as was demonstrated for example in Figs. 3.2 and 3.4, the very mechanism that facilitated the damping of oscillatory modes seemed to have an adverse effect on convergence speed, which is what we mean when referring to efficiency (see the discussion after (3.14)). Therefore our focus here is on KFAD and its performance relative to LDHD.

To generate an initial condition for the optimiser, we proceeded as follows. We started at the known (tabulated) minimum energy configuration. We then applied stochastic dynamics (Langevin dynamics) for a short period (400 steps, $\delta t = 0.001$, $\beta^{-1} = 2$ with β proportional to the inverse temperature, similarly to (3.7)) producing a new state with a significantly higher energy. The challenge then for optimisation is to return to the ground state. An example starting structure can be seen in the left panel of Fig. 3.14. It can be compared with the right panel which shows the highly ordered minimum energy cluster.

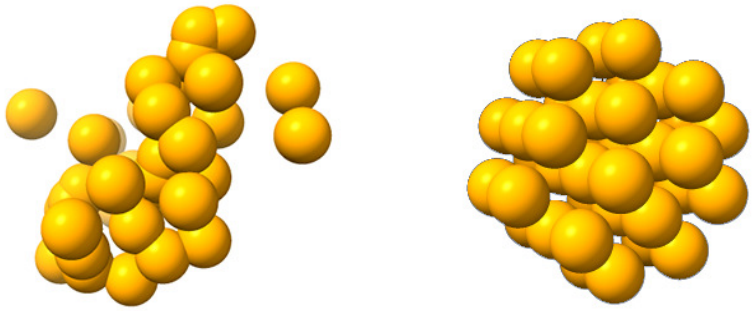


Figure 3.14: Instances of 38 atom clusters for Lennard–Jones interaction potential. Left: starting state for optimisation tests, energy = -141.77. Right: global minimizer, energy = -173.91.

Optimal choice of α and μ

We tuned the artificial parameters α and μ and the friction γ of KFAD by considering various choices of all three. We found that KFAD performed best with small values of γ and with moderate values of μ and α. It was somewhat sensitive to the selection of these two parameters but less so than LDHD was to the choice of the linear damping γ. We show one series in Fig. 3.15 which illustrates the convergence at $\mu = 0.1$ for various values of α.

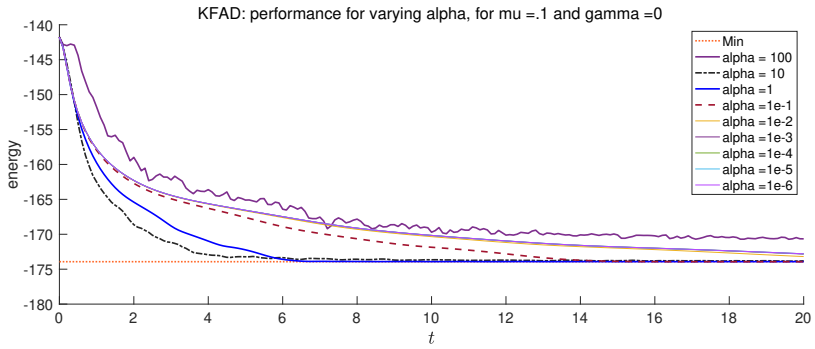


Figure 3.15: Lennard–Jones 38: exploration of the optimal choice of α for $\mu = 0.1$. The objective function is reported as a function of the physical time.

When $\mu = 0.1$, convergence is seen for all values of $\alpha < 100$ and is particularly rapid for $\alpha = 1$ or $\alpha = 10$.

Exploration of LJ75

Having investigated the parameters, we considered direct comparisons between KFAD and LDHD on a specific challenge related to the 75-atom system. The minimum for $N = 75$ is non-icosahedral, with energy -397.492331 (see [25]). We again followed the standard practice of initializing molecular optimisation from Langevin samples. From the minimizer we allowed 400 steps of Langevin dynamics (BAOAB scheme [69], $\beta = 0.5$, stepsize $\delta t = 0.001$ with the friction coefficient in Langevin set to $\gamma = 20$). The result of this process was an initialization at an energy between approximately -260 and -230, significantly higher than the global minimum.

We found that the previous parameter testing in the case of LJ38 was transferable to the larger molecule. We fixed the parameters of KFAD at $\mu = 0.1$, $\alpha = 10$, $\gamma = 10^{-5}$ and considered various choices of friction in LDHD. We found that LDHD required some tuning of the friction and specifically that friction needed to be greater than 1. We allowed 2000 steps for optimisation by each scheme.

In our experiments we observed that the performance of KFAD was relatively independent of the momentum initialization. In our first set of experiments, we initialized both KFAD and LDHD using the thermalized momenta corresponding to $\beta = 0.5$ which were obtained during the initial equilibration phase. This is certainly not an uncommon initialization strategy, but we found that for LDHD the results obtained using these momenta were very poor. In most cases the resulting runs did not produce approximations to the minimum.

We show in Figs. 3.16 and 3.17 a comparison of the energies achieved. KFAD reached the global minimum in more than 83% of runs. For no value of linear damping did LDHD find the minimizer in more than 15% of trials; in most cases the performance was much worse.

These results suggest that KFAD is capable of controlling the descent process even in the setting that it starts with a high kinetic energy, which is perhaps not surprising given that it has an in-built kinetic energy control. We then initialized the system in a different way. After producing a canonical sample using Langevin dynamics as above, we simply set the momenta to 0. In this setting, we rely on properties of the potential surface to guide the state of the system toward the minimum. When started in this way, the results obtained by LDHD were strikingly different; see Fig. 3.18. With friction properly tuned, LDHD found the minimum rapidly (as rapidly as KFAD) and in 93% of cases, whereas KFAD only succeeded in around 77% of runs. This suggests that standard dissipation mechanisms are able to find nearly optimal paths in this example. Increasing the distance to the minimum (by longer equilibration) typically resulted in both methods getting stuck in an elevated energy state.

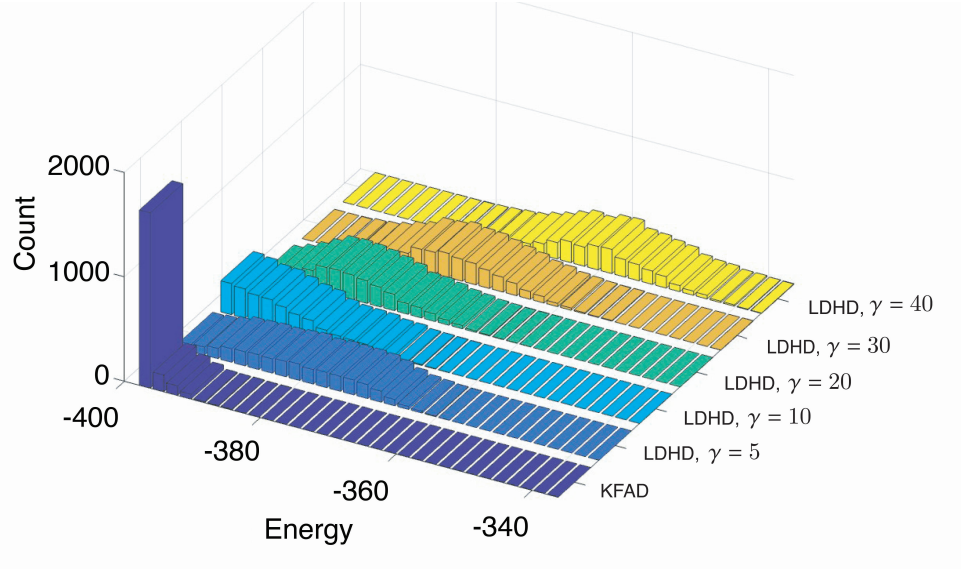


Figure 3.16: Minimizers for runs of the LJ-75 system initiated from the canonical ensemble. The comparison shows the states obtained by LDHD (with various linear damping coefficients) and KFAD, arranged by energy level.

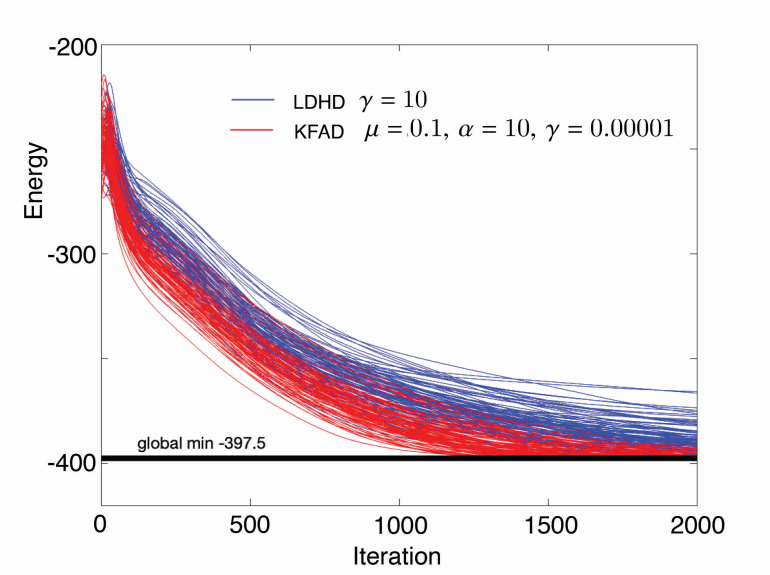


Figure 3.17: Optimisation paths show the decay of energy for LJ-75 in a typical case with momenta initialized from the canonical ensemble. We see that LDHD is typically significantly slower in descent and ends at higher values than in the KFAD runs.

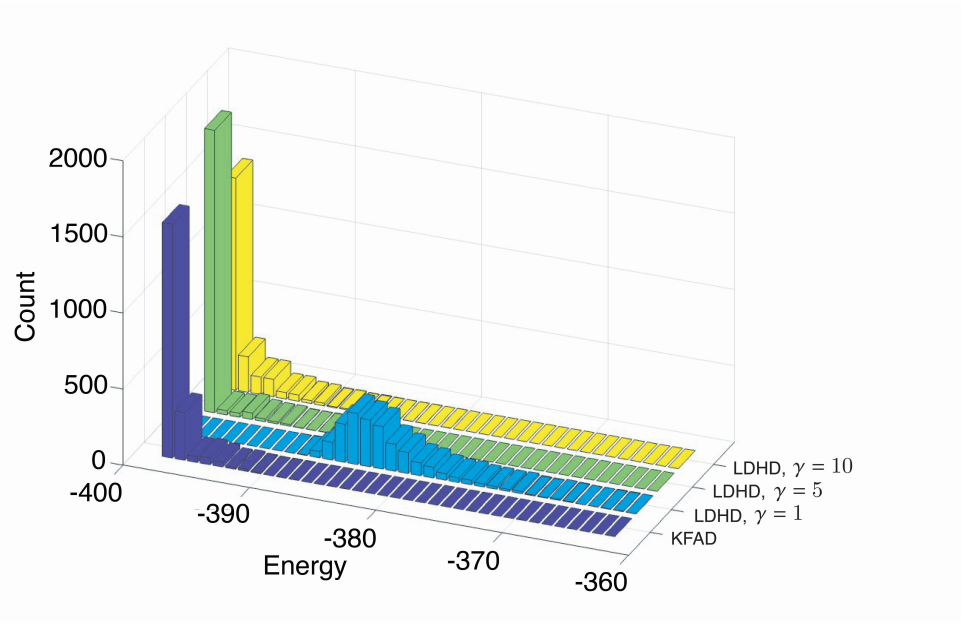


Figure 3.18: Minimizers for runs of the LJ-75 system initiated by zeroing the momenta at the start.

Approximate global minimization of a Morse cluster

Although the results for LJ clusters initialized near the global minimum were inconclusive, we attribute this to the steepness of the Lennard–Jones pair potential and the consequent strong propensity for trapping into a single, nearly-harmonic basin. We next turned our attention to a Morse cluster in which the pair potential is defined by

$$\varphi_{\text{Morse}}(r) = \left(1 - \mathrm{e}^{-a(r-r_0)}\right)^2.$$

Such clusters are discussed in depth in several articles in the chemistry literature [23–25, 88, 144]. Morse potentials were also extensively used in [142] to study why long-range potentials give rise to simpler landscapes. Using the Morse potential in place of the Lennard-Jones potential undoubtedly changes the problem, but the Morse cluster is still regarded as a difficult optimisation problem.

For modest values of a, the propensity for trapping is reduced compared to Lennard–Jones and it is possible to consider the use of FAD methods for (approximate) global optimisation. We used $a = 3$ and $r_0 = 1$ which is one of the cases studied by Doye, Wales and Berry [25]. We initialized a system of 64 atoms by simply placing the atoms on the vertices of a Cartesian lattice (x_i, y_i, z_i) of the form $\{0,1,2,3\} \times \{0,1,2,3\} \times \{0,1,2,3\}$. This leads to a rather large initial energy. We then added an equilibration phase using Langevin dynamics to randomize

the starting point (1000 steps, stepsize $\delta t = 0.001$, $\beta = 0.1$). After this we applied various optimisation methods. The initialization procedure results in a very high starting energy in $[-50, 0]$, compared to the global optimum energy of -512.83. In this setting we found it did not matter how we initialized momenta, so we set the momenta to zero for simplicity.

We explored the choice of stepsize. Choosing the right stepsize is nontrivial. There is a significant range below the stability threshold where the LDHD method results in a poor accuracy, regardless of the friction used. Even though LDHD appeared to be stable (in the sense of bounded energy) for $\delta t < 0.1$ the results were much better with LDHD if $\delta t < 0.05$, whereas KFAD was less sensitive to the choice of δt. This probably is due to the LDHD paths "popping out" of some basins or missing some relevant pathways during minimization. The fact that we can, at least in some cases, use large steps and still obtain accurate minima using KFAD is a favorable feature. In LDHD, the optimal friction was found to lie around $\gamma = 1$; at larger stepsize, higher friction (e.g. $\gamma \approx 2$) gave better results, suggesting that the friction was compensating for numerical error. We found that $\alpha = \mu = 1$ gave the best results in KFAD, although the choice was not critical, and in KFAD, the optimal linear damping was $\gamma = 0$, so all damping comes from the use of the auxiliary term. The use of zero friction in the Morse cluster example stands in stark contrast to our observation that a moderate friction is needed in KFAD when it is applied to the Rosenbrock function.

Unlike in the case of the local minimization of Lennard–Jones we considered earlier, in this optimisation problem we need relatively long trajectories. We initially used 50,000 steps in each run. The results shown in Fig. 3.19 clearly demonstrate that in this case the KFAD method can reach an energy near to that of the global minimum, with high reliability, whereas the LDHD method fails in most cases. We did not pursue the challenge of specifically finding the global minimum, which would require additional machinery, since our interest here was simply the comparison of native LDHD to KFAD.

We next conducted several series of 100 runs with each of three stepsizes and three choices of γ (in LDHD); in this case taking 10^5 steps each time. We have graphed all the obtained minimum energies in each series in Fig. 3.20, along with the mean of all the runs of the series.

In this experiment, we see that KFAD trajectories appear to accumulate in lowest energy basins, whereas LDHD paths appear to get stuck at elevated energies. To test this hypothesis we reran the previous set of experiments using 10^6 steps in every run to see how the figure changed. These results are reported in Fig. 3.21. In this example, it appears that KFAD paths are finding transitions between states that are rarely found by dissipated Hamiltonian descent. At the largest stepsize, most of the minimizers are at (or within 1% of) the global minimum.

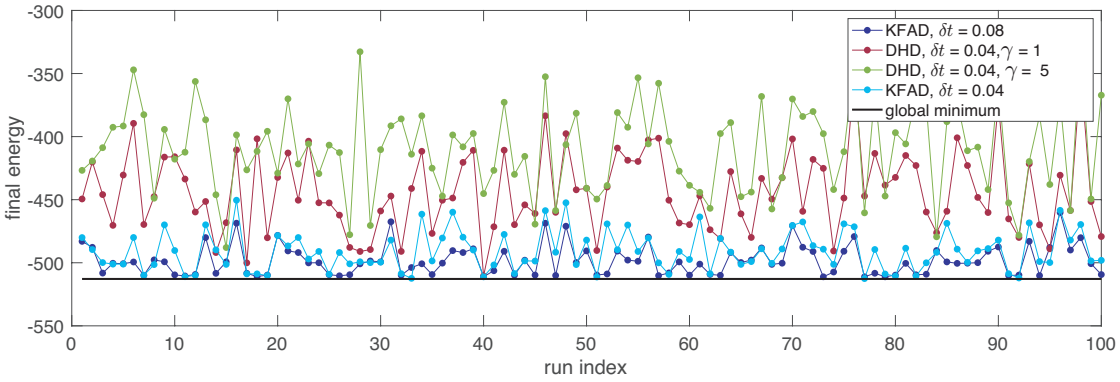


Figure 3.19: Energies obtained using LDHD and KFAD for the 64 atom Morse cluster. The KFAD method approaches the minimum energy in almost all runs, whereas LDHD, for all choices of friction tested, does not come close. Results are shown for KFAD at stepsizes of both $\delta t = 0.04$ and $\delta t = 0.08$, but LDHD only for the smaller stepsize since its performance was much worse at larger stepsize. Moreover, in KFAD, the larger stepsize gave visible improvements over the smaller one, suggesting that exploration was enhanced.

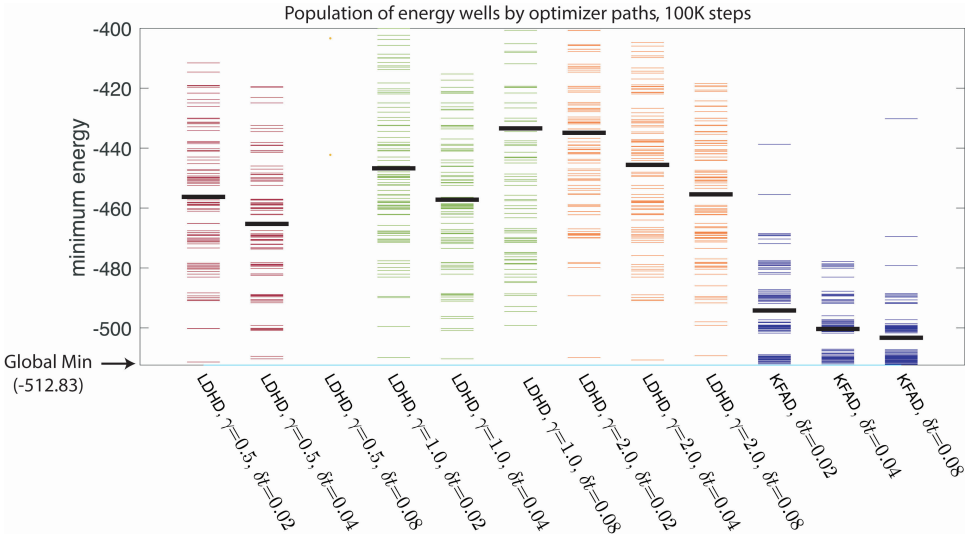


Figure 3.20: A diagram comparing minimum energies obtained for the 64 atom Morse cluster in long runs, for different choices of stepsize and, for LDHD, for different choices of friction. Parameters for KFAD were $\alpha = \mu = 1$ and $\gamma = 0$. The solid black bars show the means of all the minima of the column. The graph shows that the minima achieved by KFAD are overall much lower than those achieved by LDHD for various parameters. We also see that KFAD obtains improved accuracy when using a large stepsize ($\delta t = 0.08$) for which LDHD would be inaccurate.

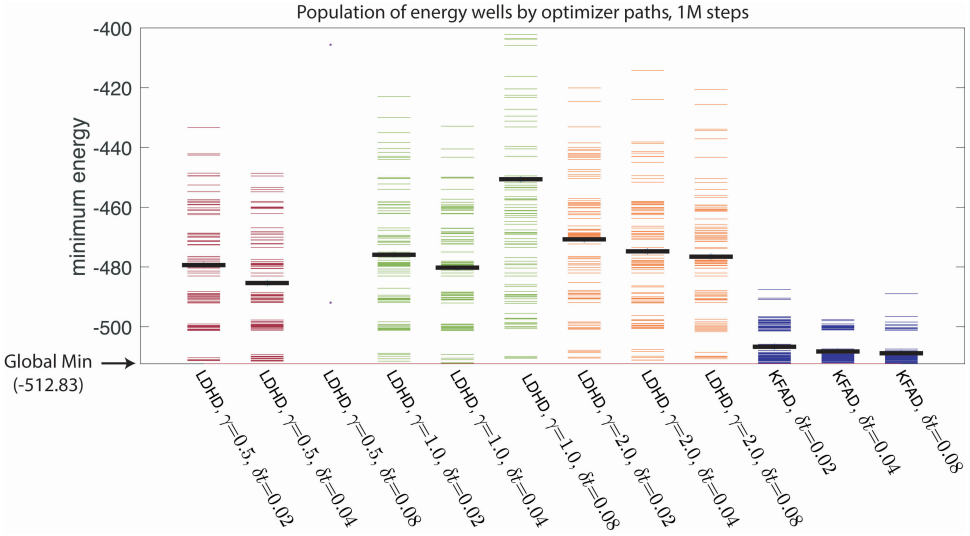


Figure 3.21: The energies of the 64 atom Morse cluster obtained in runs of 10^6 steps (compare with Fig. 3.20). We call attention to the fact that for $h = 0.08$ in KFAD (last column), nearly all the trajectories that had previously landed in the well at depth around -500 have moved into the minimum energy state (or very near of it).

A plausible explanation for the substantial improvement of KFAD over LDHD in this case is the following. Compared to LDHD, the KFAD method, which is used here without linear damping (i.e. $\gamma = 0$), has very mild damping in the later stages of minimization (when $\|p\|$ is small). Compared to LDHD, KFAD looks more like Hamiltonian dynamics at low kinetic energy. This increases the ability of the method to continue to explore the low-lying minima of the potential landscape. We plan to further study this issue in application settings in subsequent work.

Chapter 4

Higher-Order Damping Mechanisms for Neural Network Training

4.1 Introduction

Machine learning landscapes are highly non-convex, multi-modal and often very hard to navigate [15], [76]. Finding high quality minima fast is therefore one of the key challenges. Loss landscape complexity, noisy gradients, model architecture as well as the choice of optimiser hyperparameters can all potentially excite oscillatory modes of the dynamics, which in turn can have a detrimental effect on the convergence of the optimisation algorithm.

Appropriate control of the training process is crucial to improve efficiency, enhance generalization, allow for more extensive investigation of the network parameters and even to reduce the carbon footprint of training. An important aspect of training using SGD and popular variants is the proper choice of the momentum coefficient which can help to suppress oscillations in directions of high loss landscape curvature and improve convergence in low curvature regions. Although it is standard practice to fix the momentum coefficient during training, some authors have advocated scheduling this hyperparameter. Specifically, the authors of [131] propose a scheduling scheme for the momentum inspired from [92] and suggest a gradual increase of the momentum parameter as training progresses.

Employing methods that use exponentially weighted averages of the gradients, such as mSGD can help control oscillations introduced both from noisy gradients and from the steepness of the landscape and improve stability. Adaptive learning rate methods such as Adam, RMSprop, or Adagrad, which allow for an individual learning rate for each model parameter can also help alleviate the effects of noisy gradients to some extent. However, adaptive step size methods have been demonstrated to generalize poorly, often much worse than mSGD [148].

One of the most popular optimisation methods for machine learning applications is momentum stochastic gradient descent (mSGD). The continuous dynamics corresponding to the discrete mSGD equations [110] correspond to linearly dissipative Hamiltonian dynamics (LDHD) [34, 82, 122]

$$\dot{x} = p, \tag{4.1}$$

$$\dot{p} = \nabla f(x) - \gamma p, \tag{4.2}$$

where the mSGD momentum parameter is directly related to the linear friction/dissipation coefficient γ. Standard mSGD/LDHD employs a fixed friction coefficient γ throughout training. This means, the same dissipation is applied on the momenta throughout the optimisation process regardless of local landscape and curvature variations and the kinetic energy of the system.

Having very high momenta during the optimisation process can lead to oscillations and instability and therefore be detrimental in terms of convergence. On the other hand, if we set set the linear friction γ to be too high to avoid high momenta, then this leads to excessive energy dissipation and the momentum mechanism starts being a hindrance rather than an aid. Having a friction that dynamically adapts based on the kinetic energy allows us to increase the damping when momenta are getting too high and decrease it when momenta are lower. This acts like a gentle thermostat on the system, allowing us to control the magnitude of the momenta. This adaptive friction mechanism was introduced in the context of optimisation in [58]. The dynamics are as follows

$$\dot{x} = p \tag{4.3}$$

$$\dot{p} = -\nabla f(x) - \gamma p - \xi * p, \tag{4.4}$$

$$\dot{\xi} = \frac{[p]^2}{\mu} - \alpha\xi, \tag{4.5}$$

where $x, p, \xi \in \mathbb{R}^d$ and "$*$", "$[\cdot]^n$" denote element-wise multiplication and exponentiation respectively.

The benefits of adapting the momentum parameter μ during training have also been demonstrated in [131], where a time-dependent formula for μ is introduced based on theoretical convergence results by [92, 93].

Both [131] and [92] advocate the gradual increase of the momentum parameter μ for non strongly convex functions, which is usually the case for loss landscapes encountered in neural network training. Our scheme of regulating the friction based on the kinetic energy aligns with this reasoning. The equilibrium of the continuous dynamics for momentum gradient descent

4.1, 4.2 corresponds to $(\dot{x}, \dot{p}) = 0 \rightarrow (F(x^*), p^*) = (0, 0)$. Therefore, as the dynamics moves towards equilibrium, the kinetic energy (momenta p) of the system decreases which means that the adaptive friction also decreases, which corresponds to an increase in the momentum parameter μ as we move towards equilibrium.

The use of cubic damping as a vibration suppression mechanism has been investigated in several engineering papers. In [100], the authors examine the effect of adding a cubic damping component in three different simplified models of a vehicle suspension system. They find that cubic damping can be beneficial in single degree of freedom (SDOF) models with force excitation. Furthermore, [105] shows that adding cubic and quintic damping components to a force-excited SDOF isolator system can significantly reduce vibration transmissibility over the resonant frequency range of the system while having an almost negligible effect on transmissibility over frequency ranges that are below or above the resonant frequency range. The authors point out that this behaviour is desirable and helps overcome a key issue with linear damping systems where increasing the damping in order to reduce vibrations over the resonant frequency range also has the undesirable side-effect of increasing vibration transmissibility over high frequency ranges (high linear damping in that case leads to the system behaving like a rigid body). The authors of [155] also examine the effect of antisymetric (involving odd powers of the momenta) nonlinear damping in the case of general (non-harmonic) loadings for an SDOF system and demonstrate that cubic damping is also beneficial in that scenario.

The aforementioned articles are concerned with the study of physical/mechanical systems modelled as systems of ODEs. They study a structure set in motion by an excitation force (introduced by moving on a landscape/road). A parallel can therefore be made between such mechanical systems and optimiser ODE systems. The geometry of the optimisation loss landscape offers the “excitation” at each point of the trajectory, which combined with the optimiser dynamics and choice of hyperparameters can lead to resonance phenomena.

In this work, we demonstrate that the use of individual adaptive friction parameters and a cubic damping mechanism in the optimiser dynamics can significantly speed up convergence, matching and even surpassing Adam performance on the training set, while maintaining the generalization properties of mSGD.

4.2 Algorithms

4.2.1 Individual Kinetic Friction Adaptive Descent (iKFAD)

We augment the continuous dynamics for momentum gradient descent 4.1, 4.2 by introducing an auxilliary variable $\xi \in \mathbb{R}^d$ that acts as an adaptive friction coefficient regulated based on the kinetic energy/temperature of the system. We build on work done in [58], but examine the case where ξ is a vector rather than a scalar, i.e. we introduce an individual adaptive friction coefficient for each degree of freedom. This allows us to control the momenta of each parameter and treat each descent direction individually. For $x, p, \xi \in \mathbb{R}^d$ we have

$$\dot{x} = p, \tag{4.6}$$

$$\dot{p} = -\nabla f(x) - \xi * p - \gamma p, \tag{4.7}$$

$$\dot{\xi} = \frac{[p]^2}{\mu} - \alpha\xi, \tag{4.8}$$

where $\gamma, \alpha, \mu > 0$ and "$*$" and "$[\cdot]^n$" denote element-wise multiplication and exponentiation respectively. The adaptive friction ξ therefore acts as a control mechanism that regulates the system's temperature, keeping the kinetic energies within a reasonable range, similarly to the function of a thermostat in molecular dynamcics [47, 55, 99].

We use the following splitting scheme

$$\begin{pmatrix} \dot{x} \\ \dot{p} \\ \dot{\xi} \end{pmatrix} = \underbrace{\begin{pmatrix} p \\ 0 \\ 0 \end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix} 0 \\ -\nabla f(x) \\ 0 \end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix} 0 \\ -\xi * p \\ \frac{[p]^2}{\mu} - \alpha\xi \end{pmatrix}}_{\text{C}} + \underbrace{\begin{pmatrix} 0 \\ -\gamma p \\ 0 \end{pmatrix}}_{\text{D}}. \tag{4.9}$$

The discretization of each of the individual components in the above splitting is as follows (see [58] for more details): **Step A:** $x_{n+1} = x_n + \delta t p_n$. **Step B:** $p_{n+1} = p_n + \delta t F(x_n)$. **Step C:** $p_{n+1/2} = \exp(-\xi_n \delta t) p_n$, $\xi_{n+1} = \exp(-\alpha\delta t)\xi_n + (\alpha\mu)^{-1}[p_{n+1/2}]^2(1 - \exp(-\alpha\delta t))$ and $p_{n+1} = \exp(-\xi_n \delta t) p_{n+1/2}$. **Step D:** $p_{n+1} = \exp(-\gamma\delta t) p_n$.

It is interesting to examine the effect of the initialisation of adaptive friction ξ as well that of the momenta p on the performance of iKFAD on some standard ML datasets. We produce $\gamma - h$ plots colored according to the ratio of training accuracies of the two scenarios being compared in each case. Green areas in the graphs that follow (Figures 4.1,4.3,4.5, 4.6) are areas where the accuracy ratio is greater than one and red indicates that the ratio is less than one. White means that the ratio is equal to one, which means the two methods being compared yield the same accuracy in white regions.

We first examine the effect that the initialisation of the adaptive friction ξ has on the training accuracy of iKFAD. For this experiment, we initialise the momenta to zero and set $\alpha = 1$, $\mu = 1$. The initialisation of ξ does not seem to be making a great difference when training on MNIST, FMNIST, SST2 and CIFAR-10. For CIFAR-100 we see some benefit of initialising ξ to one in the lower γ, higher step size region of the CIFAR-100 graph in Figure 4.1. Initialising ξ to zero seems to be beneficial in the lower h (red) region in the CIFAR-100 graph. However, the actual accuracy values in this region are quite low for both methods making this region less important. We therefore see that when the momenta are initialised to zero and $\alpha = \mu = 1$, the initialiation of ξ does not seem to make a great difference in the performance of iKFAD. In order to produce the $\gamma - h$ plots in figures 4.1-4.6, we used the following neural network architectures: For MNIST and FMNIST a Resnet18 architecture was used. For CIFAR-10 and CIFAR-100, we used a Resnet34 architecture and for SST-2, a DistilBERT network was used. All tasks involve a binary classification problem, therefore we use a binary cross-entropy loss function.

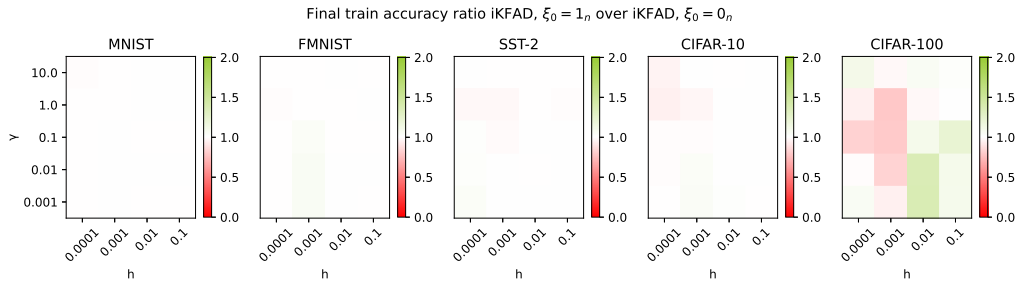


Figure 4.1: Effect of the initialisation of the adaptive friction ξ in iKFAD. We compare iKFAD with $\xi_0 = 0_n$ to iKFAD with $\xi_0 = 1_n$. The momenta are initialised as $p_0 = 0_n$ in both cases and we set $\alpha = 0.1$, $\mu = 0.0001$. We see that initialising ξ to zero versus one does not seem to dramatically affect final training accuracy. For CIFAR-100 initialising ξ to one seems to lead to a slight performance improvement over initialising it to one for higher step sizes and lower γ's, but the situation is reversed for lower step sizes.

We also repeat the above comparison in the case where $\gamma = 0$. The results can be seen in Figure 4.2. We can see that - again - initialising ξ to zero versus one does not seem to make a big difference. For CIFAR-100, $\xi_0 = 1$ seems to be better for intermediate step sizes but for the rest of the step size values/datasets, the two initialisations yield similar performance.

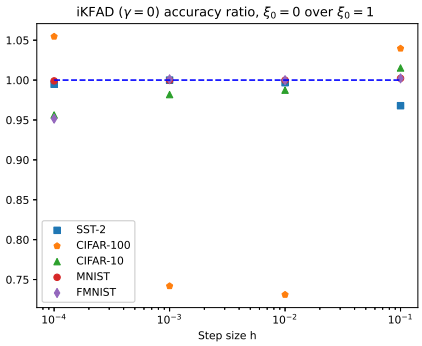


Figure 4.2: Examining effect of ξ initialisation on training accuracy.

Next, we examine the effect that the initialisation of the momenta has on the training accuracy of iKFAD. For this experiment, we set $\xi_0 = 0$, $\alpha = 1$ and $\mu = 1$ (similar results to those of Figure 4.3 are also obtained for $\alpha = 0.1$ and $\mu = 0.001$). We see that initialising the momenta to zero can lead to significantly higher accuracies especially in the CIFAR-10 and CIFAR-100 benchmarks.

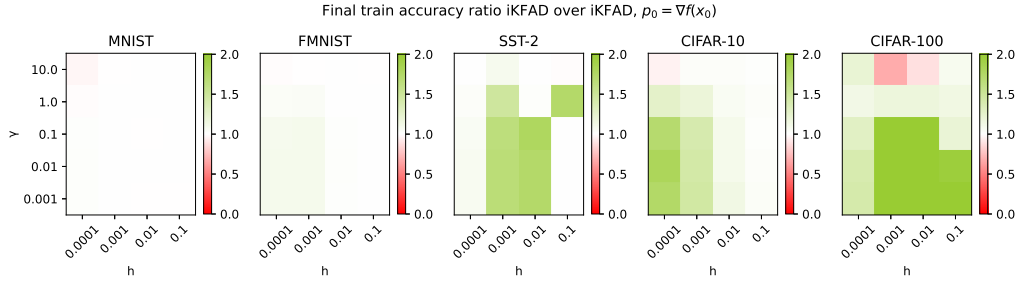


Figure 4.3: Effect of the initialisation of the momenta p in iKFAD. We compare iKFAD with $p_0 = 0_n$ to iKFAD with $p_0 = \nabla f(x)$. The adaptive friction is initialised as $\xi_0 = 0_n$ in both cases and we set $\alpha = 1$, $\mu = 1$ (Using $\alpha = 0.1$, $\mu = 0.001$ yields similar results, but the difference between the two initialisation methods becomes less pronounced). Green areas indicate that a zero initialisation is better than initialising the momenta to be equal to the gradients. We observe that a cold initialisation leads to a higher accuracy in most cases for SST-2, CIFAR-10 and CIFAR-100.

Finally, we examine the effect that the linear friction parameter γ has on the performance of iKFAD. Having a non-zero linear friction coefficient γ seems to improve performance when $\alpha = 1$, $\mu = 1$ (Figure 4.4). However, this is due to the low values of the momenta p encountered during training which necessitate the use of lower μ values in order for ξ to meaningfully contribute to the dynamics. This is confirmed by the results in Figure 4.5. When we use a lower value for μ, the use of a linear friction γ does not seem to be making as big of a difference.

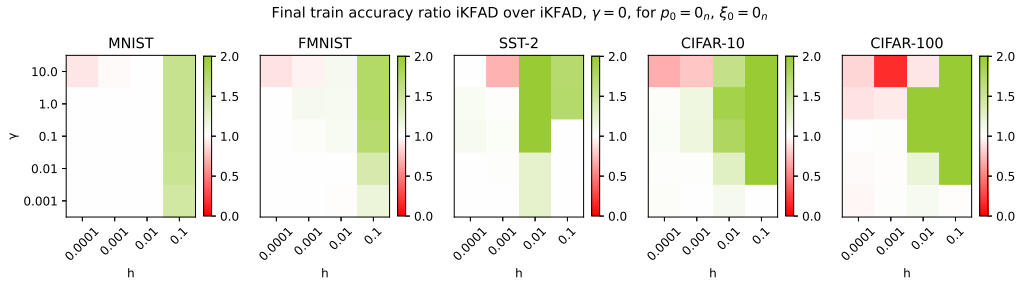


Figure 4.4: Effect of linear friction γ in iKFAD. We compare iKFAD with $\gamma = 0$ to iKFAD with non-zero γ. The momenta and adaptive friction are initialized as $p_0 = 0_n$ and $\xi_0 = 0_n$ respectively. We set $\alpha = 1$, $\mu = 1$. Note that $\alpha = \mu = 1$ in combination with low momenta values lead to the dynamics of iKFAD being very similar to those of with LDHD in the sense that the adaptive friction mechanism does not meaningfully contribute to the dynamics and we must rely on the linear damping γ for dissipation.

Repeating the experiment from Figure 4.4 for $\alpha = 0.1$, $\mu = 0.0001$, gives us the results in Figure 4.5. We see that when using lower values for μ and α, the fixed linear friction γ is not as important as the adaptive friction mechanism gets activated when the product $\alpha\mu$ becomes lower.

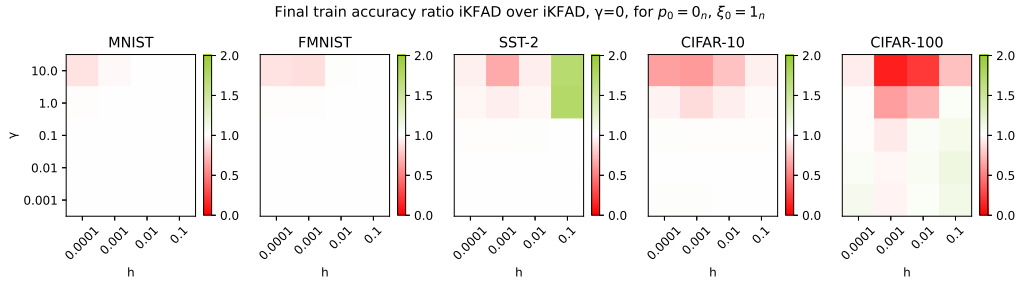


Figure 4.5: Effect of linear friction γ in iKFAD. We compare iKFAD with $\gamma = 0$ to iKFAD with iKFAD with non-zero γ. In both cases we initialise the momenta and adaptive friction as $p_0 = 0_n$ and $\xi_0 = 1_n$ respectively and we set $\alpha = 0.1$, $\mu = 0.0001$.

Given the above results, we choose a "cold" initialisation for both iKFAD and mSGD and compare the two methods. For iKFAD, we also set $\gamma = 0$ to further simplify the dynamics and we set $\alpha = 0.1$ and $\mu = 0.0001$. The value for μ was chosen after trying different values in $\{0.0001, 0.001, 0.01, 0.1, 1\}$. We observed that lower μ values within that range led to improved iKFAD performance.

Finally, we also compare the performance of iKFAD for $\gamma = 0$, to mSGD in Figure 4.6. We see that even without a linear friction term, iKFAD outperforms mSGD for most $\gamma - h$ combinations in this setting.

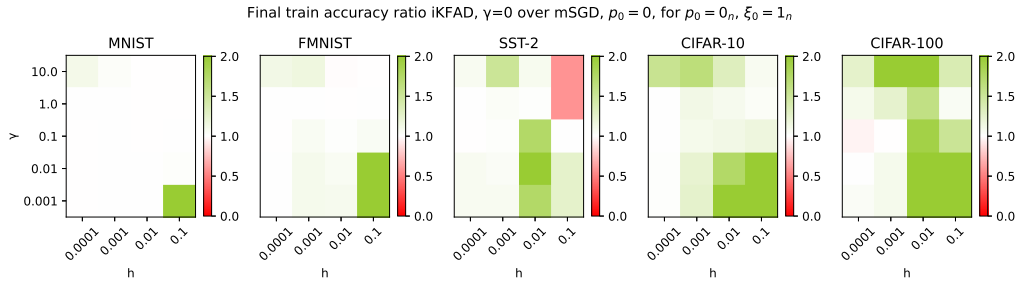


Figure 4.6: Comparison between iKFAD with $\gamma = 0$, $\xi_0 = 1_n$, $p_0 = 0_n$, to mSGD with $p_0 = 0_n$. For iKFAD, we set $\alpha = 0.1$, $\mu = 0.0001$. We see that iKFAD outperforms mSGD in terms of training accuracy for most $\gamma - h$ combinations especially for the SST-2, CIFAR-10 and CIFAR-100 datasets.

Note that iKFAD consistently outperforms mSGD even though we are not properly tuning α and μ and we have set $\gamma = 0$. We set $\gamma = 0$ to demonstrate that even without a fixed linear disspiation, the adaptive friction mechanism adds sufficient damping to the system and iKFAD can still outperform mSGD even without a linear friction γ.

In order to get an idea for how the adaptive friction coefficient ξ behaves during training, we plot the L1-norm of the adaptive friction vector ξ, as well as the $L1 - norm$ of the momenta. The results, together with the corresponding train and test accuracies can be seen in Figure 4.7. We see that there is - as expected - a direct correlation between the L1-norm of the momenta and the L1-norm of the ξ vector. For FMNIST and CIFAR-100, the value of the norm of ξ tends to spike and then decrease as we move towards an equilibrium. For SST-2, convergence is slower and ξ does not seem to be equilibrating in most cases. Note that we are not tuning the hyperparameters of the optimisers for these plots, hence the poor accuracies in some of the runs. Our goal was to obtain an understanding of how ξ behaves during training.

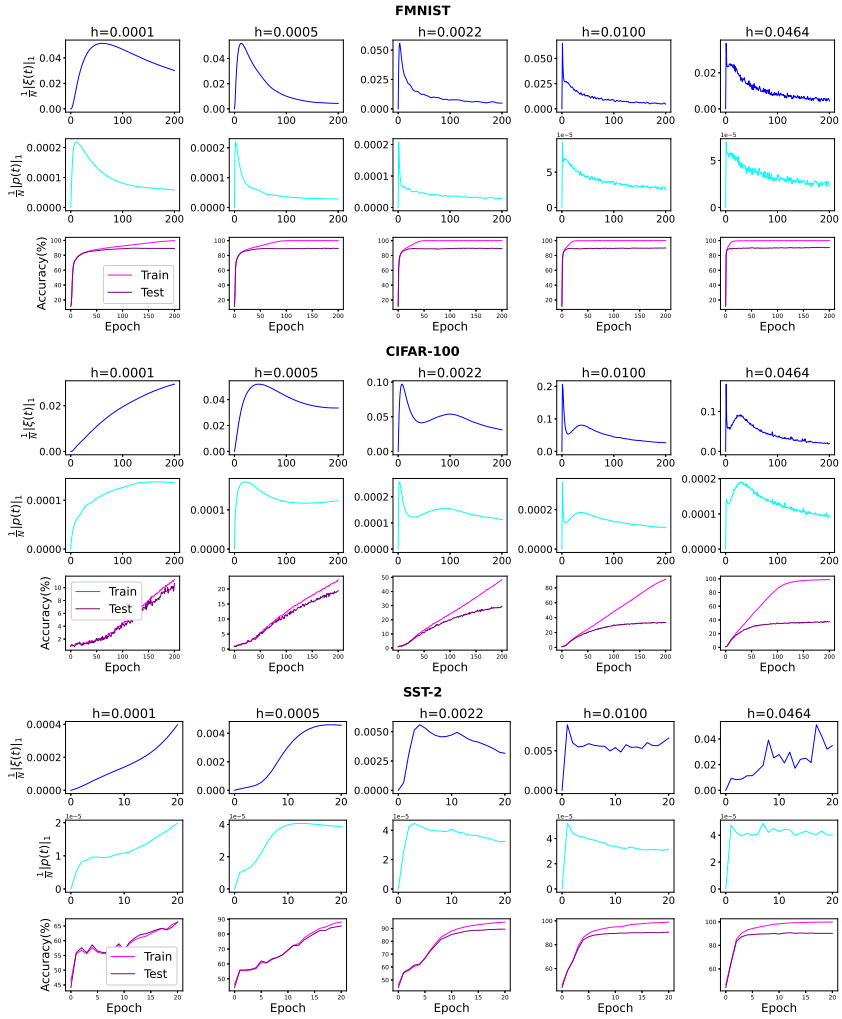


Figure 4.7: L1 norm of adaptive friction vector ξ for iKFAD, $\gamma = 0$, $\alpha = 0.1$, $\mu = 0.00001$. The different step sizes were selected from a logarithmic scale.

Dynamical properties of iKFAD

From equation 4.8, we have $\dot{\xi} = \frac{[p]^2}{\mu} - \alpha\xi$, therefore

$$\xi(t) = e^{-\alpha t}\xi(0) - \int_0^t e^{-\alpha(t-s)}\frac{[p]^2}{\mu}ds.$$

This means that, $\xi(t) \geq 0 \forall t > 0$, given $\xi(0) \geq 0$. Therefore ξ can be seen as a friction coefficient, similarly to γ. We next show that under some conditions on f the dynamics of (4.6)-(4.8) is well posed on infinite time horizons.

Lemma 4.2.1. *Assume that f is smooth and $f(x) \to +\infty$ as $\|x\| \to +\infty$. Then, for any initial condition $(x_0, p_0, \xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d_+$, the solution of* (4.6)-(4.8) *is well defined for all times $t \geq 0$, and there exists $R > 0$ such that*

$$\forall t \geq 0, \qquad \|x(t)\| \leq R, \quad \|p(t)\| \leq R, \quad \|\xi(t)\| \leq R.$$

Proof. The Cauchy–Lipschitz theorem ensures the existence and uniqueness of a solution for a positive time. To show that the solution is global in time, we introduce the following Lyapunov function

$$\mathscr{G}(x,p,\xi) = f(x) - f(x^*) + \frac{1}{2}\|p\|^2 + \frac{\mu}{2}\xi^2.$$

The function $\mathbb{G}(t) = \mathscr{G}(x(t), p(t), \xi(t))$ satisfies

$$\begin{aligned}\dot{\mathbb{G}}(t) &= \nabla f(x(t)) \cdot \dot{x}(t) + p(t)\dot{p}(t) + \mu\xi(t)\dot{\xi}(t) \\ &= -\gamma\|p(t)\|^2 - (\xi * p) \cdot p + \xi \cdot [p]^2 - \alpha\mu\xi(t)^2 \leq 0 \\ &= -\gamma\|p(t)\|^2 - \alpha\mu\xi(t)^2 \leq 0,\end{aligned}$$

where we used that $\xi(t) \geq 0$. This shows that $\mathbb{G}(t) \leq \mathbb{G}(0)$, from which the result easily follows since $f(x) - f(x^*) \geq 0, \quad \forall x \in \mathbb{R}^d$. □

The next result, whose derivation is straightforward, characterizes equilibria of the dynamics.

Proposition 4.2.2. *The equilibria of the system (4.6)-(4.8) correspond to*

$$(\dot{x}, \dot{p}, \dot{\xi}) = 0 \Rightarrow (p^*, F(x^*), \xi^*) = 0,$$

i.e. they coincide with the physical equilibria.

Exponential convergence of the continuous iKFAD dynamics

Next, we show exponential convergence of the iKFAD dynamics 4.6-4.8 to the equilibria of Proposition 4.2.2.

Theorem 4.2.3. *Assume that $\exists a, b > 0$ such that*

$$a\,[f(x) - f(x^*)] + b\|x - x^*\|^2 \leq (x - x^*) \cdot (\nabla f(x) - \nabla f(x^*)). \tag{4.10}$$

Then, for any initial condition $(x_0, p_0, \xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}_+$, and assuming $\gamma > 0$, there exist $\kappa > 0$ and $C \in \mathbb{R}_+$ such that the solution of (4.6)-(4.8) *satisfies*

$$f(x(t)) - f(x^*) + \|p(t)\| + \|\xi(t)\| \leq C\mathrm{e}^{-\kappa t}.$$

The condition (3.16) is satisfied for $f \in C^2$ with $0 < m \leq \nabla^2 f(x) \leq M < +\infty$. A similar condition is considered in [83].

Proof. Consider $\varepsilon > 0$ and introduce the following Lyapunov function (which is, up to the additional term $\|\xi\|^2$, a common choice for stochastic Langevin dynamics [83], also considered for dissipated Hamiltonian dynamics [90]):

$$\mathscr{W}_\varepsilon(x,p,\xi) = f(x) - f(x^*) + \frac{1}{2}\|p\|^2 + \frac{\mu}{2}\|\xi\|^2 + \varepsilon(x-x^*)\cdot p + \varepsilon\|x(t)-x^*\|^2. \tag{4.11}$$

A discrete Cauchy–Schwarz inequality implies, for $\varepsilon \in [0,1/2]$, the lower bound

$$\mathscr{W}_\varepsilon(x,p,\xi) \geq f(x) - f(x^*) + \frac{1}{4}\|p\|^2 + \frac{\mu}{2}\|\xi\|^2 + \frac{\varepsilon}{2}\|x(t)-x^*\|^2, \tag{4.12}$$

as well as the upper bound

$$\mathscr{W}_\varepsilon(x,p,\xi) \leq f(x) - f(x^*) + \frac{3}{4}\|p\|^2 + \frac{\mu}{2}\|\xi\|^2 + \frac{3\varepsilon}{2}\|x(t)-x^*\|^2. \tag{4.13}$$

A simple computation shows that the function $\mathscr{W}_\varepsilon(t) = \mathscr{W}_\varepsilon(x(t),p(t),\xi(t))$ satisfies

$$\begin{aligned}
\dot{\mathscr{W}}_\varepsilon(t) &= \nabla f(x(t))\cdot\dot{x}(t) + p(t)\dot{p}(t) + \mu\xi(t)\dot{\xi}(t) + \varepsilon p(t)\cdot\dot{x}(t) \\
&\quad + \varepsilon\dot{p}(t)\cdot(x(t)-x^*) + 2\varepsilon\dot{x}(t)\cdot(x(t)-x^*) \\
&= -(\gamma-\varepsilon)\|p(t)\|^2 - \alpha\mu\|\xi(t)\|^2 - \varepsilon(x(t)-x^*)\cdot\nabla f(x(t)) \\
&\quad + \varepsilon(2-\gamma)(x(t)-x^*)\cdot p(t) - \varepsilon(x(t)-x^*)\cdot(\xi * p) \\
&\leq -(\gamma-\varepsilon)\|p(t)\|^2 - \alpha\mu\|\xi(t)\|^2 - \varepsilon(x(t)-x^*)\cdot\nabla f(x(t)) \\
&\quad + \varepsilon(2+\gamma)\left(\eta\|x(t)-x^*\|^2 + \frac{1}{4\eta}\|p(t)\|^2\right) + \varepsilon\left(\delta\|x(t)-x^*\|^2 + \frac{1}{4\delta}\|\xi(t)*p(t)\|^2\right) \\
&\leq -(\gamma-\varepsilon)\|p(t)\|^2 - \alpha\mu\|\xi(t)\|^2 - \varepsilon(x(t)-x^*)\cdot\nabla f(x(t)) \\
&\quad + \varepsilon(2+\gamma)\left(\eta\|x(t)-x^*\|^2 + \frac{1}{4\eta}\|p(t)\|^2\right) + \varepsilon\left(\delta\|x(t)-x^*\|^2 + \frac{R^2}{4\delta}\|p(t)\|^2\right),
\end{aligned}$$

where we used Cauchy–Schwarz inequalities and Lemma 4.2.1. Specifically, $\|\xi\| \leq R \Rightarrow \sum\xi_i^2 \leq R^2 \Rightarrow \xi_i^2 \leq R^2$. We can therefore bound $\|\xi * p\|^2$ as $\|\xi * p\|^2 = \sum(\xi_i p_i)^2 \leq R^2\|p\|^2$. Using assumption 4.10, we have

$$\begin{aligned}
\dot{\mathscr{W}}_\varepsilon(t) \leq &-\left[\gamma - \varepsilon\left(1 - \frac{R^2}{4\delta} - \frac{2+\gamma}{4\eta}\right)\right]\|p(t)\|^2 - \alpha\mu\|\xi(t)\|^2 - a\varepsilon(f(x)-f(x^*)) \\
&- \varepsilon\Big(b - (2+\gamma)\eta - \delta\Big)\|x(t)-x^*\|^2.
\end{aligned}$$

Setting $\eta = \frac{b}{4(2+\gamma)}$ and $\delta = b/4$, we obtain

$$\dot{\mathscr{W}}_\varepsilon(t) \leq -\left[\gamma - \varepsilon\left(1 - \frac{R^2-(2+\gamma)^2}{b}\right)\right]\|p(t)\|^2 - \alpha\mu\|\xi(t)\|^2 - a\varepsilon(f(x)-f(x^*)) - \frac{\varepsilon b}{2}\|x(t)-x^*\|^2 \Rightarrow$$
$$\dot{\mathscr{W}}_\varepsilon(t) \leq -\min\left\{\frac{4}{3}\left[\gamma - \varepsilon\left(1 - \frac{R^2-(2+\gamma)^2}{b}\right)\right], 2\alpha, a\varepsilon, \frac{b}{3}\right\}\mathscr{W}_\varepsilon(t),$$

from which exponential convergence follows from Grönwall's inequality. □

Convergence analysis of discrete iKFAD dynamics

Condider the following splitting of the continuous dynamics 4.6-4.8

$$\begin{pmatrix}\dot{x}\\ \dot{p}\\ \dot{\xi}\end{pmatrix} = \underbrace{\begin{pmatrix}p\\ 0\\ 0\end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix}0\\ -\nabla f(x)\\ 0\end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix}0\\ -\xi * p\\ \frac{[p]^2}{\mu}\end{pmatrix}}_{\text{C}} + \underbrace{\begin{pmatrix}0\\ -\gamma p\\ -\alpha\xi\end{pmatrix}}_{\text{D}}, \tag{4.14}$$

and the integration scheme $CDBA$. For step C, similarly to the analysis performed in [58], we observe that $p_i^2 + \mu\xi_i^2$ is an invariant quantity. Using this and defining $\Xi = \sum_{i=1}^d \xi_i e_i \bigotimes e_i$, where $e_i \in \mathbb{R}^d$ is a vector with zeroes in all entries apart from entry i, which is equal to one, the integration scheme $CDBA$ can be written as follows

$$\tilde{p}_{n+1/2} = e^{-\Xi_n \delta t} p_n \tag{4.15}$$
$$\tilde{\xi}_{n+1} = \xi_n \sqrt{1 + \frac{(I - e^{-2\Xi_n\delta t})[p_n]^2}{\mu[\xi_n]^2}} \tag{4.16}$$
$$\tilde{p}_{n+1} = e^{-\gamma\delta t}\tilde{p}_{n+1/2} \tag{4.17}$$
$$\xi_{n+1} = e^{-\alpha\delta t}\tilde{\xi}_{n+1} \tag{4.18}$$
$$p_{n+1} = \tilde{p}_{n+1} - \delta t \nabla f(x_n) \tag{4.19}$$
$$x_{n+1} = x_n + \delta t\, p_{n+1}. \tag{4.20}$$

Upon defining $\beta_{n,\delta t} = e^{-(\gamma I + \Xi_n)\delta t}$, we can rewrite equations (4.15)-(4.20) as

$$p_{n+1} = \beta_{n,\delta t} p_n - \delta t \nabla f(x_n) \tag{4.21}$$
$$x_{n+1} = x_n + \delta t \beta_{n,\delta t} p_n - \delta t^2 \nabla f(x_n) \tag{4.22}$$
$$\xi_{n+1} = e^{-\alpha\delta t}\sqrt{[\xi_n]^2 + \frac{(I - e^{-2\Xi_n\delta t})[p_n]^2}{\mu}}. \tag{4.23}$$

Proposition 4.2.4. *A state $s = (x, p, \xi)$ is an equilibrium point of the discrete iKFAD dynamics given by equations (4.21)-(4.23) if and only if it is an equilibrium of the continuous iKFAD dynamics (4.6)-(4.8).*

Proof. Let s^* be an equilibrium of the continuous dynamics as described in proposition 4.2.2 and let $s_n = (x_n, p_n, \xi_n)$ be an equilibrium point of the discrete dynamics (4.21)-(4.23) . This implies that if we take a step of the discrete dynamics starting from s_n, we will remain at equilibrium, that is $s_{n+1} = s_n$. Using equation (4.20) and setting $x_{n+1} = x_n$ gives us $p_{n+1} = 0$ which implies $p_n = 0$. Substituting $p_n = 0$ into equation (4.23) and using $\xi_{n+1} = \xi_n$, yields $\xi_n = e^{-\alpha\delta t}\xi_n \Rightarrow \xi_n = 0$ as $\alpha\delta t > 0$. Finally, for $p_{n+1} = p_n = 0$, equation (4.21) yields $\nabla f(x_n) = 0$. The above implies $s_n = s^*$, that is, every equilibrium point of the discrete dynamics is also an equilibrium point of the continuous dynamics. Thus the discretization does not introduce any "artificial" equilibrium points.

We will finally show that any equilibrium point of the continuous dynamics is also an equilibrium of the discrete dynamics. Consider an equilibrium $s^* = (x^*, p^*, \xi^*) = (x^*, 0, 0)$ such that $\nabla f(x^*) = 0$. Let $z_n = z^*$ and let us take a step following the discrete dynamics as given by equations (4.21)-(4.23). It is straightforward to see that $z_n = (x^*, 0, 0)$ leads to $z_{n+1} = z_n = z^*$. □

Theorem 4.2.5. *Consider a function $f \in C^2$ with $0 < m \leq \nabla^2 f(x) \leq M < +\infty$ and assume $\gamma > 0, L > 0$. Then, for any initial condition $(x_0, p_0, \xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d$ such that*

$$\|x_0\| + \|p_0\| + \|\xi_0\| \leq L,$$

there exist $\delta t^\star > 0$, $\kappa > 0$ and $C > 0$ (depending on L) such that

$$\forall \delta t \in (0, \delta t^\star), \quad \forall n \geq 0, \qquad f(x_n) - f(x^*) + \|p_n\|^2 + \|\xi_n\|^2 \leq C\mathrm{e}^{-\kappa n \delta t}.$$

Proof. This proof follows the same steps as the convergence analysis of the discretized FAD dynamics presented in [58], the only difference being the fact that ξ in the iKFAD dynamics is a vector, rather than a scalar, as was the case for FAD and that the matrix $A(x)$ involved in the FAD dynamics is now taken to be the identity $A(x) = I$. □

4.2.2 Cubically Damped Momentum Gradient Descent (CD)

Let us examine the dynamics (4.6)-(4.8). Assuming that α is large and that we are near equilibrium, i.e. $\dot{\xi} = 0$, we can solve (4.8) for ξ and obtain $\xi \sim (\alpha\mu)^{-1}[p]^2$. Plugging this into (4.7), we obtain

$$\dot{p} \sim -\nabla f(x) - (\alpha\mu)^{-1}[p]^3.$$

This observation [58] leads us to the introduction of our next optimiser. The system of equations describing the optimiser dynamics consists of a Hamiltonian part combined with a linear dissipation term similarly to mSGD but the dynamics is also augmented by a cubic dissipation term. Cubic damping is as previously mentioned used as a vibration suppression mechanism in engineering applications [100, 155] and was studied extensively in the scalar case by A. Babister [6]. The dynamics of the optimiser are as follows

$$\dot{x} = p, \tag{4.24}$$

$$\dot{p} = -\nabla f(x) - \gamma p - c[p]^3 \ , \tag{4.25}$$

where $c > 0$ and "$[\cdot]^n$" denotes element-wise exponentiation to the n^{th} power.

Dynamical properties of CD

In this section, we study the dynamical properties of the system (4.24)-(4.25). We propose a Lyapunov function to prove exponential convergence of the dynamics under the assumption that f is strongly convex.
A first result is that the dynamics is well-posed on infinite time horizons under some conditions on f.

Lemma 4.2.6. *Assume f is a smooth function and $f(x) \to \infty$ as $\|x\| \to +\infty$. Then, for any initial condition $(x_0, p_0) \in \mathbb{R}^d \times \mathbb{R}^d$, the solution of (4.24)-(4.25) is well defined $\forall t \geq 0$ and $\exists R \in \mathbb{R}_+$ such that $\|x(t)\| \leq R$ and $\|p(t)\| \leq R$, $\forall t \geq 0$.*

Proof. Existence and uniqueness of the solution of (4.24)-(4.25) follow from the Cauchy-Lipschitz theorem. To prove that the solution is global in time we define the following Lyapunov function

$$\mathscr{G}(x, p, \xi) = f(x) - f(x^*) + \frac{1}{2}\|p\|^2.$$

A simple computation shows that the function $\mathscr{G}(t) = \mathscr{G}(x(t), p(t), \xi(t))$ satisfies

$$\begin{aligned}\dot{\mathscr{G}}(t) &= \nabla f(x(t)) \cdot \dot{x}(t) + p(t)\dot{p}(t) \\ &= -\gamma\|p(t)\|^2 - cp(t) \cdot p(t)^3 \leq 0,\end{aligned}$$

since $\gamma, c \geq 0$. This means that $\mathscr{G}(t) \leq \mathscr{G}(0)$, from which the result easily follows since $f - f(x^*)$ and $\|p\|^2$ are non-negative. □

The next proposition, whose proof is immediate, characterizes the equilibria of the dynamics.

Proposition 4.2.7. *The equilibria of the system (4.24)-(4.25) coincide with the physical equilibria, i.e.*

$$(\dot{x}, \dot{p}) = 0 \Rightarrow (p^*, F(x^*)) = 0$$

Theorem 4.2.8. *Assume that* $\gamma, c > 0$ *and that* $\exists a, b > 0$:

$$a\,[f(x) - f(x^*)] + b\|x - x^*\|^2 \leq (x - x^*) \cdot (\nabla f(x) - \nabla f(x^*))\,. \tag{4.26}$$

Condition (3.16) *is satisfied for a twice continuously differentiable function* $f \in C^2$ *with* $0 < m \leq \nabla^2 f(x) \leq M < +\infty$.

Then, for any initial condition $(x_0, p_0, \xi_0) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}_+$, *there exist* $\kappa > 0$ *and* $C \in \mathbb{R}_+$ *such that the solution of* (4.24)-(4.25) *satisfies*

$$f(x(t)) - f(x^*) + \|p(t)\| \leq C\mathrm{e}^{-\kappa t}.$$

Proof. Consider $\varepsilon > 0$ and introduce the following Lyapunov function (which is a common choice for stochastic Langevin dynamics [83], also considered for dissipated Hamiltonian dynamics [90]):

$$\mathscr{V}_\varepsilon(x, p) = f(x) - f(x^*) + \frac{1}{2}\|p\|^2 + \varepsilon(x - x^*) \cdot p + \varepsilon\|x(t) - x^*\|^2. \tag{4.27}$$

Using a Cauchy–Schwarz inequality we can derive the following upper bound, for $\varepsilon \in [0, 1/2]$

$$\mathscr{V}_\varepsilon(x, p) \leq f(x) - f(x^*) + \frac{3}{4}\|p\|^2 + \frac{3\varepsilon}{2}\|x(t) - x^*\|^2. \tag{4.28}$$

We define $\mathscr{V}_\varepsilon(t) = \mathscr{V}_\varepsilon(x(t), p(t))$. The derivative of $\mathscr{V}_\varepsilon(t)$ satisfies

$$\begin{aligned}
\dot{\mathscr{V}}_\varepsilon(t) &= \nabla f(x(t)) \cdot \dot{x}(t) + p(t) \cdot \dot{p}(t) + \varepsilon \dot{x}(t) \cdot p(t) \\
&\quad + \varepsilon(x(t) - x^*) \cdot \dot{p}(t) + 2\varepsilon(x(t) - x^*) \cdot \dot{x}(t) \\
&= -(\gamma - \varepsilon)\|p(t)\|^2 - \varepsilon(x(t) - x^*) \cdot \nabla f(x(t)) \\
&\quad + \varepsilon\,(2 - \gamma)\,(x(t) - x^*) \cdot p(t) - c\varepsilon(x(t) - x^*) \cdot p(t)^3 - c p(t) \cdot p(t)^3 \\
&\leq -(\gamma - \varepsilon)\|p(t)\|^2 - \varepsilon(x(t) - x^*) \cdot \nabla f(x(t)) \\
&\quad + \varepsilon\,(2 + \gamma)\left(\delta\|x(t) - x^*\|^2 + \frac{1}{4\delta}\|p(t)\|^2\right) + c\varepsilon\left(\eta\|x(t) - x^*\|^2 + \frac{R^4}{4\eta}\|p(t)\|^2\right),
\end{aligned}$$

where we used a weighted Young's inequality (with $\delta, \eta > 0$) to bound the last two terms as well as Lemma 4.2.6, to bound the norm of the element-wise cubed momentum vector as $\|p^3\|^2 = \sum (p_i^3)^2 = \sum (p_i^2)^3 \leq (\sum p_i^2)^3 = (\|p\|^2)^3 \leq \|p\|^2 R^4$. We have also used the fact that $p \cdot p^3 \geq 0$.

In view of Condition (4.26), we have

$$\dot{\mathscr{V}}_\varepsilon(t) \leq -a\varepsilon(f(x)-f(x^*)) - \left(\gamma - \varepsilon\left(1+\frac{(2+\gamma)^2}{b}+c^2\frac{R^4}{b}\right)\right)\|p(t)\|^2 - \frac{b\varepsilon}{2}\|x(t)-x^*\|^2,$$

where we set $\delta = \frac{b}{4(2+\gamma)}$ and $\eta = \frac{b}{4c}$. This bound for $\dot{\mathscr{V}}_\varepsilon(t)$, combined with the upper bound in equation (4.28) allows us to conclude exponential convergence of the dynamics (4.24), (4.25) with

$$\dot{\mathscr{V}}_\varepsilon(t) \leq -\min\left\{\alpha\varepsilon, \frac{b}{3}, \frac{4}{3}\left(\gamma - \varepsilon\left(1+\frac{(2+\gamma)^2}{b}+c^2\frac{R^4}{b}\right)\right)\right\}\mathscr{V}_\varepsilon.$$

For a sufficiently small ε we can use Gronwall's inequality to conclude exponential convergence of the dynamics. □

Numerical Discretization

For the numerical implementation of the dynamics (4.24), (4.25) we choose the following splitting scheme

$$\begin{pmatrix}\dot{x}\\ \dot{p}\end{pmatrix} = \underbrace{\begin{pmatrix}p\\ 0\end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix}0\\ -\nabla f(x)\end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix}0\\ -cp^3\end{pmatrix}}_{\text{C'}} + \underbrace{\begin{pmatrix}0\\ -\gamma p\end{pmatrix}}_{\text{D}}. \tag{4.29}$$

Steps A, B and D are the same as for projective McFAD. We perform the discretization for step C' as follows

$$\dot{p} = -cp^3 \Rightarrow \int \frac{\dot{p}}{p^3}dt = -c\int dt \Rightarrow$$
$$-\frac{1}{2p^2} = -ct + c_1 \Rightarrow p = \frac{1}{\sqrt{2ct+c_1}}.$$

Using the initial condition for the momenta $p(0) = p_0$, we have $c_1 = 1/p_0^2$, therefore

$$p = \frac{p_0}{\sqrt{2cp_0^2t+1}}. \tag{4.30}$$

Convergence analysis of discrete dynamics

We consider an Euler discretization of 4.24, 4.25, which is simpler to work with than the splitting scheme proposed in the previous subsection.

$$p_{n+1} = (1-\gamma\delta t)p_n - c\delta t[p_n]^3 - \delta t\nabla f(x_n), \tag{4.31}$$

$$x_{n+1} = x_n + \delta t\, p_n, \tag{4.32}$$

where we assume $\gamma\delta t$ is sufficiently small. We next consider a perturbation of the Lyapunov function 4.27

$$\mathscr{W}(x_n,p_n) = f(x_n) + A_{\delta t}\frac{\|p_n\|^2}{2} + B_{\delta t}\|x_n\|^2 + C_{\delta t}x_n\cdot p_n, \tag{4.33}$$

with coefficients $A_{\delta t}, B_{\delta t}, C_{\delta t}$ to be determined.

Proposition 4.2.9. *Consider $f\in C^2$ and assume $\exists m,M\in\mathbb{R}_+ : m\le\nabla^2 f(x)\le M\forall x\in\mathbb{R}^d$. Then, for any initial condition (x_0,p_0) such that*

$$\|x_0\| + \|p_0\| \le L,$$

where $L>0$, there exists $\delta t^*>0, r>0$ and $C>0$:

$$\forall n\ge 0 \text{ and } \forall\delta t\in(0,\delta t^*),\quad f(x_n)-f(x^*)+\|p_n\|^2 \le Ce^{-\kappa n\delta t},$$

where x_n, p_n are the iterates of the discrete dynamics (4.31)-(4.32)*.*

Proof. Assuming $0<m<\nabla^2 f(x)\le M$, where these upper and lower bounds on the Hessian correspond to m-strong convexity and M-smoothness respectively, we have

$$f(x_{n+1}) \le f(x_n) + \delta t\nabla f(x_n)\cdot p_n + \frac{\delta t^2 M}{2}\|p_n\|^2.$$

Next,

$$\begin{aligned}
\|p_{n+1}\|^2 =& \|(1-\gamma\delta t)p_n - c\delta t[p_n]^3 - \delta t\nabla f(x_n)\|^2\\
=& \left\|(1-\gamma\delta t)p_n - c\delta t[p_n]^3\right\|^2 - 2\left[(1-\gamma\delta t)p_n - c\delta t[p_n]^3\right]\delta t\nabla f(x_n) + \|\delta t\nabla f(x_n)\|^2\\
=& \|(1-\gamma\delta t)p_n\|^2 - 2c(1-\gamma\delta t)\delta t\, p_n\cdot[p_n]^3 + \|c\delta t[p_n]^3\|^2\\
& - 2\left[(1-\gamma\delta t)p_n - c\delta t[p_n]^3\right]\delta t\nabla f(x_n) + \|\delta t\nabla f(x_n)\|^2\\
\le& (1-\gamma\delta t)^2\|p_n\|^2 + c^2\delta t^2 R^4\|p_n\|^2\\
& - 2\delta t(1-\gamma\delta t)p_n\cdot\nabla f(x_n) + 2c\delta t^2\|[p_n]^3\cdot\nabla f(x_n)\| + \delta t^2\|\nabla f(x_n)\|^2\\
\le& \|p_n\|^2 - 2\gamma\delta t\|p_n\|^2 + \gamma^2\delta t^2\|p_n\|^2 + c^2\delta t^2R^4\|p_n\|^2\\
& - 2\delta t(1-\gamma\delta t)p_n\cdot\nabla f(x_n) + 2c\delta t^2\left(\frac{1}{4\theta}\|[p_n]^3\|^2 + \theta\|\nabla f(x_n)\|^2\right) + \delta t^2\|\nabla f(x_n)\|^2
\end{aligned}$$

$$
\begin{aligned}
&\leq \|p_n\|^2 - 2\gamma\delta t\|p_n\|^2 + \gamma^2\delta t^2\|p_n\|^2 + c^2\delta t^2 R^4\|p_n\|^2 \\
&\quad - 2\delta t(1-\gamma\delta t)p_n\cdot\nabla f(x_n) + 2c\delta t^2\Big(\frac{R^4}{4\theta}\|p_n\|^2 + \theta\|\nabla f(x_n)\|^2\Big) + \delta t^2\|\nabla f(x_n)\|^2 \\
&= \|p_n\|^2 - 2\gamma\delta t\|p_n\|^2 + \gamma^2\delta t^2\|p_n\|^2 + c^2\delta t^2 R^4\|p_n\|^2 \\
&\quad - 2\delta t p_n\cdot\nabla f(x_n) + 2\gamma\delta t^2 p_n\cdot\nabla f(x_n) + \frac{2cR^4}{4\theta}\delta t^2\|p_n\|^2 + 2c\delta t^2\theta\|\nabla f(x_n)\|^2 + \delta t^2\|\nabla f(x_n)\|^2 \\
&\leq \|p_n\|^2 - 2\gamma\delta t\|p_n\|^2 + \gamma^2\delta t^2\|p_n\|^2 + c^2R^4\delta t^2\|p_n\|^2 - 2\delta t p_n\cdot\nabla f(x_n) \\
&\quad + 2\gamma\delta t^2\left(\frac{1}{4\xi}\|p_n\|^2 + \xi\|\nabla f(x_n)\|^2\right) + \frac{2cR^4}{4\theta}\delta t^2\|p_n\|^2 + 2c\theta\delta t^2\|\nabla f(x_n)\|^2 + \delta t^2\|\nabla f(x_n)\|^2 \\
&\leq \|p_n\|^2 - 2\gamma\delta t\|p_n\|^2 - 2\delta t p_n\cdot\nabla f(x_n) + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big).
\end{aligned}
$$

Therefore we have

$$
A_{\delta t}\frac{\|p_{n+1}\|^2}{2} \leq A_{\delta t}\frac{\|p_n\|^2}{2} - A_{\delta t}\gamma\delta t\|p_n\|^2 - A_{\delta t}\delta t p_n\cdot\nabla f(x_n) + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big).
$$

We also have

$$
\begin{aligned}
\|x_{n+1}\|^2 &= \|x_n + \delta t p_n\|^2 = \|x_n\|^2 + 2\delta t x_n\cdot p_n + \delta t^2\|p_n\|^2 \Rightarrow \\
B_{\delta t}\|x_{n+1}\|^2 &= B_{\delta t}\|x_n\|^2 + 2B_{\delta t}\delta t x_n\cdot p_n + B_{\delta t}\delta t^2\|p_n\|^2 \\
&\leq B_{\delta t}\|x_n\|^2 + 2B_{\delta t}\delta t\|x_n\|\|p_n\| + B_{\delta t}\delta t^2\|p_n\|^2.
\end{aligned}
$$

Finally, using equations (4.31)-(4.32), we obtain,

$$
\begin{aligned}
x_{n+1}\cdot p_{n+1} &= \Big(x_n + \delta t p_n\Big)\cdot\Big((1-\gamma\delta t)p_n - c\delta t[p_n]^3 - \delta t\nabla f(x_n)\Big) \\
&= (1-\gamma\delta t)x_n\cdot p_n - c\delta t x_n\cdot[p_n]^3 - \delta t x_n\cdot\nabla f(x_n) + \\
&\quad (1-\gamma\delta t)\delta t p_n\cdot p_n - c\delta t\delta t p_n\cdot[p_n]^3 - \delta t\delta t p_n\cdot\nabla f(x_n) \\
&\leq x_n\cdot p_n - \gamma\delta t x_n\cdot p_n + c\delta t\|x_n\cdot[p_n]^3\| - \delta t x_n\cdot\nabla f(x_n) + \\
&\quad \delta t\|p_n\|^2 - \gamma\delta t^2\|p_n\|^2 + c\delta t^2\|p_n\cdot[p_n]^3\| + \delta t^2\|p_n\cdot\nabla f(x_n)\| \\
&\leq x_n\cdot p_n - \gamma\delta t x_n\cdot p_n + c\delta t\|x_n\|\|[p_n]^3\| - \delta t x_n\cdot\nabla f(x_n) + \\
&\quad \delta t\|p_n\|^2 - \gamma\delta t^2\|p_n\|^2 + c\delta t^2\Big(\frac{1}{4\nu}\|p_n\|^2 + \nu\|[p_n]^3\|^2\Big) + \delta t^2\Big(\frac{1}{4\kappa}\|p_n\|^2 + \kappa\|\nabla f(x_n)\|^2\Big) \\
&\leq x_n\cdot p_n + \gamma\delta t\|x_n\|\|p_n\| + cR^2\delta t\|x_n\|\|p_n\| - \delta t x_n\cdot\nabla f(x_n) + \delta t\|p_n\|^2 + \gamma\delta t^2\|p_n\|^2 \\
&\quad + c\delta t^2\Big(\frac{1}{4\nu}\|p_n\|^2 + \nu R^4\|p_n\|^2\Big) + \delta t^2\Big(\frac{1}{4\kappa}\|p_n\|^2 + \kappa\|\nabla f(x_n)\|^2\Big) \Rightarrow \\
x_{n+1}\cdot p_{n+1} &\leq x_n\cdot p_n + (\gamma + cR^2)\delta t\|x_n\|\|p_n\| - \delta t x_n\cdot\nabla f(x_n) + \delta t\|p_n\|^2 \\
&\quad + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big),
\end{aligned}
$$

therefore

$$\begin{aligned}C_{\delta t}x_{n+1}\cdot p_{n+1} \leq & C_{\delta t}x_n\cdot p_n + C_{\delta t}(\gamma + cR^2)\delta t\|x_n\|\|p_n\| - C_{\delta t}\delta t x_n\cdot\nabla f(x_n) + C_{\delta t}\delta t\|p_n\|^2\\ & + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big).\end{aligned}$$

We thus have

$$\begin{aligned}\mathscr{W}(x_{n+1},p_{n+1}) \leq & f(x_n) + \delta t\nabla f(x_n)\cdot p_n + \frac{\delta t^2 M}{2}\|p_n\|^2\\ & + A_{\delta t}\frac{\|p_n\|^2}{2} - A_{\delta t}\gamma\delta t\|p_n\|^2 - A_{\delta t}\delta t\nabla f(x_n)\cdot p_n\\ & + B_{\delta t}\|x_n\|^2 + 2B_{\delta t}\delta t\|x_n\|\|p_n\| + B_{\delta t}\delta t^2\|p_n\|^2\\ & + C_{\delta t}x_n\cdot p_n + C_{\delta t}(\gamma + cR^2)\delta t\|x_n\|\|p_n\| - C_{\delta t}\delta t x_n\cdot\nabla f(x_n) + C_{\delta t}\delta t\|p_n\|^2\\ & + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big)\\ = & \mathscr{W}(x_n,p_n) + (1 - A_{\delta t})\delta t\nabla f(x_n)\cdot p_n + \frac{M}{2}\delta t^2\|p_n\|^2\\ & - A_{\delta t}\gamma\delta t\|p_n\|^2 + 2B_{\delta t}\delta t\|x_n\|\|p_n\| + B_{\delta t}\delta t^2\|p_n\|^2\\ & + C_{\delta t}(\gamma + cR^2)\delta t\|x_n\|\|p_n\| - C_{\delta t}\delta t x_n\cdot\nabla f(x_n) + C_{\delta t}\delta t\|p_n\|^2\\ & + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big) \Rightarrow\\ \mathscr{W}(x_{n+1},p_{n+1}) \leq & \mathscr{W}(x_n,p_n) - C_{\delta t}\delta t x_n\cdot\nabla f(x_n) + (1 - A_{\delta t})\delta t\nabla f(x_n)\cdot p_n\\ & - 2(A_{\delta t}\gamma - C_{\delta t})\delta t\frac{\|p_n\|^2}{2} + \Big[2B_{\delta t} + C_{\delta t}(\gamma + cR^2)\Big]\delta t\|x_n\|\|p_n\|\\ & + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big).\end{aligned}$$

We set $A_{\delta t} = 1$, and demand $C_{\delta t} < \gamma$, upon which we have

$$\begin{aligned}\mathscr{W}(x_{n+1},p_{n+1}) \leq & \mathscr{W}(x_n,p_n) - C_{\delta t}\delta t x_n\cdot\nabla f(x_n) - (\gamma - C_{\delta t})\delta t\|p_n\|^2\\ & + \Big[2B_{\delta t} + C_{\delta t}(\gamma + cR^2)\Big]\delta t\|x_n\|\|p_n\| + C\delta t^2\Big(\|p_n\|^2 + \|\nabla f(x_n)\|^2\Big).\end{aligned}$$

Assuming that $x^* = 0, f(x^*) = 0$ (where $\nabla f(x^*) = 0$) and using the strong convexity and Lipschitz continuity assumption we have $f(x) \geq m\frac{\|x\|^2}{2}$ and $\|\nabla f(x)\|^2 \leq M^2\|x\|^2$. Next, observe that we can derive a lower bound of the Lyapunov function using a discrete Cauchy-Schwarz inequality and this in turn implies that there exists a $C \in \mathbb{R}_+$ such that

$$\mathscr{W}(x_n,p_n) \geq C\Big(f(x_n) + \|p_n\|^2 + \|x_n\|^2\Big).$$

Using this, along with the lower bounds we derived above for $f(x)$ and $\|x\|^2$ implies that there exists a $K \in \mathbb{R}_+$ such that

$$\|p\|^2 + \|\nabla f(x)\|^2 \leq K\mathscr{W}(x,p)$$

for $B_{\delta t} > C_{\delta t}^2$.

This allows us to upper bound $\mathscr{W}(x_{n+1}, p_{n+1})$ as follows

$$\begin{aligned}\mathscr{W}(x_{n+1}, p_{n+1}) \leq & (1 + CK\delta t^2)\mathscr{W}(x_n, p_n) - C_{\delta t}\delta t x_n \cdot \nabla f(x_n) \\ & -(\gamma - C_{\delta t})\delta t \|p_n\|^2 + \left[2B_{\delta t} + C_{\delta t}(\gamma + cR^2)\right]\delta t \|x_n\| \|p_n\|\end{aligned}$$

Finally, using the strong convexity assumption and a Cauchy-Schwarz inequality (where $\lambda > 0$), we obtain

$$\begin{aligned}\mathscr{W}(x_{n+1}, p_{n+1}) \leq & (1 + CK\delta t^2)\mathscr{W}(x_n, p_n) - C_{\delta t}\delta t f(x_n) \\ & - \left(C_{\delta t}\frac{m}{2} - \frac{\lambda^2}{2}\left(2B_{\delta t} + C_{\delta t}(\gamma + cR^2)\right)\right)\delta t \|x_n\|^2 \\ & - \left((\gamma - C_{\delta t}) - \left(2B_{\delta t} + C_{\delta t}(\gamma + cR^2)\right)\frac{1}{2\lambda^2}\right)\delta t \|p_n\|^2 \\ \leq & (1 - \rho\delta t + CK\delta t^2)\mathscr{W}(x_n, p_n),\end{aligned}$$

for some $\rho > 0$. □

Convergence of discretized CD dynamics with stochastic gradients

Assumption 4.2.1. *Let G be a stochastic gradient approximation governed by a random variable ω of a function f, where $\mathbb{E}[G(\cdot, \omega)] = \nabla f(\cdot)$. We assume the variance of the stochastic gradients is uniformly bounded, that is*

$$\exists D > 0 : \forall x \in \mathbb{R}^d, \mathbb{E}\|G(x, \omega) - \nabla f(x)\|^2 \leq D.$$

Proposition 4.2.10. *Assume that $f \in C^2$ is approximated by a stochastic gradient satisfying Assumption 4.2.1 and $\exists m, M \in \mathbb{R}_+ : m \leq \nabla^2 f(x) \leq M \, \forall x \in \mathbb{R}^d$. Then considering a stochastic gradient Euler discretisation of the CD dynamics and any initial condition (x_0, p_0) such that*

$$\|x_0\| + \|p_0\| \leq L,$$

where $L > 0$, there exists $\delta t^* > 0, r > 0$ and $C > 0$:

$$\forall n \geq 0 \text{ and } \forall \delta t \in (0, \delta t^*), \quad \mathbb{E}[f(x_n) - f(x^*)] + \mathbb{E}\|p_n\|^2 \leq Ce^{-\kappa n \delta t}.$$

Proof. We use an Euler discretization, where the update of the momenta is performed based on the stochastic gradient.

$$p_{n+1} = (1 - \gamma\delta t)p_n - c\delta t[p_n]^3 - \delta t G(x_n), \tag{4.34}$$

$$x_{n+1} = x_n + \delta t p_n, \tag{4.35}$$

where we assume $\gamma\delta t$ is sufficiently small. We consider the same Lyapunov function as in the full gradient setting (4.33). Repeating the estimate replacing the full gradient with $\nabla f(x) + G(x) - \nabla f(x)$, noting that $G - \nabla f$ has zero expectation we can prove the required result. □

4.2.3 CADAM

We present a cubically damped version of the continuous dynamics of Adam where we introduce an extra cubic damping term in the momentum equation

$$\dot{x} = \frac{p}{\sqrt{\zeta + \varepsilon}}, \tag{4.36}$$

$$\dot{p} = -\nabla f(x) - \gamma p - c[p]^3, \tag{4.37}$$

$$\dot{\zeta} = [\nabla f(x)]^2 - \alpha\zeta. \tag{4.38}$$

Numerical discretization of CADAM

We choose the following splitting scheme for the numerical implementation of the dynamics (4.36)-(4.38)

$$\begin{pmatrix} \dot{x} \\ \dot{p} \\ \dot{\zeta} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{p}{\sqrt{\zeta+\varepsilon}} \\ 0 \\ 0 \end{pmatrix}}_{\text{A}} + \underbrace{\begin{pmatrix} 0 \\ -\nabla f(x) \\ 0 \end{pmatrix}}_{\text{B}} + \underbrace{\begin{pmatrix} 0 \\ -c[p]^3 \\ 0 \end{pmatrix}}_{\text{C'}} + \underbrace{\begin{pmatrix} 0 \\ -\gamma p \\ 0 \end{pmatrix}}_{\text{D}} \underbrace{\begin{pmatrix} 0 \\ 0 \\ [\nabla f(x)]^2 - \alpha_\zeta \zeta. \end{pmatrix}}_{\text{E}} \tag{4.39}$$

All of the terms in this splitting have been encountered in the previous sections where their discretisation can be found.

4.3 Numerical experiments

Next, we perform a series of numerical studies to compare our proposed algorithms (iKFAD, CD, CADAM) with Adam and mSGD. We compare the optimisers on some standard image classification benchmarks, specifically the Fashion MNIST (FMNIST) [149], CIFAR-10 [63], CIFAR-100 [63] datasets. Finally, we demonstrate the potential of our proposed algorithms on two natural language processing classification tasks. We train a DistilBERT model on the SST-2 and QNLI GLUE [145] benchmarks. We use “Optuna” to tune the hyperparameters of the various optimisers and choose the set of hyperparameters that yields the minimum validation loss or best final validation loss.

To further demonstrate the potential of the adaptive friction and cubic damping mechanisms and minimize hyperparameter tuning and the potential “uncertainty” that it introduces with regard to how thorough and exhaustive the searches are as well as how robust the methods are with respect to the choice of hyperparameters, we also perform some straightforward

comparisons of mSGD with iKFAD and CD on the SST-2 dataset. We choose some reasonable "recommended" values for α and μ for iKFAD and c for CD and then compare the two methods to mSGD for a grid of γ and h values. We demonstrate that we nearly always benefit from the use of the adaptive friction mechanism. We demonstrate that for almost any γ and h combination, adaptive friction and cubic damping improve performance. We summarize the models used and parameter choices made for each benchmark in Table 4.1.

	FMNIST	**CIFAR-10**	**CIFAR-100**	**SST2**	**QNLI**
Network	Resnet18	Resnet34	Resnet18,34,50 |EfficientNet-B5	DistilBERT	DistilBERT
Batch size	64	64	64 | 128	16	16
# Epochs	40	100 | 40	100 | 40	8	15
# *Optuna* trials	200	120	120	100	70-90
Training set size	56000	48000	48000	47754	77144
Test set size	7000	6000	6000	10233	16531
Validation set size	7000	6000	6000	10233	16531
Criterion	**Minimum validation loss / Final validation loss**				

Table 4.1: Experimental Configurations

4.3.1 FMNIST

We first test our optimisers on the Fashion-MNIST dataset. We use a ResNet-18 and a batch size of 64. As can be seen in Figure 4.8, our methods yield a significant performance improvement both in terms of test accuracy and test loss.

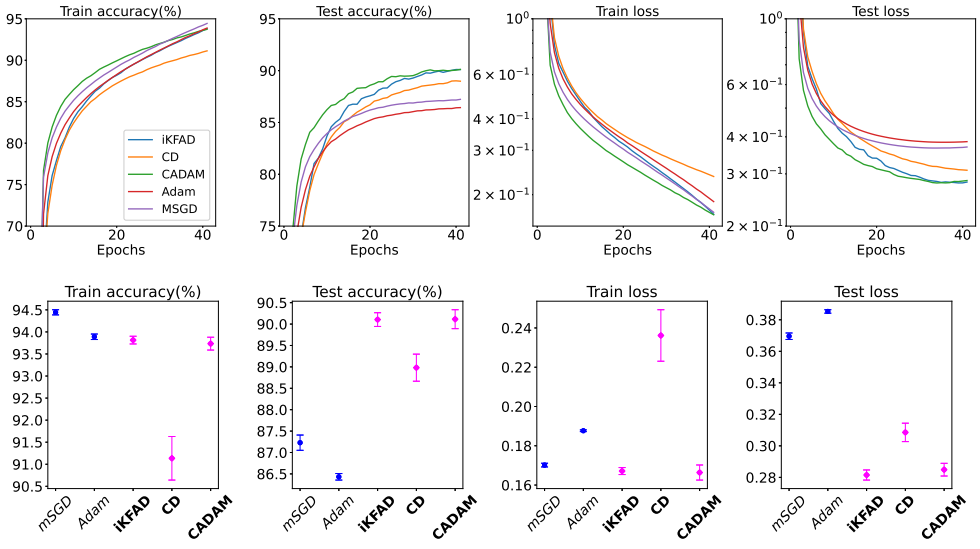


Figure 4.8: Training a ResNet-18 on FMNIST (batch size 64). Training and test loss curves (averaged over ten different random seeds) as well as final train and test losses and accuracies for the optimal hyperparameters. Hyperparameter ranges: $\gamma \in [0.1, 10.0], \alpha \in [0.01, 10.0], \mu = [0.01, 10.0]$. Initialisation of adaptive friction $\xi(0) = 0$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.00035	0.25	4.04	0.0472	-	-	-	-
CD	0.00071	0.25	-	-	1.63	-	-	-
CADAM	2.87e-5	0.13	0.12	-	1.12	-	-	-
mSGD	1.36e-5	-	-	-	-	-	-	0.94
Adam	1.04e-6	-	-	-	-	0.96	0.99	-

Table 4.2: FMNIST - Optimal hyperparameters corresponding to the results in Figure 4.8.

The results of Figure 4.8 show that iKFAD, CD and CADAM significantly outperform mSGD and Adam in terms of test accuracy and loss.

Next, repeat the tuning process for iKFAD, CD and CADAM, allowing for smaller values for γ, μ, α to see how this will affect the results. Specifically, the new ranges are $\gamma \in [0.0001, 10.0]$, $\mu \in [0.00001, 10.0]$ and $\alpha \in [0.01, 10.0]$. We also choose a non-zero initialisation for the adaptive friction vector ξ, specifically $\xi(0) = 1$.

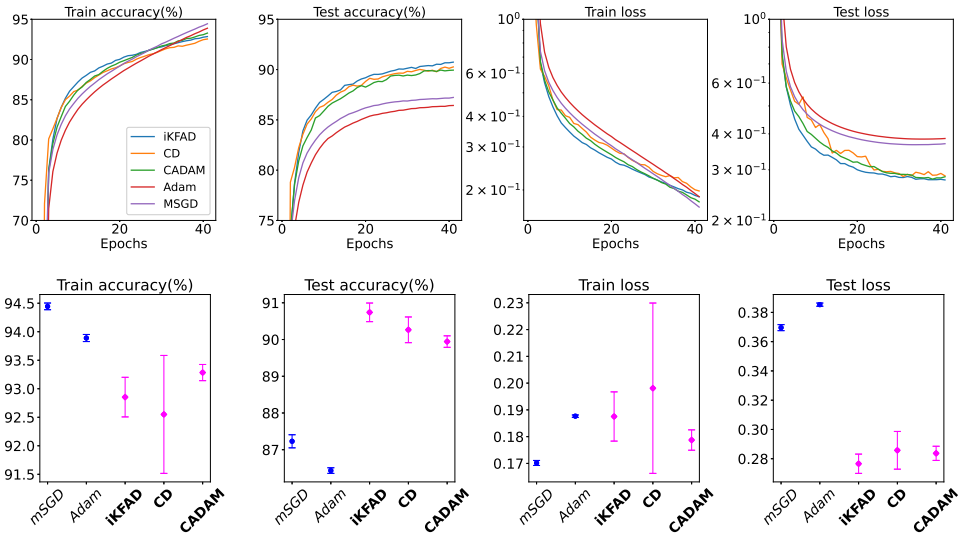


Figure 4.9: Repeating the experiments in Figure 4.8 with $\xi(0) = 1$ and increased ranges for γ, μ, α (for iKFAD, CD, CADAM, with $\gamma \in [0.0001, 10.0]$, $\mu \in [0.00001, 10.0]$ and $\alpha \in [0.01, 10.0]$. We observe that iKFAD, CD, CADAM consistently outperforming mSGD and Adam in terms of test loss and accuracy.

iKFAD, CD and CADAM, continue to outperform mSGD and Adam for the new hyperparameter ranges.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.00089	0.0005	4.37	1.08	-	-	-	-
CD	0.01804	0.0243	-	-	0.44	-	-	-
CADAM	2.70e-5	0.0308	0.31	-	11609	-	-	-
mSGD	1.36e-5	-	-	-	-	-	-	0.94
Adam	1.04e-6	-	-	-	-	0.96	0.99	-

Table 4.3: FMNIST optimal hyperparameters corresponding to the results of Figure 4.9.

Finally, we examine what happens if we set the linear friction to $\gamma = 0$ for iKFAD, CD and CADAM. We also set $\xi(0) = 1$ and set the ranges for μ and α to $\mu \in [0.00001, 1.0]$ and $\alpha \in [0.01, 1.0]$ respectively.

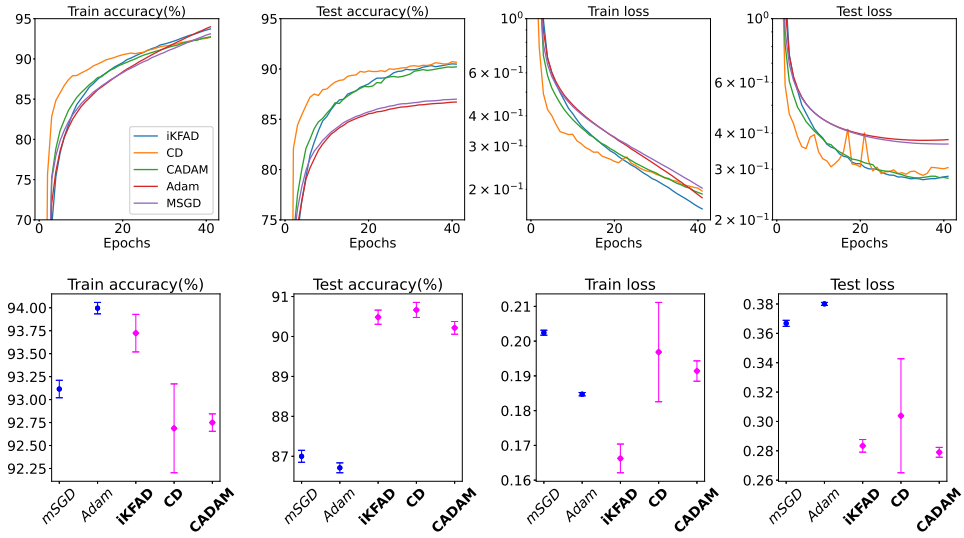


Figure 4.10: Repeating the hyperparameter tuning for FMNIST with $\gamma = 0$ for iKFAD, CD, CADAM. We see that iKFAD, CD, CADAM consistently outperform mSGD and Adam in terms of test loss and accuracy, despite the fact that the linear friction has been set to zero.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	4.50e-4	0	0.87	0.00037	-	-	-	-
CD	0.052	0	-	-	115.42	-	-	-
CADAM	2.30e-05	0	6.68	-	38550.76	-	-	-
mSGD	2.60e-05	-	-	-	-	-	-	0.857
Adam	1.06e-06	-	-	-	-	0.999	0.988	-

Table 4.4: FMNIST optimal hyperparameters corresponding to the results of Figure 4.10.

4.3.2 CIFAR-10

Next, we test our optimisers on the CIFAR-10 dataset. We use a standard ResNet-34 architecture from PyTorch.

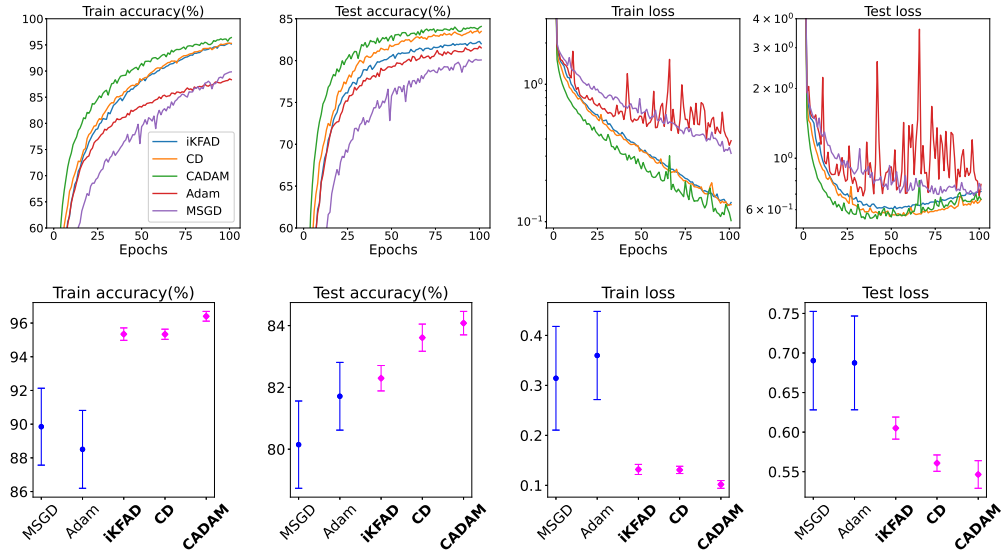


Figure 4.11: Training a ResNet-34 on CIFAR-10 data (batch size 64). The plots show the training and test loss curves (averaged over fifteen different random seeds) and final values for the optimal hyperparameters. Initialisation of adaptive friction $\xi(0) = 0$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.048	0.5979	0.026	8.601	-	-	-	-
CD	0.062	0.27	-	-	7851.89	-	-	-
CADAM	0.013	7.73	0.119	-	0.70	-	-	-
mSGD	0.042	-	-	-	-	-	-	0.96
Adam	0.030	-	-	-	-	0.95	0.93	-

Table 4.5: CIFAR-10 optimal hyperparameters corresponding to the results in Figure 4.11.

Next, we repeat the tuning on CIFAR-10 the standard pytorch ResNet-34 setting the linear friction to zero ($\gamma = 0$). We set $\xi(0) = 0$, $\mu \in [0.00001, 1.0]$ and $\alpha \in [0.01, 1.0]$. The results can be found in Figure A.1 in Appendix A.

The results presented in Figure 4.11 show that our methods outperform mSGD and Adam, but the accuracies we are getting during training, although high, could be improved to get even closer to state of the art results. To achieve higher accuracies, we use a ResNet-34 architecture specifically designed for the CIFAR-10 dataset. We use the ResNet-34 designed by [79], upon which we see an improvement in accuracy. However, with the new architecture, we do not see as dramatic a difference in performance between our methods and mSGD and Adam (see Figure A.2 in Appendix A).

We also train a custom ResNet-18 [79] (with batch size=64, $\gamma \in [1e-9, 10]$, $\mu \in [1e-9, 10]$, $c \in [0.01, 1e10]$ and $\alpha \in [0.001, 10.0]$.)

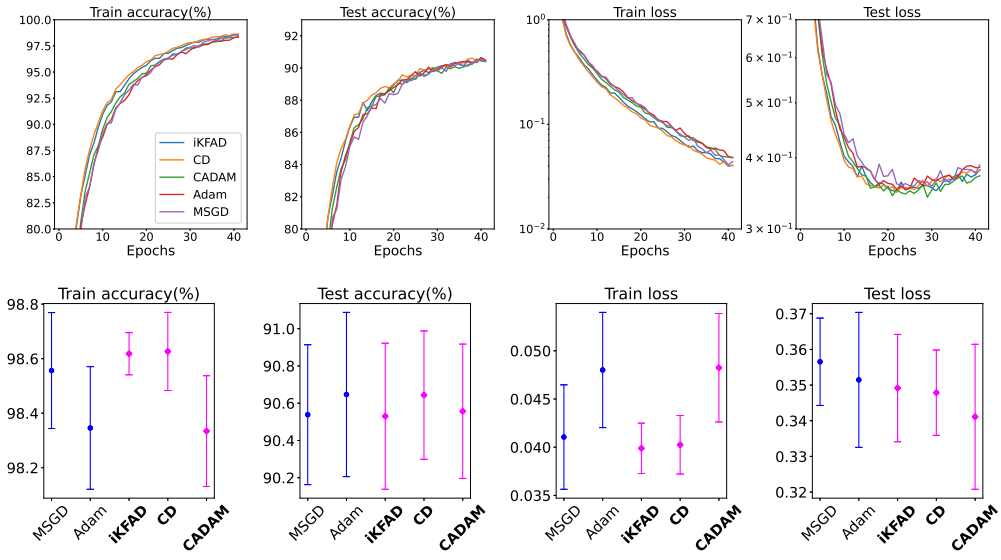


Figure 4.12: Training a ResNet-18 on CIFAR-10 data (batch size 64). The plots show the training and test loss curves (averaged over three different random seeds) and best loss and accuracy values for the optimal hyperparameters. Parameter ranges $\gamma \in [1e-9, 10]$, $\mu \in [1e-9, 10]$, $c \in [0.01, 1e10]$ and $\alpha \in [0.001, 10.0]$

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0973	0.40	0.402	1.87	-	-	-	-
CD	0.0448	0.17	-	-	56814.47	-	-	-
CADAM	0.0037	3.55	0.005	-	82552.90	-	-	-
mSGD	0.0260	-	-	-	-	-	-	0.86
Adam	0.0008	-	-	-	-	0.97	0.99	-

Table 4.6: CIFAR-10 optimal hyperparameters for the results presented in Figure 4.12.

We conclude that for the CIFAR-10 dataset and the ResNet-34 architecture from [79], our methods yield comparable accuracies to mSGD and Adam. Repeating our experiments with a ResNet-18 network (4.12), also leads to similar accuracies for all the methods. It is interesting to note however, that for a standard ResNet-34 (Figure 4.11), not specifically tailored to the CIFAR-10 data, we see a dramatic difference between our methods and mSGD and Adam.

4.3.3 CIFAR-100

We now try our optimisers on the CIFAR-100 dataset. We again start with a standard PyTorch ResNet-34, the same model we used for the CIFAR-10 results in Figure 4.11. We set $\xi(0) = 0$ and choose the following ranges for the hyperparameters of our methods: $\gamma \in [0.1, 10.0]$, $\mu \in [0.01, 10.0]$ and $\alpha \in [0.01, 10.0]$.

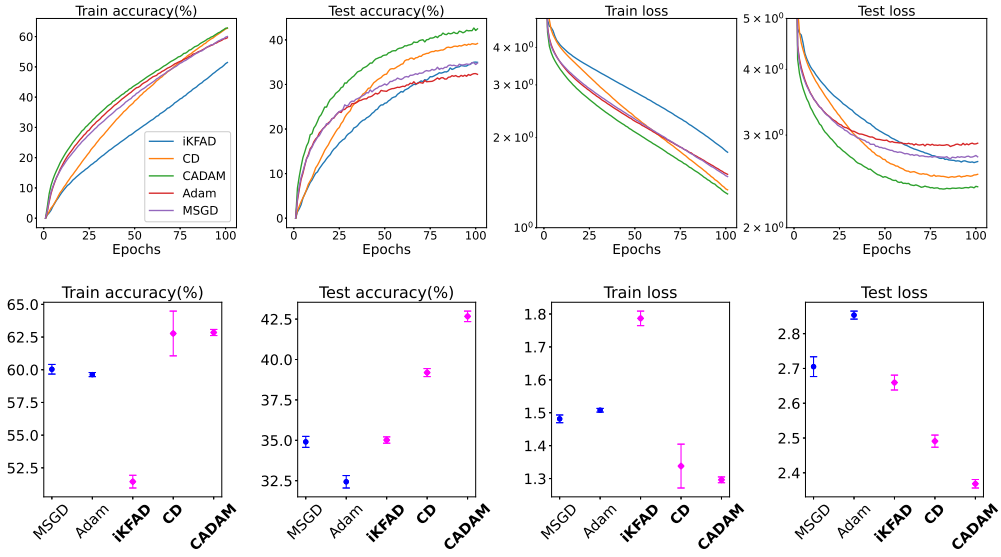


Figure 4.13: Training a ResNet-34 on CIFAR-100 data (batch size 64). The plots show the training and test loss curves (averaged over ten different random seeds) and best accuracies and losses for the optimal hyperparameters. Initialisation of adaptive friction $\xi(0) = 0$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0006	0.10	0.43	7.04	-	-	-	-
CD	0.0012	0.38	-	-	9442.09	-	-	-
CADAM	0.0002	0.16	0.02	-	1122.69	-	-	-
mSGD	0.0001	-	-	-	-	-	-	0.89
Adam	9.53e-6	-	-	-	-	0.85	0.98	-

Table 4.7: CIFAR-100 optimal hyperparameters for the results presented in Figure 4.13.

As was the case when we trained the same network on the CIFAR-10 dataset (4.11), our methods (especially CD and CADAM) outperform mSGD and Adam in terms of both test accuracy and test loss. The test accuracies however, are very low, which prompts us to try different network architectures.

In an attempt to get higher accuracies, we use a bigger residual network. We repeat our experiments using a standard PyTorch ResNet-50 architecture (batch size of 128, $\xi(0) = 0$, $\gamma \in [0.01, 10.0]$, $\mu \in [0.0001, 10.0]$ and $\alpha \in [0.01, 10.0]$). We also test the algorithms on an EfficientNet-B5 (batch size=256, $\mu \in [0.00001, 1.0]$, $\alpha \in [0.01, 1.0]$, $\gamma = 0$, $c \in [0.01, 10^10]$). These results can be found in Appendix A.

Finally, we train a ResNet-18 on CIFAR-100. We use the following parameter ranges for the hyperparameter search: $\gamma \in [1e-9, 10]$, $\mu \in [1e-9, 10]$, $c \in [0.01, 1e10]$ and $\alpha \in [0.001, 10.0]$. The results can be seen in Figure 4.14.

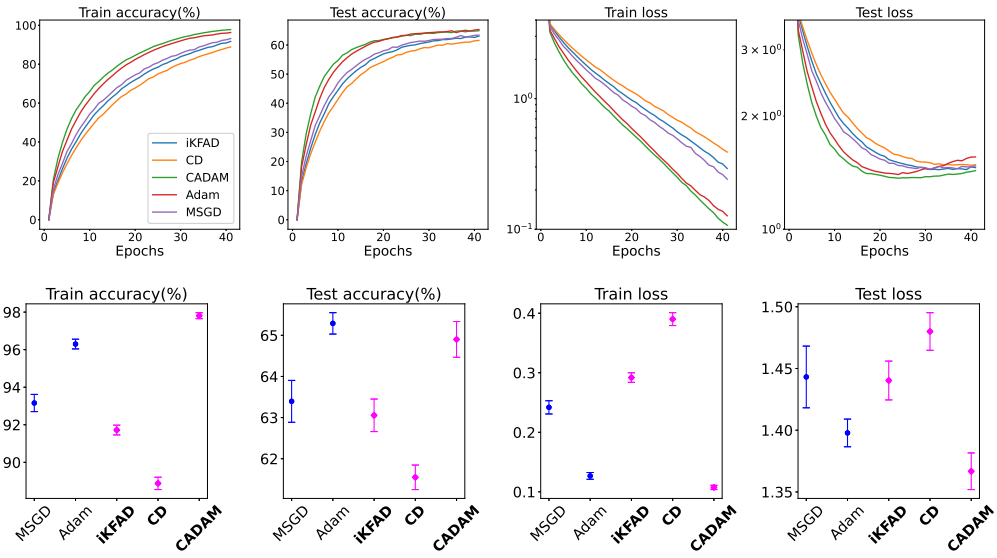


Figure 4.14: Training a ResNet-18 [79] on CIFAR-100 data (batch size 64). The plots show the training and test loss curves (averaged over ten different random seeds) and final loss and accuracy values for the optimal hyperparameters. Parameter ranges $\gamma \in [1e-9, 10]$, $\mu \in [1e-9, 10]$, $c \in [0.01, 1e10]$ and $\alpha \in [0.001, 10.0]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0086	1.40	1.6851	1.4746e-05	-	-	-	-
CD	0.0112	2.34	-	-	94980.09	-	-	-
CADAM	0.0018	9.19	0.003	-	3788057.06	-	-	-
mSGD	0.0004	-	-	-	-	-	-	0.946
Adam	0.0001	-	-	-	-	0.996	0.985	-

Table 4.8: CIFAR-100 optimal hyperparameters for results in Figure 4.14.

4.3.4 Training DistilBERT on QNLI dataset

In this section, we test our optimisers on a DistilBERT model that we train on the QNLI dataset. We choose a pre-trained DistilBERT and finetune it on QNLI data as training the model from scratch requires a considerably longer number of epochs/time for this task.

Unlike what we did for FMNIST, CIFAR-10 and CIFAR-100, where we used the final validation loss (or mean validation loss over the last N epochs) as the criterion for choosing the optimal hyperparameters, here, we use the minimum validation loss as the criterion and allow. The hyperparameter ranges used in the search are the following: $\gamma \in [1e-9, 10.0], c \in [0.01, 10e13], \alpha \in [0.001, 10.0], \mu \in [1e-13, 10.0]$. The results of this experiment can be seen in Figure 4.15.

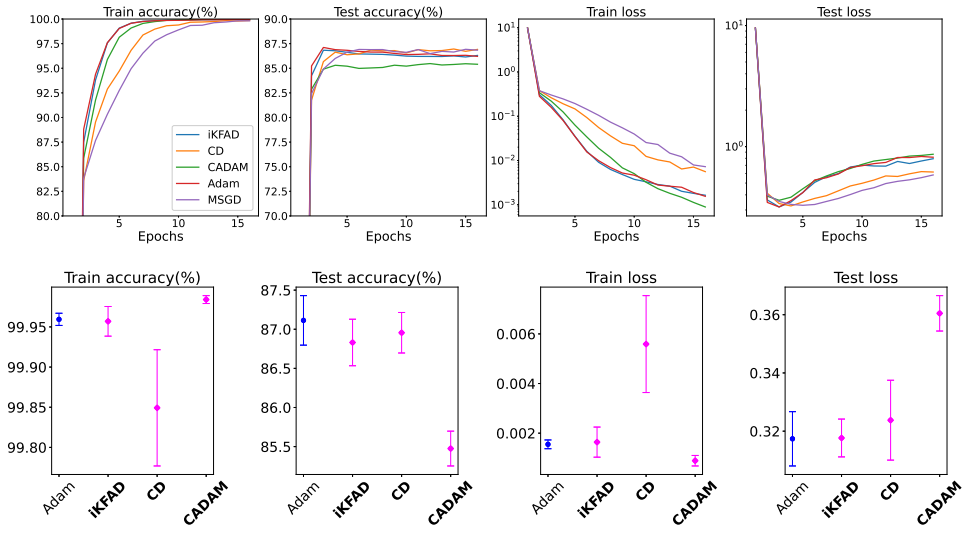


Figure 4.15: Training DistilBERT (from pretrained) on QNLI data (batch size 16). Initialisation of adaptive friction $\xi(0) = 0$. The parameter ranges used are $\gamma \in [1e-9, 10.0], c \in [0.01, 10e13], \alpha \in [0.001, 10.0], \mu \in [1e-13, 10.0]$. The criterion used for hyperparameter selection is the minimum validation loss over all epochs.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0413	1.36e-08	2.80	6.71	-	-	-	-
CD	0.0995	4.69	-	-	3802216.14	-	-	-
CADAM	0.0002	9.66	0.0026	-	22754.57	-	-	-
mSGD	0.0015	-	-	-	-	-	-	0.895
Adam	3.52e-05	-	-	-	-	0.855	0.933	-

Table 4.9: QNLI optimal hyperparameters corresponding to the results in Figure 4.15

4.3.5 Training DistilBERT on SST-2 dataset

In this section we train a DistilBERT model from scratch, which is a distilled version of BERT [21], on the SST-2 [123] dataset from the GLUE benchmark. The dataset consists of sentences labelled as either zero or one, depending on whether the sentiment they express is negative or positive.

We use the minimum validation loss as a metric to determine the best set of hyperparameters. A batch size of 16 is used for all experiments. As done previously, after tuning the hyperparameters for each method, we use the "optimal" hyperparameters for each optimiser and rerun the optimisation process using ten different random seeds and computing the mean and variance of the loss and accuracy for each optimiser at each epoch.

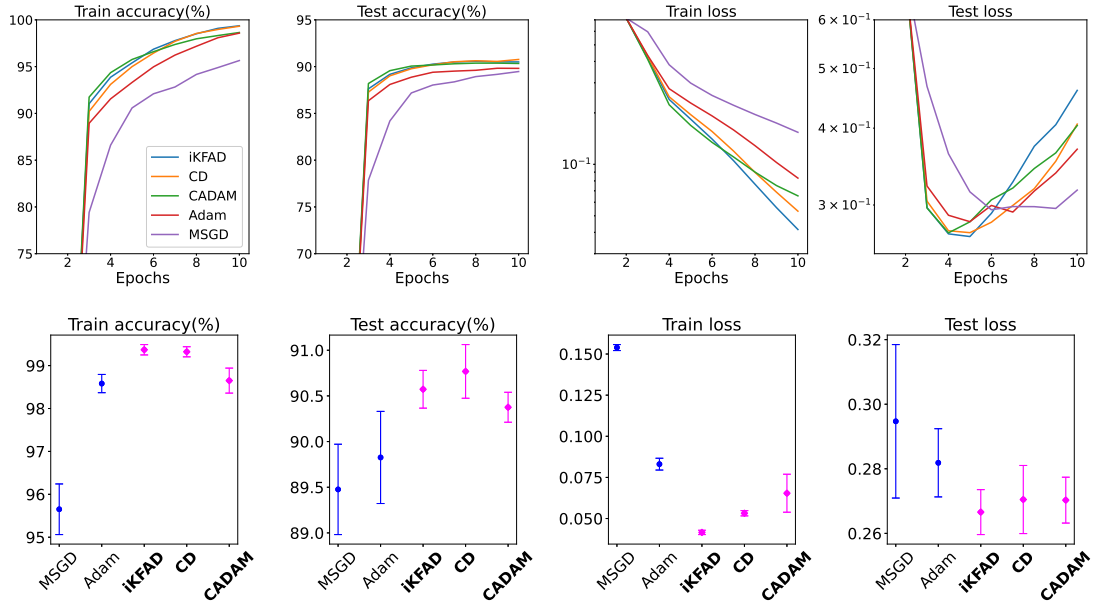


Figure 4.16: Training DistilBERT on SST2 data (batch size 16). Setting $\gamma \in [0.000001, 1]$ for CD, CADAM, iKFAD and allowing for lower values of $\mu \in [1e-9, 1.0]$ and $\alpha \in [0.001, 1.0]$. For the cubic damping coefficient for CD and CADAM we have $c \in [0.01, 10e9]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0270	1.28e-06	0.047	3.61e-07	-	-	-	-
CD	0.0422	0.052	-	-	33438053.5	-	-	-
CADAM	0.0003	9.22	0.032	-	0.024	-	-	-
mSGD	0.0024	-	-	-	-	-	-	0.83
Adam	3.82e-05	-	-	-	-	0.92	0.98	-

Table 4.10: Optimal hyperparameters corresponding to Figure 4.16

We note that simply enhancing the mSGD dynamics by a cubic damping term or introducing an adaptive friction in the case of iKFAD allows us to surpass Adam performance in terms of both training and test accuracy for this task.

We also repeated the above experiments with $\gamma = 0$ for iKFAD, CD and CADAM and while iKFAD and CADAM performance declined, CD still performs considerably better for mSGD and slightly better than Adam, as can be seen in Figure 4.17.

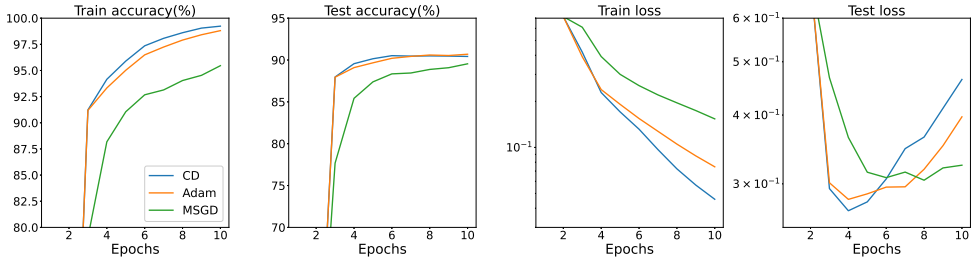


Figure 4.17: Training DistilBERT on SST2 data (batch size 16). Setting $\gamma = 0$ for CD.

	h	γ	α	μ	c	β_1	β_2	ρ
CD	0.0363	0.0	-	-	9780080.16	-	-	-
mSGD	0.0111	-	-	-	-	-	-	0.83
Adam	3.11e-05	-	-	-	-	0.94	0.85	-

Table 4.11: Optimal hyperparameters corresponding to Figure 4.17.

Finally, we perform a different form of comparison between iKFAD, CD and mSGD (LDHD). Rather that rely on *Optuna* to find the optimal set of hyperparameters for each method, we perform a more direct comparison. We create a grid of $\gamma - h$ pairs. Figure 4.18 shows how

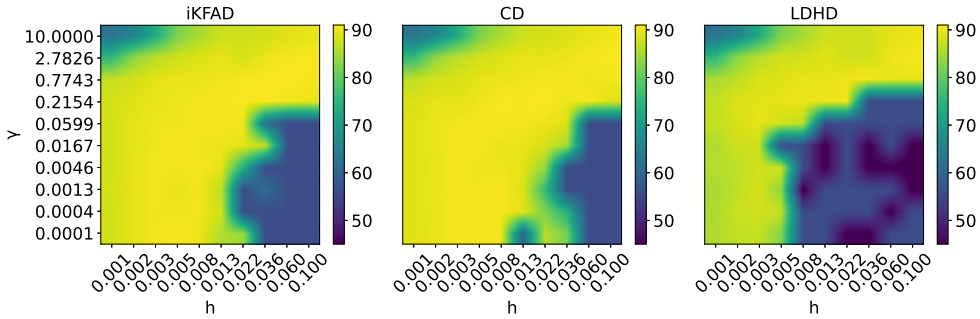


Figure 4.18: Training a DistilBERT model from scratch on the SST-2 dataset. Test accuracy at the final epoch for iKFAD, CD and LDHD (equivalent to mSGD). Each plot is coloured according to the final test accuracy after eight epochs of training. We see that iKFAD and CD are more robust in terms of the choice of step size and also yield higher accuracies.

the choice of the linear friction γ and step size h affect the performance in terms of final test

accuracy of iKFAD, CD and LDHD. We see that iKFAD and CD allow for the use of higher step sizes and also lead to greater (or in the worst case same) accuracies as LDHD as can be seen more clearly from Figure 4.19. Note also that we did not perform any tuning on the hyperparameters α and μ for iKFAD and c for CD to produce these plots. We set $\alpha = 0.1$, $\mu = 10^{-4}$ for iKFAD and $c = 10^5$ for CD. If we had also tuned α, μ and c, the performance improvement that our methods yield over LDHD would have been greater.

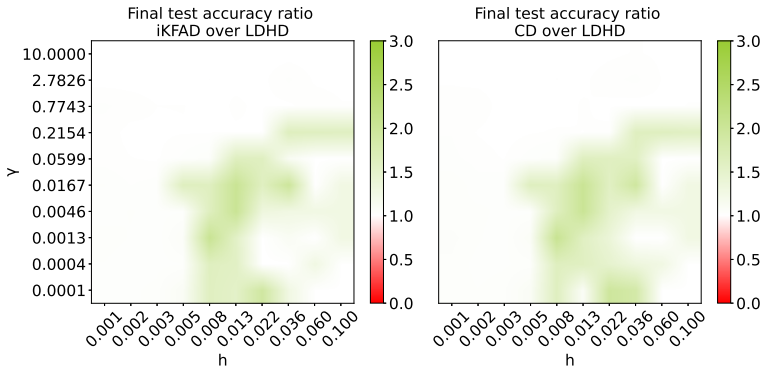


Figure 4.19: Test accuracy ratio at the final epoch for iKFAD over LDHD and CD over LDHD. We can see that there is a large region where we can get up to three times higher accuracy using iKFAD or CD over LDHD.

For visualisation purposes, we crop the colorbar to have a maximum of 1.5 and we replot the results in Figure 4.19. The modified plot is presented in Figure 4.20. We can see that

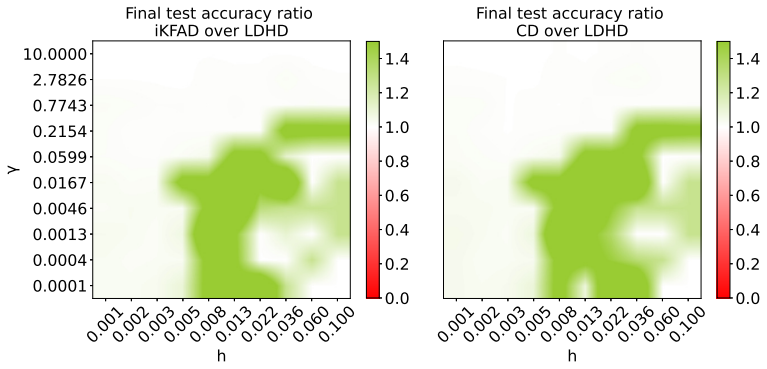


Figure 4.20: Replotting the results in 4.19 after cropping the maximum value of the colorbar down to 1.5 for better visualisation.

iKFAD and CD outperform LDHD for any combination of γ and h that was tested and that there is a region in the γ-h space where iKFAD and CD, even without proper tuning of their hyperparameters can lead to up to three times higher test accuracy than LDHD.

Chapter 5

Preconditioned Learning Rates for Continual Learning

5.1 Introduction

The ability of a model to learn in an incremental fashion is key for many real-world applications. This manner of learning is inherent in most biological cognitive systems like the human brain, where new knowledge can be obtained with exposure to new data, whilst previously acquired knowledge is retained to a large extent or only gradually forgotten [33]. Unlike classical neural network models, biological systems are not subject to sudden disruptions of previously acquired knowledge termed as *catastrophic forgetting* [85]. In this work we examine how the use of preconditioned learning rates (PLR) can enable learning in a continual manner.

Continual learning refers to a setting where a model is updated as new data becomes available, while preserving knowledge acquired from previous tasks. In a supervised learning setting this can be expressed as follows. Let us consider a sequence of datasets

$$\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}, \quad \text{where} \quad \mathcal{D}_n = \{(x_i^{(n)}, y_i^{(n)})\},$$

on which we wish to train a model. Here $x_i^{(n)}$ denotes the features and $y_i^{(n)}$ the corresponding labels of dataset $\mathcal{D}^{(n)}$. In a non-incremental scenario, the model would be trained once over the whole dataset $\mathcal{D}$, which would be available from the outset. In an incremental learning setting however, the system accepts new data over a span of time [116], while old data often needs to be partially or completely erased. The challenge one faces in such a scenario is to maintain old knowledge, even if old data is no longer available, or retraining on it is not computationally viable.

Several real world applications necessitate this type of learning, such as dynamic control systems, e.g. planning an optimised future trajectory for an autonomous system, automated disease diagnosis systems trained and refined on a series of classified data which cannot be stored for privacy reasons, or ad serving systems that have to deal with huge streams of data, which would be impossible to store [27]. As well as simulating the learning process of biological cognitive systems and being necessary in cases where old data is no longer

available, models that are able to learn in a continual manner are also more sustainable in the sense that they reduce the computational time needed to retrain the model to adjust it to new data [16], therefore decreasing the impact that the training of large models has on the environment. It should be noted here that forgetting is not something completely undesirable. It is important to also be able to forget information that is no longer relevant.

Continual learning refers to a setting where a model is updated as new data becomes available, while preserving knowledge acquired from previous tasks. In a supervised learning setting this can be expressed as follows. Let us consider a sequence of datasets

$$\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}, \quad \text{where} \quad \mathcal{D}_n = \{(x_i^{(n)}, y_i^{(n)})\},$$

on which we wish to train a model. Here $x_i^{(n)}$ denotes the features and $y_i^{(n)}$ the corresponding labels of dataset $\mathcal{D}^{(n)}$. In a non-incremental scenario, the model would be trained once over the whole dataset $\mathcal{D}$, which would be available from the outset. In an incremental learning setting however, the system accepts new data over a span of time [116], while old data often needs to be partially or completely erased. The challenge one faces in such a scenario is to maintain old knowledge, even if old data is no longer available, or retraining on it is not computationally viable.

Several real world applications necessitate this type of learning, such as dynamic control systems, e.g. planning an optimised future trajectory for an autonomous system, automated disease diagnosis systems trained and refined on a series of classified data which cannot be stored for privacy reasons, or ad serving systems that have to deal with huge streams of data, which would be impossible to store [27]. As well as simulating the learning process of biological cognitive systems and being necessary in cases where old data is no longer available, models that are able to learn in a continual manner are also more sustainable in the sense that they reduce the computational time needed to retrain the model to adjust it to new data [16], therefore decreasing the impact that the training of large models has on the environment. It should be noted here that forgetting is not something completely undesirable. It is important to also be able to forget information that is no longer relevant.

5.2 Natural gradient descent and adaptive gradient methods

In this section we examine the concepts of natural gradient descent [4] and adaptive gradient algorithms such as Adam, which are in a way related to our proposed continual learning algorithm.

Let us assume we are optimising a function $F(x;\theta)$ over the values of the parameters vector $\theta \in \mathbb{R}^d$ of a neural network, where N is the number of parameters (weights and biases) of the network and $x \in \mathbb{R}^M$ is the input to the network. The loss landscape of multi-layer neural networks has many local minima. As is shown in [15], finding global minima becomes harder as the size of a network increases, but we also often wish to avoid finding global minima as this can lead to overfitting. [15] also show that for large networks critical points exhibit a layered structure where high-quality local minima can be found close to the global minimum.

This means that, rather than having a unique solution/optimal parameter vector θ^*, there is a distribution over parameter values that we want to explore. From a Bayesian perspective, optimising a model's parameters is equivalent to finding their most likely values given a dataset [62].

When there is an underlying structure to a parameter space, the steepest descent direction is given not by the ordinary gradient, but by the natural gradient [4]. The way popular optimisation algorithms like Gradient Descent (GD) work, is by computing the gradient of the objective function and then taking a small step, regulated by the step size η in the direction of the gradient. This will in turn lead to a small change in each parameter. A small change in a single parameter however can potentially correspond to a large shift in distributional space. To illustrate this, consider a one-dimensional Gaussian distribution, $\mathcal{N}(\mu, \sigma^2)$, where σ is low. We first apply a small shift $d\mu$ on the mean μ of the distribution, which results in the new distribution $\mathcal{N}(\mu + d\mu, \sigma_1^2)$. We applied a small shift in parameter space, but how big is the shift in distributional space? How different is the resulting distribution to the original one? The answer depends on the variance σ. Since sigma is low, even a small shift in the mean can lead to a distribution with a very small overlap with the original distribution. This is illustrated in fig.5.1

Natural gradient descent (NGD) has been shown to have a higher convergence rate than gradient descent in the case where the loss function can be approximated by a convex quadratic [4]. In the neural network setting, exact natural gradient descent can be quite expensive, both in terms of computational resources and memory [154], as it involves computation and storage of the inverse of the Fisher information matrix [4]. The ADAM optimiser, offers an alternative, cheaper way of taking second-order information into account, by computing a running average of the Fisher information matrix [61].

This demonstrates that when trying to learn a set of parameters which has an underlying distribution, as is the case in neural network fitting, we want to ensure we take appropriately small steps in distributional space and that cannot be ensured by taking small steps in parameter space.

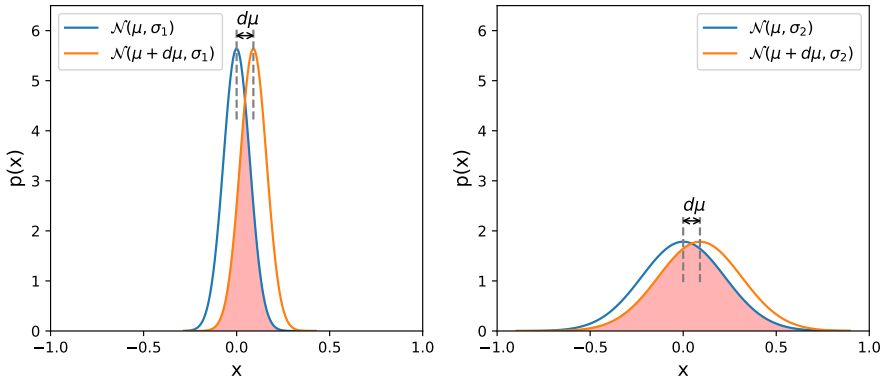


Figure 5.1: Low variance Gaussian distribution $\mathcal{N}(\mu, \sigma_1^2)$ (left graph) and high variance Gaussian $\mathcal{N}(\mu, \sigma_2^2)$ (right graph). We shift the mean of both distributions by $d\mu$. The shifted distribution $\mathcal{N}(\mu + d\mu, \sigma_1^2)$ has a much smaller overlap with $\mathcal{N}(\mu, \sigma_1^2)$, compared to the overlap between $\mathcal{N}(\mu, \sigma_2^2)$ and $\mathcal{N}(\mu + d\mu, \sigma_2^2)$. This demonstrates that a small shift in parameter space is not necessarily tantamount to a small distributional shift.

5.3 Continual learning

Continual learning approaches can be divided into three main groups [49].

Replay methods: These methods either store sampled data from previous tasks for the model to rehearse while being trained on new tasks [20], or use generative models to synthesize data similar to the data encountered in previous tasks [120], [28].

Regularisation methods: These methods identify model parameters that are important for maintaining performance on previous tasks [62], [152], [3] constraining their change, or they constrain the model output [77]. In this last approach, the model consists of some shared parameters and some task specific parameters and the output of the old tasks on new data is constrained so that it does not change too much during training, which consequently leads to preservation of performance on old data.

Architectural methods: These methods utilize dynamic architectures, adding for instance new neurons or layers to the model in response to new data. [111] propose such an architectural method where a new neural network is added for each new task. The disadvantage of this method and other architecture-based methods is that while they preserve performance on old tasks the complexity of the architecture grows with the number of tasks [101].

Our preconditioned learning rates method does not fall under any of the above categories but draws inspiration from a popular regularisation method called “elastic weight consolidation”, which uses the Fisher information matrix to quantify parameter importance. The main ideas behind the elastic weight consolidation method are presented in Subsection 5.3.1.

5.3.1 Elastic Weight Consolidation

Elastic weight consolidation (EWC) [62] is an example of a weight regularisation method. EWC uses the Fisher information matrix (FIM) to quantify the importance of each of the model parameters and regularise them accordingly. The Fisher information matrix F is defined as the variance of the score function $s(\theta) = \nabla_\theta \log p(x|\theta)$

$$F = \mathbb{E}_{p(x|\theta)}\left[\left(s(\theta) - \mathbb{E}_{p(x|\theta)}s(\theta)\right)\left(s(\theta) - \mathbb{E}_{p(x|\theta)}s(\theta)\right)^T\right]. \tag{5.1}$$

and is a way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ of a distribution that models X.

Since $\mathbb{E}_{p(x|\theta)}s(\theta) = 0$ at the true parameter value θ, equation 5.1 can be simplified into the following form

$$F = \mathbb{E}_{p(x|\theta)}\left[\nabla_\theta \log p(x|\theta)\nabla_\theta \log p(x|\theta)^T\right].$$

We cannot compute the true FIM from a finite dataset, however, we can estimate the empirical FIM as

$$F = \frac{1}{N}\sum_{i=1}^{N}\left[\nabla_\theta \log p(x_i|\theta)\nabla_\theta \log p(x_i|\theta)^T\right]. \tag{5.2}$$

Let us assume that we have trained a model on a task A and wish to train the model on a new task B, while maintaining performance on task A. [62] augment the loss function for task B by a quadratic penalty, which constrains the change of model parameters according to their importance (FIM diagonal value). The loss when training on task B would therefore be

$$L(\theta) = L_B(\theta) + \sum_j \frac{\lambda}{2}F_{jj}(\theta_j - \theta^*_{A,j})^2, \tag{5.3}$$

where the first term is a loss function such as Mean Squared Error and the second term is the regularisation term that constrains the values of the new weights to be close to those of the old ones. Each weight is constrained to a different extent, depending on the value of the corresponding diagonal element of the FIM, F_{jj}. This quadratic regularisation constraint can therefore be thought of as a spring, keeping the parameters close to optimal values for previous tasks, which is why the term “elastic” is used [62].

For a justification of the use of a quadratic penalty from a Bayesian perspective we refer the reader to [50], [2].

In the following section, we will describe how the combined use of the Fisher Information Matrix with multirate training methods can lead to an efficient online learning methodology.

5.4 Evaluation of continual learning benchmarks

In order to test the performance of our method, we need to choose some benchmarks commonly used for continual learning applications. [27] propose the following five fundamental desiderata for tasks used in the evaluation of continuous learning algorithms

(A) Tasks must resemble each other to some extent to be representative of real world continuous learning scenarios. Benchmarks like the permuted MNIST violate this requirement by having tasks that are too dissimilar to each other, thus making distinction amongst them easier.

(B) All tasks should share the same output nodes.

(C) No test-time assumed labels. For example in handwritten digits classification, knowing during testing that you are classifying between e.g. 2s and 3s is cheating. To test the continual learning capacity of the model, it should be left to distinguish amongst all ten digits.

(D) In a truly continual learning setting, there should be no need to retrain on old tasks.

(E) The benchmark should involve more than two tasks. Only demonstrating that a continual learning method works on a set of two tasks is not enough to prove the efficiency of the method.

The authors of [138] introduce a categorization of learning scenarios, based on whether task identity is provided during test time and if not, whether it must be inferred. Each of the standard continual learning benchmark tasks, such as Permuted or Split MNIST can be performed according to the three scenarios proposed by [137]. The three scenarios in increasing difficulty are the following. **Task Incremental Learning**: In this scenario, task identity is always provided. A common network architecture for this scenario would have a different head (set of output units) for each task, while the rest of the network would be shared for all tasks. **Domain Incremental Learning** In this scenario, task identity is not provided during testing, since the task structure typically remains the same throughout training (e.g the task is for an autonomous system to navigate through various terrain types). What changes is the distribution of the input for each task. Permuted MNIST was designed as a benchmark for this type of problem. The task structure is always the same (distinguish amongst digits 0 to 9), but the distribution of the training and test data changes from task to task (different input pixels permutation for each task). The problem with permuted MNIST that has been pointed out by [27], is that because we are applying random permutations to the pixels, each task will be very different to previously encountered ones. This makes it easier for the network to remember old tasks, since there is not much overlap amongst them. In real world scenarios like terrain navigation, old tasks are 'confusingly similar' [27] to new ones. **Class Incremental Learning** In this third case, the model is required to incrementally learn new classes. The single-headed split MNIST is an example of such a continual learning scenario.

5.5 Multirate Methods

We examine the potential of employing multirate methods in an online learning context. Multirate methods have been extensively used to solve systems of differential equations characterized by fast and slow time scales. The presence of fast and slow dynamics in brain synapses is the motivation behind the use of multirate methods in neural network training [29], [5], and more recently [141]. [141] clearly demonstrated the potential of multirate methods in capturing different features present in the data as well as in transfer learning applications and as a regularisation approach which can boost generalization performance similarly to dropout techniques [125].

The idea behind multirate training of neural networks as presented in [141] consists in splitting the neural network parameters $\theta \in \mathbb{R}^d$ in two (or more groups) and updating them using different learning rates. Let us assume there are only two time scales (fast and slow) i.e. $\theta = (\theta_F, \theta_S)$. One set of parameters, the "fast" θ_F is updated every step, with step size h_F, whereas the "slow" parameters are updated every k steps with step size $h_S = k \cdot h_S$.

[141] first demonstrate the applicability of multirate methods in capturing different features present in the data. To that end, they train a neural network using fixed learning rates on a dataset consisting of a mix of patch-free CIFAR-10 images, patch-augmented CIFAR-10 images (a noisy patch is added to the center of each image) and patch-only images. They then test the performance of the network on three different validation sets (one consists of patch-free CIFAR-10 images, the other of patch-augmented CIFAR-10 images and the third of patch-only images) and show that when the model is trained using a multirate approach, it is able to memorize the patches while also obtaining high accuracy on clean data.

Common methods for transfer learning include using the pretrained weights as an initialization and retraining / fine-tuning the whole network, or only retraining the last N layers of the network while keeping the rest of the weights fixed. Only retraining the last layers makes sense as the last layers tend to capture more task-specific features [150], whereas early layers in the network tend to capture more general features, which reduces the need for retraining those layers in transfer learning applications.

Instead of retraining all layers or retraining some of the last layers while keeping the rest of the model parameters fixed, [141] suggest a multirate approach where the last layer parameters are treated as the fast part of the network. The gradients for those parameters are computed at every step and used to update the fast parameter values with a step size h_F, while the gradients for the rest of the network parameters (slow part) are computed every k steps. The slow parameters are thus updated every k steps with step size $h_S = k \cdot h_F$. This can lead to a model that is highly accurate with very fast training times.

Also demonstrated in [141] is the potential multirate techniques have in acting as regularisers, improving neural network generalisation, similarly to dropout methods [125]. More specifically, a set of weights is randomly selected by the optimiser to form the slow parameters. This set is deactivated (parameter values set to zero) for k steps, after which the slow parameters are re-activated and updated with a larger time step $h_S = k \cdot h_F$. Subsequently, another, new set of parameters is chosen as the slow set for the next k optimisation steps.

In the present work, we demonstrate that a form of multirate method can also be used in continual learning. Similarly to [62], [152] and [3], we make use of the FIM to identify important model parameters. However, instead of penalising the change in those parameters by augmenting the loss function by a regularisation term, we precondition the learning rates so that important parameters are updated with smaller step sizes and less important parameters are updated with larger step sizes. We aim to show that this preconditioned learning rates approach imposes a softer constraint on parameters than the introduction of a penalty term in the loss function.

5.6 Experiments

We test our preconditioned learning rates method on two popular continual learning benchmark cases. Permuted MNIST, though criticized as non-representative of actual continual learning scenarios, is a popular benchmark and can thus be used as a first test case for our algorithm. As a next step, we will test our algorithm on the single-headed Split MNIST benchmark.

5.6.1 Permuted MNIST

Permuted MNIST is a standard continual learning benchmark, first introduced by [41] and has been widely used in testing of continual learning methodologies [59], [138],[62]. However, it has received criticism for not being representative of actual continual learning scenarios, often masking the weaknesses of online learning algorithms. Randomly permuting the pixels of a set of images to obtain a new dataset leads to a new task that is very different to the previous one, thus making differentiation between tasks easier for the model. In real world scenarios, new datasets are similar to older ones, making the distinction amongst them harder [27]. We will, however, as mentioned use it as a first benchmark for testing our algorithm.

Our model is a fully-connected neural network with four layers. The input layer has 784 (28x28) nodes, the hidden layers have 400 nodes each and the output layer has 10 nodes. We train the network using stochastic gradient descent with momentum. For each model parameter, the FIM is computed and the learning rate for that parameter is set to be proportional to the inverse of the corresponding FIM value.

The training cost can be greatly reduced if we choose not to update (or update every k steps) a certain percentage of the model weights corresponding to the higher FIM values. Apart from saving computational resources, this approach can also reduce catastrophic forgetting as it leaves important parameters completely unchanged.

Algorithm 1 Multirate training

- Compute diagonal of Fisher information matrix $F_d = (f_{ii}), \quad i = 1,2,\ldots,d$, where $F = (f_{ij}) \in \mathbb{R}^{dxd}$ is the Fisher information matrix.
- Set all zero elements of F_d to be equal to a small number ε. This is to avoid division by zero in the next step.
- Compute the element-wise inverse of the diagonal of the Fisher information matrix as $F_d^{-1} = (1/f_{ii}), \quad i = 1,2,\ldots,N$.
- Compute $F_{d,norm}^{-1} = F_d^{-1} / \max(F_d^{-1})$
- Compute learning rates vector as $\boldsymbol{h} = h_{max} F_{d,norm}^{-1}$
- Optional: Set the lowest x% of the learning rates to 0. This can help with catastrophic forgetting and also reduce computational cost.

  for i in 1,2,..., m

  $p := \mu p + \nabla_\theta \mathcal{L}(\theta)$

  $\theta := \theta - \boldsymbol{h} \cdot p$

  end

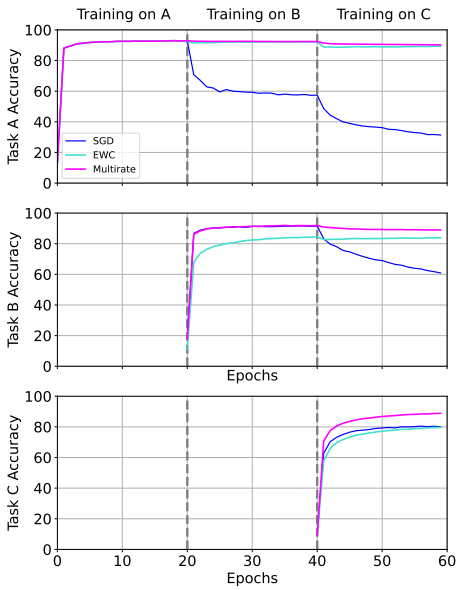


Figure 5.2: Comparing EWC and Multirate training performance of a sequence of three permuted MNIST tasks. We can see that the multirate approach maintains high precision on previous tasks, while also achieving high precision on new tasks. EWC also maintains good performance on previous tasks, but its ability to quickly achieve high accuracy on new tasks seems to be hindered, possibly by the harder nature of the constraint imposed to the model parameters through the quadratic penalty term.

5.6.2 Single-headed split MNIST

The single-headed split MNIST benchmark was introduced by [27] as a realistic continual learning benchmark. In split MNIST, the training set is divided into five groups (0-1,2-3,4-5,6-7,8-9). The model is trained on the first task, i.e. differentiating between 0 and 1 and then on the other four tasks (2-3, 4-5, 6-7, 8-9) sequentially. The term single-headed means that the five tasks share the same output layer (head) in the model. The model therefore has ten output nodes and is always required to choose amongst the digits 0-9 having no information about the task identity. Due to the fact that the tasks are much more similar to each other than the corresponding tasks in the permuted MNIST benchmark, avoiding catastrophic forgetting for Split MNIST proves to be harder.

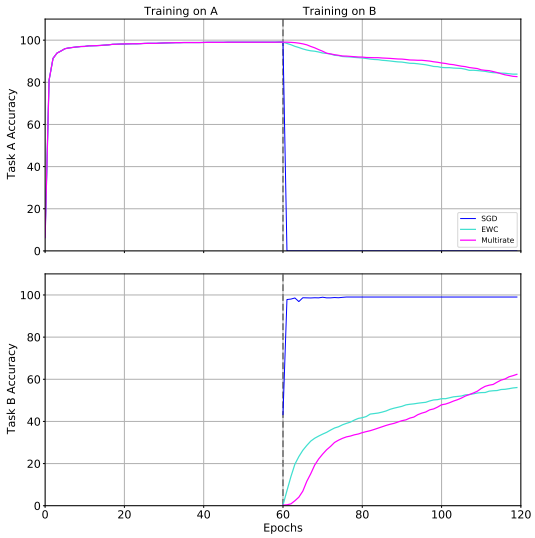


Figure 5.3: Comparing EWC and Multirate training performance on a sequence of two Split MNIST tasks. Here EWC and our multirate method have a similar performance. The split MNIST continual learning benchmark is harder than Permuted MNIST as explained by [27] and a much more realistic continual learning scenario.

5.7 Convergence Analysis

We will now perform convergence analysis of our proposed preconditioned learning rates algorithm. The following proof is for plain SGD (no momentum). We make the following assumptions about the function $f(\theta)$ that is being optimised, where θ are the model parameters.

Assumption 5.7.1. *Assume $f : \mathbb{R}^d \to \mathbb{R}$ and $f \in C^1(\mathbb{R}^d, \mathbb{R})$, is L-smooth, i.e. $\exists L > 0$:*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2,$$

$\forall x, y \in \mathbb{R}^d$.

Assumption 5.7.2. *Assume that the second moment of the stochastic gradient of f is bounded from above, i.e. $\exists M > 0$:*

$$\|\nabla f_{x_i}(\theta)\|_2^2 \leq M, \quad \forall \theta \in \mathbb{R}^d, \ \ \forall x_i \in \mathbb{R}^m,$$

where x_i is a single training example.

Lemma 5.7.1. *Every L-smooth function $f : \mathbb{R}^d \to \mathbb{R}$ can be locally bounded above by a quadratic, i.e.*

$$f(x) - f(y) \leq \nabla f(\theta^t)(x - y) + \frac{L}{2}\|x - y\|_2^2,$$

$\forall x, y \in \mathbb{R}^d$.

5.7.1 Multirate SGD based on Fisher Information Matrix

Let us now assume we wish to find $\theta^* = \min_{\theta \in \mathbb{R}^\times} f(\theta)$, where $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth. Using Lemma 5.7.1, and assuming we follow the optimisation process described in Algorithm 1 (but without momentum) , we have

$$f(\theta^{t+1}) - f(\theta^t) \leq \nabla f(\theta^t)(\theta^{t+1} - \theta^t) + \frac{L}{2}||\theta^{t+1} - \theta^t||_2^2. \tag{5.4}$$

The weights update rule is as follows

$$\theta^{t+1} = \theta^t - h * \nabla f_{x_i}(\theta^t), \tag{5.5}$$

where $\theta \in \mathbb{R}^d$, $h \in \mathbb{R}^d$ and the symbol "*" denotes element-wise multiplication between vectors. Using eq. 5.5, eq. 5.7.1 can be written as

$$\begin{aligned} f(\theta^{t+1}) - f(\theta^t) &\leq \nabla f(\theta^t)(\theta^{t+1} - \theta^t) + \frac{L}{2}||\theta^{t+1} - \theta^t||_2^2 \\ &= -\nabla f(\theta^t) \cdot (h * \nabla f_{x_i}(\theta^t)) + \frac{L}{2}||-h * \nabla f_{x_i}(\theta^t)||_2^2. \end{aligned}$$

Applying double expectations on both sides we have

$$\begin{aligned}\mathbb{E}[f(\theta^{t+1}) - f(\theta^t)] &\leq -\mathbb{E}[\nabla f(\theta^t) \cdot (h * \nabla f_{x_i}(\theta^t))] + \frac{L}{2}\mathbb{E}||h * \nabla f_{x_i}(\theta^t)||_2^2 \\ &= -\underbrace{\mathbb{E}\big[\nabla f(\theta^t) \cdot \mathbb{E}[h * \nabla f_{x_i}(\theta^t)]\big]}_{\mathcal{A}} + \frac{L}{2}\underbrace{\mathbb{E}||h * \nabla f_{x_i}(\theta^t)||_2^2}_{\mathcal{B}}. \end{aligned} \tag{5.6}$$

Using that $\mathbb{E}[\nabla f_{x_i}(\theta)] = \nabla f(\theta)$, term $\mathcal{A}$ becomes

$$\begin{aligned}&\mathcal{A} = \mathbb{E}\big[\nabla f(\theta^t) \cdot \mathbb{E}[h * \nabla f_{x_i}(\theta^t)]\big] = \mathbb{E}[\nabla f(\theta^t) \cdot (h * \nabla f(\theta^t))] \Rightarrow \\ &\mathcal{A} = \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 \end{aligned} \tag{5.7}$$

where $\circ$ denotes an element-wise operation on a vector. Denoting the maximum element of h by $h_{max} \in \mathbb{R}_0^+$, term $\mathcal{B}$ becomes

$$\mathcal{B} = \mathbb{E}||h * \nabla f_{x_i}(\theta^t)||_2^2 \leq ||h_{max}\nabla f_{x_i}(\theta^t)||_2^2 = h_{max}^2\mathbb{E}||\nabla f_{x_i}(\theta^t)||_2^2$$

Assuming that the second moment of the stochastic gradient is bounded above, i.e.

$$||\nabla f_{x_i}(\theta)||_2^2 \leq M, \quad \forall \theta \in \mathbb{R}^d,$$

we can write

$$\mathcal{B} \leq h_{max}^2 M \tag{5.8}$$

Using the obtained equations for terms $\mathcal{A}$ and $\mathcal{B}$, we can now rewrite eq.5.6 as

$$\mathbb{E}[f(\theta^{t+1}) - f(\theta^t)] \leq -\mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 + \frac{L}{2}h_{max}^2 M. \tag{5.9}$$

For T steps, we have

$$\begin{aligned}\mathbb{E}\big[f(\theta^T) - f(\theta^0)\big] =& \mathbb{E}\big[f(\theta^T) - f(\theta^{T-1}) + f(\theta^{T-1}) - f(\theta^{T-2}) - f(\theta^{T-2}) + ... \\ &... + f(\theta^3) - f(\theta^2) + f(\theta^2) - f(\theta^1) + f(\theta^1) - f(\theta^0)\big] \\ =& \mathbb{E}\big[f(\theta^T) - f(\theta^{T-1})\big] + ... + \mathbb{E}\big[f(\theta^1) - f(\theta^0)\big]. \end{aligned} \tag{5.10}$$

For each of the expectation terms on the right hand side of eq.5.10, the inequality of eq.5.9 holds. We therefore have

$$\begin{cases} \mathbb{E}[f(\theta^T) - f(\theta^{T-1})] \leq -\mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^{T-1})||_2^2 + \frac{L}{2}h_{max}^2 M \\ \mathbb{E}[f(\theta^{T-1}) - f(\theta^{T-2})] \leq -\mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^{T-2})||_2^2 + \frac{L}{2}h_{max}^2 M \\ \vdots \\ \mathbb{E}[f(\theta^1) - f(\theta^0)] \leq -\mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^0)||_2^2 + \frac{L}{2}h_{max}^2 M \end{cases}$$

Adding up the above inequalities we get the expectation of the difference of the loss value at time T with the loss at time 0, i.e.

$$\mathbb{E}[f(\theta^T) - f(\theta^0)] \leq -\sum_{t=0}^{T-1} \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 + \frac{L}{2} h_{max}^2 MT. \tag{5.11}$$

If θ^* is the global minimiser of f, we have

$$f(\theta^*) \leq \mathbb{E}[f(\theta^T)] \Rightarrow f(\theta^*) - f(\theta^0) \leq \mathbb{E}[f(\theta^T)] - f(\theta^0) = \mathbb{E}[f(\theta^T) - f(\theta^0)].$$

Eq.5.11 then becomes

$$\begin{aligned}
&f(\theta^*) - f(\theta^0) \leq \mathbb{E}[f(\theta^T) - f(\theta^0)] \leq -\sum_{t=0}^{T-1} \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 + \frac{L}{2} h_{max}^2 MT \Rightarrow \\
&f(\theta^*) - f(\theta^0) \leq -\sum_{t=0}^{T-1} \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 + \frac{L}{2} h_{max}^2 MT \Rightarrow \\
&\frac{f(\theta^*) - f(\theta^0)}{h_{max} T} \leq -\frac{1}{h_{max} T} \sum_{t=0}^{T-1} \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 + \frac{L}{2} h_{max} M \Rightarrow \\
&\frac{1}{h_{max} T} \sum_{t=0}^{T-1} \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 \leq \frac{f(\theta^0) - f(\theta^*)}{h_{max} T} + \frac{L}{2} h_{max} M
\end{aligned}$$

However,

$$\mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 \geq \mathbb{E}||h_{min}^{1/2} * \nabla f(\theta^t)||_2^2 = h_{min} \mathbb{E}||\nabla f(\theta^t)||_2^2,$$

where $h_{min} = \min_{1 \leq i \leq N} h_i$.

Therefore

$$\begin{aligned}
&\frac{h_{min}}{h_{max} T} \sum_{t=0}^{T-1} \mathbb{E}||\nabla f(\theta^t)||_2^2 \leq \frac{1}{h_{max} T} \sum_{t=0}^{T-1} \mathbb{E}||h^{\circ 1/2} * \nabla f(\theta^t)||_2^2 \leq \frac{f(\theta^0) - f(\theta^*)}{h_{max} T} + \frac{L}{2} h_{max} M \Rightarrow \\
&\frac{h_{min}}{h_{max} T} \sum_{t=0}^{T-1} \mathbb{E}||\nabla f(\theta^t)||_2^2 \leq \frac{f(\theta^0) - f(\theta^*)}{h_{max} T} + \frac{LM}{2} h_{max}.
\end{aligned}$$

The above inequality can be rewritten as

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}||\nabla f(\theta^t)||_2^2 \leq \frac{f(\theta^0) - f(\theta^*)}{h_{\min} T} + \frac{LM}{2} \frac{h_{\max}^2}{h_{\min}}. \tag{5.12}$$

We note that the first term on the right hand side asymptotically tends to zero as $T \to \infty$. The term $\frac{L}{2} M \frac{h_{\max}^2}{h_{\min}}$ is the one that controls the upper bound, based on the choice of $h_{\max}$. The minimum step size is determined based on $h_{\max}$ and the values of the Fisher information matrix. The Fisher information matrix therefore also indirectly influences the upper bound through $h_{\min}$.

5.8 Next steps

The authors of [141] have demonstrated that a multirate method with two groups of parameters, namely fast (last layer(s)) and slow (earlier layers), can be used as a successful transfer learning technique that achieves very high accuracy at low computational cost. The reasoning behind the split into fast and slow parameters is that earlier layers capture more general features and can therefore be updated less frequently. We want to examine whether this is reflected on the FIM values for earlier layers. If so, that would offer a justification as to why this splitting is successful. It would also be interesting to see how the multirate method proposed by [141] would perform in combination with stochastic gradient Langevin dynamics something that the authors have listed as a possible future direction.

We also want to examine whether a splitting of parameters based on the FIM values would allow efficient transfer learning - we have so far only demonstrated our method works for continual learning applications, where we care about maintaining performance on older tasks.

It would also be interesting to look at applications of continual learning for molecular dynamics problems. One of the possible future directions would be to study how we can learn potential energy surfaces on the fly. Such a framework could work by sampling datapoints from an approximation of the potential energy surface and use them to refine the potential using our preconditioned learning rates approach as we go along.

Finally, it would be interesting to examine whether our multirate approach can be used in combination with our FAD or cubic damping optimisers. Using an optimiser like Adam that adapts the learning rates could potentially interfere with the FIM-based preconditioning of the learning rates we perform in PLR. Therefore our method might not benefit from acceleration mechanisms that come with adaptive step sizes. It would be interesting to examine other ways in which the descent dynamics could be accelerated. One way to do that would be to combine the multirate approach with the iKFAD or CD dynamics which, from what we have observed so far, can offer improved convergence properties without the need for step size adaptation.

Chapter 6

Conclusions

In this thesis we have introduced and studied new, dynamics-inspired optimisation methods. We have considered an alternative to dissipated Hamiltonian descent as a scheme for local optimisation of smooth potential functions based on Nosé–Hoover dynamics and its generalizations; these methods embed Hamiltonian dynamics in a framework including one or more auxiliary variables. We have only explored some of the simplest methods that can be built on this foundation and have shown convergence, both for the dynamics itself, and for simple discretisations of the dynamics.

In terms of practical application, it seems the KFAD scheme, which is particularly simple to implement and which introduces virtually no computational overhead, is already of great potential utility. The experiments on Rosenbrock systems hint at the robustness of these methods. The efficiency of the methods was also demonstrated for more challenging examples, including a 192 degree-of-freedom Morse cluster, with highly anharmonic potential landscape, where the KFAD method was able to reach near-global energy minimizing states. For this global optimisation problem there may of course be other schemes besides LDHD which would make more realistic comparisons, but we were impressed that a pure dynamics-based method could find global optima without the trapping clearly visible in the LDHD runs (note in particular the results for the Morse cluster with 10^6 steps and $\delta t = 0.08$, where 86% of the miminizers obtained have energies within 1% of the global minimum).

Although the FFAD method showed a strong benefit in terms of effectiveness in suppressing oscillations in initial experiments with the Rosenbrock problem, it generally performed worse than KFAD. This is due to the nature of the FFAD dynamics, which as we explain does reduce oscillations, but also tends to slow down descent towards the minimum. We suspect that the idea of adding some directed cubic damping will nonetheless prove useful in some other settings.

In Chapter 3, we tested our FAD methods on simple optimisation benchmarks (Rosenbrock) as well as on particle cluster optimisation applications. We also observed that the FAD methods are related to nonlinear (cubic) damping, a relation that is examined in Chapter 3, where we gain further insight into the range of application for FAD methods and cubic damping methods. In Chapter 4, we examine systems with stochastic forcing (stochastic gradients) that

arise in deep learning applications and demonstrate that our methods can lead to significant performance improvement on certain machine learning benchmarks compared to mSGD and Adam. In Chapter 5, we introduce a preconditioned learning rates continual learning approach and demonstrate its potential on two standard continual learning benchmarks. It would be interesting to examine how our FAD and cubic damping-based methods perform in a continual learning setting.

This thesis opens up several new directions of research. We have only scratched the surface of the FAD family of methods and the design of effective control mechanisms. The dynamics of some of our optimisers are closely related to dynamics used in gradient-based sampling methods, such as Adaptive Langevin and Nosé–Hoover Dynamics [47, 55, 70, 99]. Optimisation and sampling are however, as mentioned, closely related and the connection between them has been the study of several recent works [80, 146]. Specifically, the authors of [80] show that sampling using underdamped Langevin dynamics is equivalent to optimisation over the probability space with the Kullback-Leibler divergence being the loss function and show, using a suitable Lyapunov function that underdamped Langevin has a convergence rate that is accelerated compared to that of overdamped Langevin dynamics, as underdamped Langevin incorporates a momentum mechanism similar to Nesterov acceleration. Accelerated convergence of underdamped Langevin dynamics has also been recently shown in [13]. In practice, underdamped Langevin dynamics as well as other higher-order methods have been used within the molecular dynamics community, as they are known to speed up convergence and allow for improved discretisation schemes with higher accuracy. This begs the question of what other acceleration mechanisms and algorithms from optimisation and molecular dynamics can be used in a sampling setting and conversely, how ideas from sampling can be used in optimisation, for example in the context of global optimisation. Examining the interplay between sampling and optimisation and working from a dynamical systems perspective has the potential to lead to the development of optimisers and samplers for ML problems which are fast, reliable and interpretable.

Appendix A

Additional Numerics from Chapter 4

A.1 CIFAR-10

We train a standard ResNet-34 PyTorch model on CIFAR-10 data while setting $\gamma = 0$ for iKFAD, CD and CADAM.

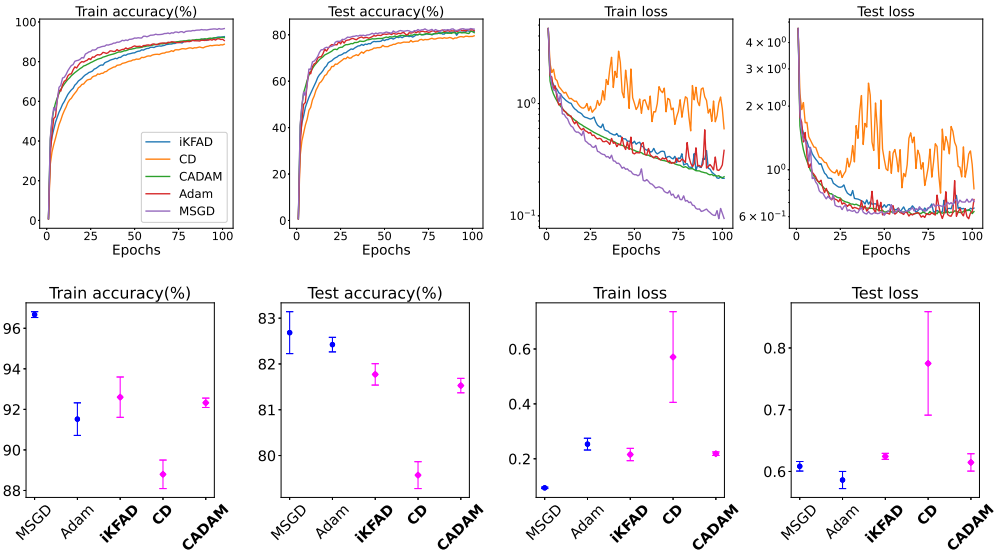


Figure A.1: Training a ResNet-34 on CIFAR-10 data (batch size 64). The plots show the training and test loss curves (averaged over three different random seeds) and final loss and accuracy values for the optimal hyperparameters. Initialisation of adaptive friction $\xi(0) = 0$, $\gamma = 0$, $\mu \in [0.00001, 1.0]$ and $\alpha \in [0.01, 1.0]$.

We observe that setting the linear friction γ to zero negatively impacts the performance of our methods for this dataset.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.013	0.0	0.791	3.05e-4	-	-	-	-
CD	0.036	0.0	-	-	1380.56	-	-	-
CADAM	0.002	0.0	0.013	-	1342554.56	-	-	-
mSGD	0.017	-	-	-	-	-	-	0.86
Adam	0.024	-	-	-	-	0.97	0.97	-

Table A.1: CIFAR-10 optimal hyperparamers corresponding to the results in Figure A.1.

Training of ResNet-34 [79] on CIFAR-10 data.

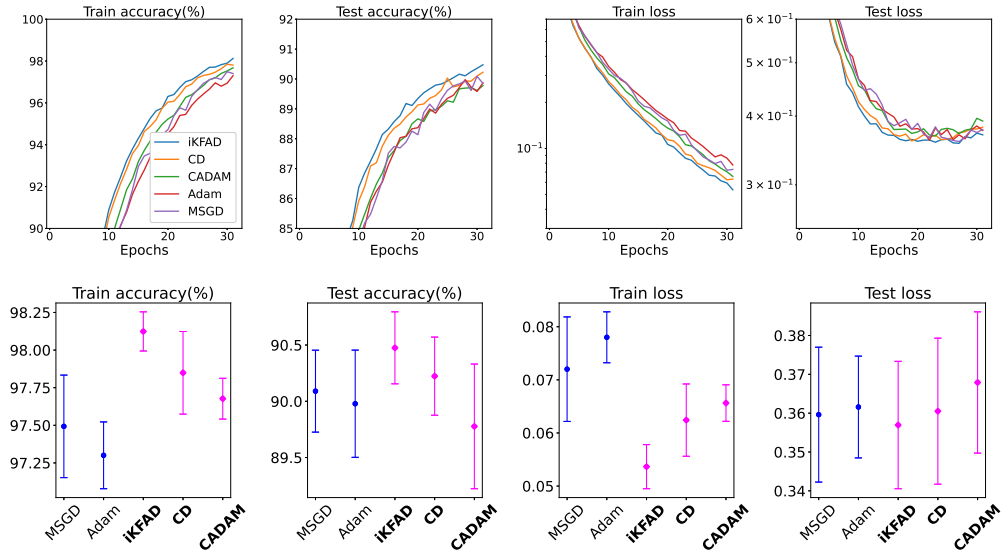


Figure A.2: Training a ResNet-34 (see [79]) on CIFAR-10 data (batch size 64). Parameter ranges: $\gamma \in [1e-8, 0.1]$, $\mu \in [1e-12, 10]$, $c \in [1, 1e10]$ and $\alpha \in [0.001, 1.0]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0724	0.0009	0.1352	7.92e-06	-	-	-	-
CD	0.0470	0.0015	-	-	274464.56	-	-	-
CADAM	0.0001	0.2612	0.1433	-	1369.19	-	-	-
mSGD	0.0066	-	-	-	-	-	-	0.930
Adam	0.0003	-	-	-	-	0.854	0.998	-

Table A.2: CIFAR-10 optimal hyperparamers for the results presented in Figure A.2.

Next, we again train the ResNet-34 from [79] but change the ranges of α, μ, γ, c to see how this affects our results. All of the methods seem to be comparable for this particular setting.

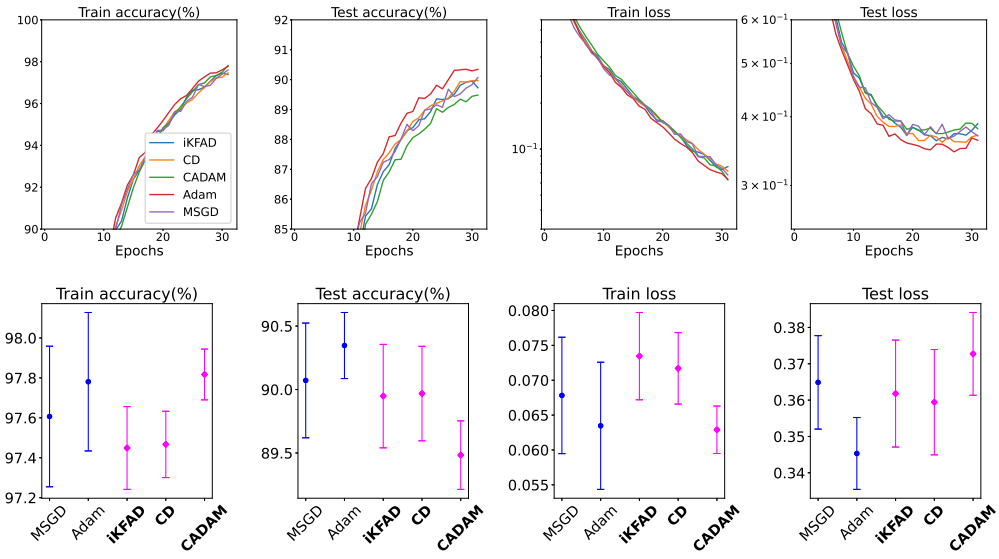


Figure A.3: Training a ResNet-34 on CIFAR-10 data (batch size 64). The plots show the training and test loss curves (averaged over three different random seeds) and final loss and accuracy values for the optimal hyperparameters. Parameter ranges $\gamma \in [0.1, 10.0]$, $\mu \in [0.01, 10.0]$ and $\alpha \in [0.01, 10.0]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0973	0.40	0.402	1.87	-	-	-	-
CD	0.0448	0.16	-	-	56814.47	-	-	-
CADAM	0.0037	3.55	0.005	-	82552.90	-	-	-
mSGD	0.0260	-	-	-	-	-	-	0.86
Adam	0.0008	-	-	-	-	0.97	0.99	-

Table A.3: CIFAR-10 optimal hyperparamers with $\gamma \in [0.1, 10.0]$, $\mu \in [0.01, 10.0]$ and $\alpha \in [0.01, 10.0]$.

A.2 CIFAR-100

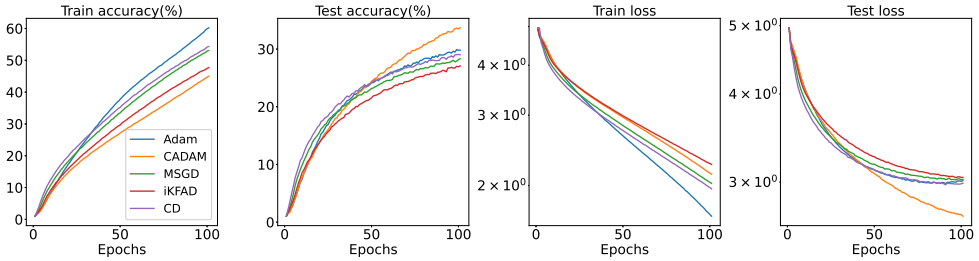


Figure A.4: Training a ResNet-50 on CIFAR-100 data (batch size 128). Initialisation of adaptive friction $\xi(0) = 0$. We take $\gamma \in [0.01, 10.0]$, $\mu \in [0.0001, 10.0]$ and $\alpha \in [0.01, 10.0]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0156	8.83	0.19	2.33	-	-	-	-
CD	0.0140	6.21	-	-	890.41	-	-	-
CADAM	0.0001	0.14	0.60	-	42050.44	-	-	-
mSGD	0.0004	-	-	-	-	-	-	0.84
Adam	1.65e-05	-	-	-	-	-	0.93	0.88

Table A.4: CIFAR-100 optimal hyperparamers with $\xi(0) = 0$, $\gamma \in [0.01, 10.0]$, $\mu \in [0.0001, 10.0]$ and $\alpha \in [0.01, 10.0]$.

Here, we train a pretrained EfficientNetB5, using a batch size of 256, $\xi(0) = 1$, with $\gamma = 0$, $\mu \in [0.00001, 1.0]$ and $\alpha \in [0.01, 1.0]$.

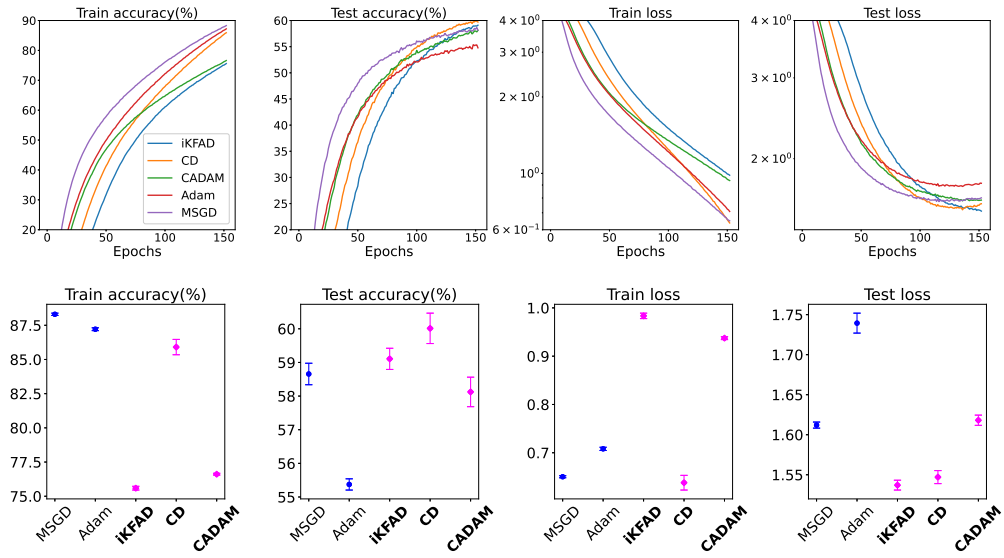


Figure A.5: Training EfficientNet-B5 in CIFAR-100, batch size=256. Averaging over 5 different random seeds. For iKFAD, CD and CADAM, we set $\gamma = 0$. For iKFAD we search for μ and α within the ranges $\mu \in [0.00001, 1.0]$ and $\alpha \in [0.01, 1.0]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0018	0.0	0.15	0.0085	-	-	-	-
CD	0.0026	0.0	-	-	149787.48	-	-	-
CADAM	6.38e-05	0.0	0.15	-	881468.22	-	-	-
mSGD	0.0011	-	-	-	-	-	-	0.88
Adam	1.01e-05	-	-	-	-	0.91	0.98	-

Table A.5: Training efficientnet-b5 on CIFAR-100. Optimal hyperparamers with $\xi(0) = 0$, $\gamma = 0$, $\mu \in [0.00001, 1.0]$ and $1\alpha \in [0.01, 1.0]$.

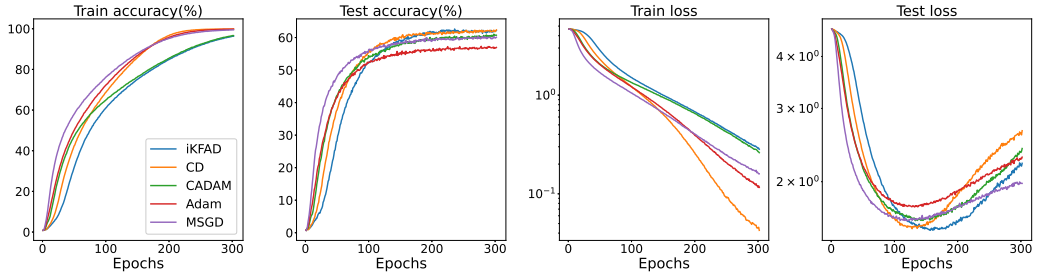


Figure A.6: Rerunning averaging of Figure A.5 for 300 epochs.

Next, we train a custom ResNet-34 [79] on CIFAR-100.

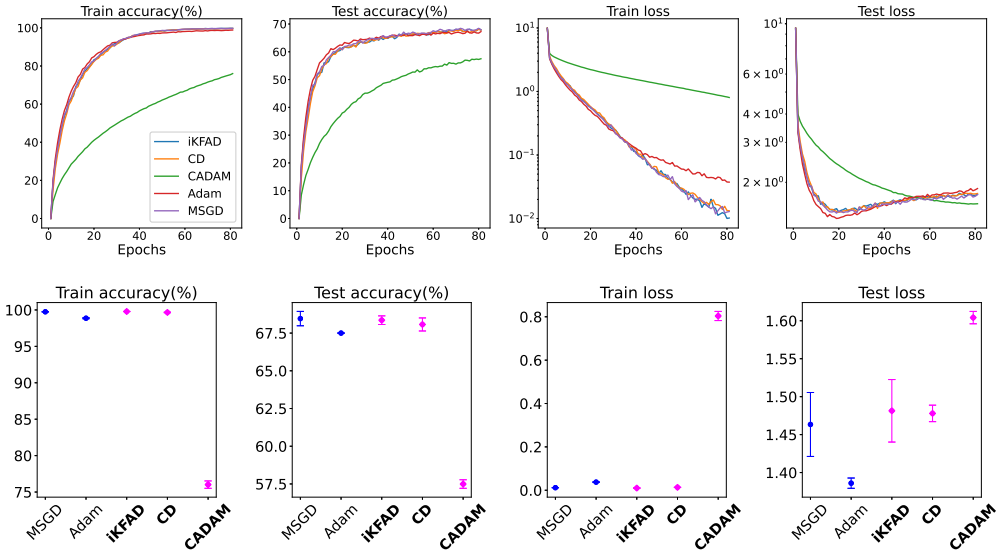


Figure A.7: Training a ResNet-34 on CIFAR-100 data (batch size 128). The plots show the training and test loss curves (averaged over three different random seeds) and final loss and accuracy values for the optimal hyperparameters. Parameter ranges $\gamma \in [1e-7, 10]$, $\mu \in [1e-10, 10]$, $c \in [0.1, 1e8]$ and $\alpha \in [0.001, 1.0]$.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.04	3.34	0.53	0.00025	-	-	-	-
CD	0.04	2.81	-	-	19865.62	-	-	-
CADAM	9.24e-05	0.17	0.02	-	5787504.89	-	-	-
mSGD	0.0010	-	-	-	-	-	-	0.92
Adam	7.78e-05	-	-	-	-	0.98	0.99	-

Table A.6: CIFAR-100 optimal hyperparamers with $\gamma \in [1e-7, 10]$, $\mu \in [1e-10, 10]$, $c \in [0.1, 1e8]$ and $\alpha \in [0.001, 1.0]$.

A.3 QNLI

We set $\gamma = 0$ for iKFAD, CD and CADAM. The results and corresponding parameters selected by *Optuna* can be seen in figure A.8 and table A.7 respectively. The criterion used for these runs was mean validation loss over the last five epochs of training.

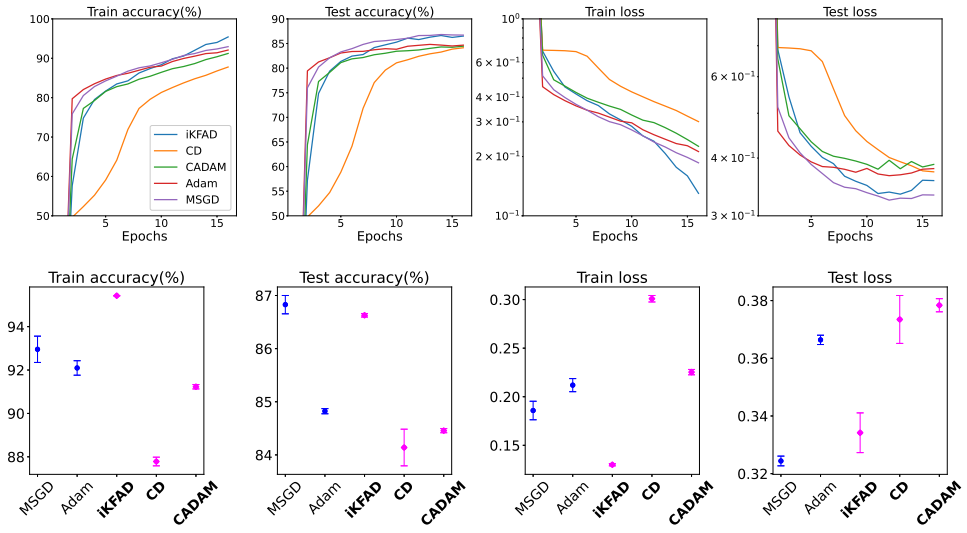


Figure A.8: Training DistilBERT (from pretrained) on QNLI data (batch size 16). Initialisation of adaptive friction $\xi(0) = 1$. iKFAD parameter ranges $\gamma = 0, \alpha \in [0.01, 1.0], \mu \in [0.00001, 1.0]$. The criterion used for hp selection was mean validation loss over the last five epochs.

	h	γ	α	μ	c	β_1	β_2	ρ
iKFAD	0.0021	0.0	0.03	0.0011	-	-	-	-
CD	0.0012	0.0	-	-	9529439.42	-	-	-
CADAM	1.21e-05	0.0	0.40	-	228347.23	-	-	-
mSGD	0.0001	-	-	-	-	-	-	0.938
Adam	1.62e-06	-	-	-	-	0.900	0.999	-

Table A.7: QNLI optimal hyperparamers with $\xi(0) = 1$, $\gamma = 0, \alpha \in [0.01, 1.0], \mu \in [0.00001, 1.0]$. Training a pretrained DistilBERT model.

Bibliography

[1] A.F. Agarap. Deep learning using rectified linear units (ReLU). *arXiv preprint arXiv:1803.08375*, 2018.

[2] A. Aich. Elastic weight consolidation (ewc): Nuts and bolts. *arXiv preprint arXiv:2105.04093*, 2021.

[3] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory aware synapses: Learning what (not) to forget. pages 139–154, 2018.

[4] S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[5] J. Ba, G.E. Hinton, V. Mnih, J.Z. Leibo, and C. Ionescu. Using fast weights to attend to the recent past. *Advances in Neural Information Processing Systems*, 29, 2016.

[6] A. Babister. Non-linear differential equations having both cubic damping and stiffness. Technical Report 7601, University of Glasgow, 1976. Department of Aeronautics and Fluid Mechanics.

[7] F. Bach. *Learning Theory from First Principles*. MIT press, 2024.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[9] A. Bhatt, D. Floyd, and B. Moore. Second order conformal symplectic schemes for damped Hamiltonian systems. *Journal of Scientific Computing*, 66:1234–1259, 2016.

[10] S. Blanes, F. Casas, and A. Murua. Splitting methods for differential equations. *arXiv preprint arXiv:2401.01722*, 2024.

[11] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[12] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[13] Y. Cao, J. Lu, and L. Wang. On explicit L^2-convergence rate estimate for underdamped Langevin dynamics. *Archive for Rational Mechanics and Analysis*, 247(5):90, 2023.

[14] N. K. Chada, B. Leimkuhler, D. Paulin, and P. A. Whalley. Unbiased Kinetic Langevin Monte Carlo with Inexact Gradients. *arXiv preprint arXiv:2311.05025*, 2023.

[15] A. Choromanska, M. Henaff, M. Mathieu, G.B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. Proceedings of Machine Learning Research, 2015.

[16] A. Cossu, M. Ziosi, and V. Lomonaco. Sustainable artificial intelligence through continual learning. In *CAIP 2021: Proceedings of the 1st International Conference on AI for People: Towards Sustainable AI, CAIP 2021, 20-24 November 2021, Bologna, Italy*, page 103. European Alliance for Innovation, 2021.

[17] A.B. Da Silva and M. Gazeau. A general system of differential equations to model first-order adaptive algorithms. *The Journal of Machine Learning Research*, 21(1):5072–5113, 2020.

[18] H. Daneshmand, J. Kohler, A. Lucchi, and T. Hofmann. Escaping saddles with stochastic gradients. In *International Conference on Machine Learning*, pages 1155–1164. Proceedings of Machine Learning Research, 2018.

[19] W. C. Davidon. Variable metric method for minimization. *SIAM J. Optim.*, 1(1):1–17, 1991.

[20] C. de Masson D'Autume, S. Ruder, L. Kong, and D. Yogatama. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[23] J.P.K. Doye and D.J. Wales. The effect of the range of the potential on the structure and stability of simple liquids: from clusters to bulk, from sodium to C_{60}. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 29(21):4859, 1996.

[24] J.P.K. Doye and D.J. Wales. Structural consequences of the range of the interatomic potential. *J Chem Soc Faraday*, 93:4233–4243, 1997.

[25] J.P.K. Doye, D.J. Wales, and R.S. Berry. The effect of the range of the potential on the structures of clusters. *J. Chem. Phys.*, 103:4234–4249, 1995.

[26] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[27] S. Farquhar and Y. Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[28] S. Farquhar and Y. Gal. A unifying Bayesian view of continual learning. *arXiv preprint arXiv:1902.06494*, 2019.

[29] J.A. Feldman. Dynamic connections in neural networks. *Biological cybernetics*, 46(1):27–39, 1982.

[30] R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[31] R. Fletcher. *Practical Methods of Optimization*. Wiley-Interscience [John Wiley & Sons], New York, second edition, 2001.

[32] G. França, J. Sulam, D. P. Robinson, and R. Vidal. Conformal symplectic and relativistic optimization. *J. Stat. Mech. Theory Exp.*, (12):124008, 29, 2020.

[33] R.M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[34] X. Gao, M. Gürbüzbalaban, and L. Zhu. Global convergence of stochastic gradient hamiltonian monte carlo for non-convex stochastic optimization: Non-asymptotic performance bounds and momentum-based acceleration, 2018.

[35] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pages 797–842. Proceedings of Machine Learning Research, 2015.

[36] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[37] D.E. Golberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[38] D. Goldfarb. A family of variable-metric methods derived by variational means. *Math. Comp.*, 24:23–26, 1970.

[39] H. Goldstein. *Classical Mechanics*. Addison-Wesley, 1980.

[40] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[41] I.J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[42] T.H. Gronwall. Note on the derivatives with respect to a parameter of the solutions of a system of differential equations. *Annals of Mathematics*, 20(4):292–296, 1919.

[43] I. Gühring, G. Kutyniok, and P. Petersen. Error bounds for approximations with deep relu neural networks in w s, p norms. *Analysis and Applications*, 18(05):803–859, 2020.

[44] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360, 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[46] D.P. Herzog. Exponential relaxation of the Nosé–Hoover thermostat under Brownian heating. *Communications in Mathematical Sciences*, 16(8):2231–2260, 2018.

[47] W.G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Physical Review A*, 31(3):1695, 1985.

[48] R. Horst, P.M. Pardalos, and N. Van Thoai. *Introduction to Global Optimization*. Springer Science & Business Media, 2000.

[49] Y. Huang, Y. Zhang, J. Chen, X. Wang, and D. Yang. Continual learning for text classification with information disentanglement based regularization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2736–2746, Online, June 2021. Association for Computational Linguistics.

[50] F. Huszár. On quadratic penalties in elastic weight consolidation. *arXiv preprint arXiv:1712.03847*, 2017.

[51] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. Proceedings of Machine Learning Research, 2015.

[52] Z. Jia and B. Leimkuhler. A projective thermostatting dynamics technique. *Multiscale Modelling and Simulation*, 4:563–583, 2005.

[53] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.

[54] A. Jones and B. Leimkuhler. Adaptive stochastic methods for sampling driven molecular systems. *The Journal of Chemical Physics*, 135(8):084125, 2011.

[55] A. Jones and B. Leimkuhler. Adaptive stochastic methods for sampling driven molecular systems. *The Journal of Chemical Physics*, 135(8):084125, 2011.

[56] R.E. Kalman and J.E. Bertram. Control system analysis and design via the "second method" of Lyapunov. I. Continuous-time systems. *Trans. ASME Ser. D. J. Basic Engrg.*, 82:371–393, 1960.

[57] R.E. Kalman and J.E. Bertram. Control system analysis and design via the "second method" of Lyapunov. II. Discrete-time systems. *Trans. ASME Ser. D. J. Basic Engrg.*, 82:394–400, 1960.

[58] A. Karoni, B. Leimkuhler, and G. Stoltz. Friction-adaptive descent: a family of dynamics-based optimization methods. *J. Comput. Dyn.*, 10(4):450–484, 2023.

[59] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[60] P. Kidger and T. Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. Proceedings of Machine Learning Research, 2020.

[61] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 1412.6980, 2014.

[62] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[63] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[64] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[65] Maxime Laborde and Adam Oberman. A lyapunov analysis for accelerated gradient methods: from deterministic to stochastic case. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 602–612. PMLR, 26–28 Aug 2020.

[66] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[67] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[68] Y. LeCun, L. Bottou, G.B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[69] B. Leimkuhler and C. Matthews. Rational construction of stochastic numerical methods for molecular sampling. *Appl. Math. Res. eXpress*, 2013:34–56, 2013.

[70] B. Leimkuhler and C. Matthews. *Molecular Dynamics*. Springer, 2015.

[71] B. Leimkuhler, E. Noorizadeh, and F. Theil. A gentle stochastic thermostat for molecular dynamics. *J. Stat. Phys.*, 135:261–277, 2009.

[72] B. Leimkuhler and S. Reich. *Simulating Hamiltonian Dynamics*, volume 14 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2004.

[73] B. Leimkuhler, M. Sachs, and G. Stoltz. Hypocoercivity properties of adaptive Langevin dynamics. *SIAM J. Appl. Math.*, 80(3):1197–1222, 2020.

[74] B.J. Leimkuhler, D. Paulin, and P.A. Whalley. Contraction and convergence rates for discretized kinetic Langevin dynamics. *SIAM Journal on Numerical Analysis*, 62(3):1226–1258, 2024.

[75] Laurent Lessard, B. Recht, and A. P. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM J. Optim.*, 26(1):57–95, 2016.

[76] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[77] Z. Li and D. Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[78] D.G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*, volume 228 of *International Series in Operations Research & Management Science*. Springer, Cham, fifth edition, [2021] ©2021.

[79] L. Luo, Y. Xiong, Y. Liu, and X. Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

[80] Y.-A. Ma, N.S. Chatterji, X. Cheng, N. Flammarion, P.L. Bartlett, and M.I. Jordan. Is there an analog of Nesterov acceleration for gradient-based MCMC? *Bernoulli*, 27(3), 2021.

[81] A.L. Maas, A.Y. Hannun, Andrew Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 3. Atlanta, GA, 2013.

[82] C.J. Maddison, D. Paulin, Y.W. Teh, B. O'Donoghue, and A. Doucet. Hamiltonian descent methods. *arXiv preprint*, 1809.05042, 2018.

[83] J.C. Mattingly, A.M. Stuart, and D.J. Higham. Ergodicity for SDEs and approximations: locally Lipschitz vector fields and degenerate noise. *Stochastic Processes and their Applications*, 101(2):185–232, 2002.

[84] J. Maxwell. On governors. *Proc. Roy. Soc. Lon.*, 16:270–283, 1868.

[85] M. McCloskey and N.J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[86] R. McLachlan and M. Perlmutter. Conformal Hamiltonian systems. *Journal of Geometry and Physics*, 39:276–300, 2001.

[87] R.I. McLachlan and G.R.W. Quispel. Splitting methods. *Acta Numerica*, 11:341–434, 2002.

[88] M.A. Miller, J.P.K. Doye, and D.J. Wales. Structural relaxation in Morse clusters: Energy landscapes. *J. Chem. Phys.*, 110(1):328–334, 1999.

[89] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, second edition, 2018.

[90] C. Moucer, A.B. Taylor, and F. Bach. A systematic approach to Lyapunov analyses of continuous-time models in convex optimization. *SIAM Journal on Optimization (to appear)*, 2023.

[91] K.P. Murphy. *Machine Learning: a Probabilistic Perspective*. MIT press, 2012.

[92] Y. Nesterov. A method of solving a convex programming problem with convergence rate $\mathrm{O}(1/k^2)$. *Doklady Akademii Nauk SSSR*, 269(3):543, 1983.

[93] Y. Nesterov. *Introductory Lectures on Convex Optimization: A basic Course*, volume 87. Springer Science & Business Media, 2013.

[94] Y.E. Nesterov. A method of solving a convex programming problem with convergence rate $\mathrm{O}(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

[95] M.A. Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[96] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

[97] S. Nosé. A molecular-dynamics method for simulations in the canonical ensemble. *Mol. Phys.*, 52(2):255–268, 1984.

[98] S. Nosé. A unified formulation of the constant temperature molecular-dynamics methods. *J. Chem. Phys.*, 81(1):511–519, 1984.

[99] S. Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of Chemical Physics*, 81(1):511–519, 1984.

[100] N. Panananda, N.S. Ferguson, and T.P. Waters. The effect of cubic damping in an automotive vehicle suspension model. In *Computational Modelling and Analysis of Vehicle Body Noise and Vibration*, 2012.

[101] G.I. Parisi, R. Kemker, J.L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[102] R.G. Parker and R.L. Rardin. *Discrete Optimization*. Elsevier, 2014.

[103] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2(417):1, 2012.

[104] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Proceedings of Machine Learning Research, 2013.

[105] Z.K. Peng, Z.Q. Lang, X.J. Jing, S.A. Billings, G.R. Tomlinson, and L.Z. Guo. The transmissibility of vibration isolators with a nonlinear antisymmetric damping characteristic. 2010.

[106] P. Petersen and F. Voigtlaender. Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *arXiv preprint arXiv:1709.05289*, 2017.

[107] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17, 1964.

[108] H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statistics*, 22:400–407, 1951.

[109] H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statistics*, 22:400–407, 1951.

[110] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[111] A.A. Rusu, N.C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[112] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[113] J. M. Sanz Serna and K. C. Zygalakis. The connections between Lyapunov functions for some optimization algorithms and differential equations. *SIAM J. Numer. Anal.*, 59(3):1542–1565, 2021.

[114] J.M. Sanz Serna and K.C. Zygalakis. The connections between Lyapunov functions for some optimization algorithms and differential equations. *SIAM J. Numer. Anal.*, 59(3):1542–1565, 2021.

[115] A.M. Saxe, J.L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[116] J.C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *AAAI*, volume 86, pages 496–501, 1986.

[117] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[118] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, 24:647–656, 1970.

[119] J. Sherman and W.J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Statistics*, 21:124–127, 1950.

[120] H. Shin, J.K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

[121] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[122] U. Simsekli, L. Zhu, Y.-W. Teh, and M. Gurbuzbalaban. Fractional underdamped langevin dynamics: Retargeting sgd with momentum under heavy-tailed gradient noise. In *International conference on machine learning*, pages 8970–8980. Proceedings of Machine Learning Research, 2020.

[123] R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A.Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[124] J.C. Spall. *Introduction to stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons, 2005.

[125] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[126] R.K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[127] L. Stella, C. Lorenz, and L. Kantorovich. Generalized Langevin equation: An efficient approach to nonequilibrium molecular dynamics of open systems. *Phys. Rev. B*, 89:134303, 2014.

[128] G. Stoltz and Z. Trstanova. Langevin dynamics with general kinetic energies. *Multiscale Modeling & Simulation*, 16(2):777–806, 2018.

[129] W. Su, S. Boyd, and E. Candès. A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, 17:1–43, 2016.

[130] T. Sun, H. Ling, Z Shi, D. Li, and B. Wang. Training deep neural networks with adaptive momentum inspired by the quadratic optimization. *arXiv preprint*, 2110.09057, 2021.

[131] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. Proceedings of Machine Learning Research, 2013.

[132] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[133] M. Telgarsky. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

[134] M. Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. Proceedings of Machine Learning Research, 2016.

[135] M. Telgarsky. Neural networks and rational functions. In *International Conference on Machine Learning*, pages 3387–3393. Proceedings of Machine Learning Research, 2017.

[136] M. Tuckerman, B.J. Berne, and G.J. Martyna. Reversible multiple time scale molecular dynamics. *J. Chem. Phys.*, 97(3):1990–2001, 08 1992.

[137] G.M. Van de Ven and A.S. Tolias. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018.

[138] G.M. Van de Ven and A.S. Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[139] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[140] E.G. Virga. Rayleigh–Lagrange formalism for classical dissipative systems. *Phys. Rev. E*, 91:013203, 2015.

[141] T. Vlaar and B. Leimkuhler. Multirate training of neural networks. *arXiv preprint arXiv:2106.10771*, 2021.

[142] D.J. Wales. A microscopic basis for the global appearance of energy landscapes. *Science*, 293(5537):2067–2070, 2001.

[143] D.J. Wales. *Energy Landscapes: Applications to Clusters, Biomolecules and Glasses*. Cambridge University Press, 2004.

[144] D.J. Wales and J.P.K. Doye. Theoretical predictions of structure and thermodynamics in the large cluster regime. In T.P. Martin, editor, *Large Clusters of Atoms and Molecules (NATO ASI Series E: Vol. 313)*, pages 241–279. Kluwer Academic, Dordrecht, 1996.

[145] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S.R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[146] A. Wibisono. Sampling as optimization in the space of measures: The Langevin dynamics as a composite optimization problem. In *Conference on Learning Theory*, pages 2093–3027. Proceedings of Machine Learning Research, 2018.

[147] N. Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. MIT Press, 2nd edition, 1961.

[148] A.C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.

[149] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[150] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[151] M.D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *arXiv preprint arXiv:1311.2901*, 2013.

[152] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. Proceedings of Machine Learning Research, 2017.

[153] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[154] G. Zhang, J. Martens, and R. B Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[155] Y.-P. Zhu and Z.Q. Lang. Beneficial effects of antisymmetric nonlinear damping with application to energy harvesting and vibration isolation under general inputs. *Nonlinear Dynamics*, 108(4):2917–2933, 2022.