THE UNIVERSITY
of EDINBURGH

Stochastic Dynamics and Partitioned Algorithms for Model Parameterization in Deep Learning

Tiffany Joyce Vlaar

Doctor of Philosophy
University of Edinburgh
2022

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

Tiffany Joyce Vlaar

Lay Summary

This thesis is about mathematics for deep learning. Deep learning and neural networks have shown promising applicability to a variety of pressing real-world problems. They are enabling the development of self-driving cars, protein-folding, and nuclear fusion. However, the neural networks underpinning these breakthroughs often remain hard to interpret and understand. This thesis takes a step towards providing a stronger foundation for the mathematics behind neural networks, and in the process develops algorithms that improve the way in which they can be trained.

A neural network can be thought of as a machine that performs tasks by taking in an input, and returning an output. These inputs and outputs can be diverse and varied. For example, the input could be an image, and the output could be a classification of that image ("this image contains a cat"). The input could be a piece of text, and the output could be a continuation of that piece of text. The input could be a DNA sequence ("ACGTGTACGT"), and the output could be a 3D representation of a folded protein. The field of deep learning is therefore inherently interdisciplinary, due to its widespread applications. However, as we extend the scope of deep learning and neural networks it is crucial to also understand their limitations. For example, neural networks can easily be fooled and can misclassify images with high confidence ("this image of a cat is definitely a dog"). Therefore, taking a mathematical perspective is very important to allow us to obtain better guarantees when using neural networks for real-world applications.

The network's behaviour when performing tasks is determined by its parameters. An important aspect of deep learning involves finding ways to change the parameters to make the network better at the task of interest. Unfortunately, a lot is still unknown about why certain parameterization (training) schemes perform better than others. In this thesis we study and tease out properties of neural network training, and use those findings to improve how neural networks perform on new (unseen) tasks.

Abstract

In this thesis, we study model parameterization for deep learning applications. Part of the mathematical foundation for this work lies in stochastic differential equations and their constrained counterparts. We will study their role in deep learning, their properties, and their discretization. On the deep learning theory side we discuss questions around generalization error, optimization, the structure of neural network loss landscapes, and existing metrics of neural network training. Rather than aiming to exceed state-of-the-art results on benchmark datasets, our work in this area is aimed at studying and teasing out underlying properties of neural network optimization, and using those findings to obtain enhanced generalization performance. Our optimization schemes often draw inspiration from molecular dynamics and statistical physics and pave the way towards training robust and generalizable neural networks on datasets that arise in the physical sciences.

The contributions of this thesis are as follows: (1) We illustrate that embedding the loss gradient in a second order Langevin dynamics framework and using low temperatures leads to more exploration, increased robustness, and —in combination with partitioned integrators— can lead to enhanced generalization performance of neural networks on certain classification tasks. (2) We provide a general framework for using constrained stochastic differential equations to train deep neural networks. Constraints provide direct control of the parameter space, which allows us to directly study their effect on generalization. A statistical guarantee on the convergence of the training is provided, along with detailed implementation schemes for specific constraints –magnitude-based and orthogonality of the weight matrix– and extensive testing. (3) We illustrate the presence of latent multiple time scales in deep learning applications and introduce the use of multirate techniques for neural network training. We analyze the convergence properties of our multirate scheme and draw a comparison with vanilla stochastic gradient descent. As main application we show that using a multirate approach we can train deep neural networks for transfer learning applications in half the time, without losing generalization performance. (4) We re-evaluate existing deep learning metrics. In particular, we study the use of the loss along the linear path between the initial and final parameters of a network as a measure of the loss landscape. We show that caution is needed when using linear interpolation to make broader claims on the shape of the landscape and success of optimization. We also find that certain neural network layers are more sensitive to the choice of initialization and optimizer hyperparameter settings, and use these observations to design custom optimization schemes.

Contents

Overview

Deep learning has repeatedly defied conventional wisdom from statistical learning theory and obtained widespread success in various domains. However, our understanding of deep learning is currently still limited; there are many open questions on the generalization performance of neural networks, the success of optimizers like stochastic gradient descent, and other aspects of training. In this thesis we use mathematics to take a step towards building stronger foundations for deep learning, and in the process develop new methods for neural network parameterization that aim to make deep learning more practical and robust for real-world applications.

This thesis builds on five papers that were written during the PhD.

- Chapter 3 on "Partitioned integrators for thermodynamic parameterization of neural networks" was joint work with B. Leimkuhler and C. Matthews and appeared in *Foundations of Data Science* in 2019 [163].
- Chapter 4 on "Better training using weight-constrained stochastic dynamics" was joint work with B. Leimkuhler, T. Pouchon, and A. Storkey and appeared in *International Conference on Machine Learning (ICML)* in 2021 [165].
- Chapter 5 on "Multirate training of neural networks" is joint work with B. Leimkuhler and is currently under review. The work is available at *arXiv:2106.10771* [278].
- Section 6.1 on "What can linear interpolation of neural network loss landscapes tell us?" is joint work with J. Frankle and is currently under review. The work is available at *arXiv:2106.16004* [277].
- Section 6.2 on "Neural network behavior at the classification boundary" appeared as a *NeurIPS ICBINB workshop* paper in 2021 [276].

These papers involve the properties and discretization of stochastic differential equations and their constrained counterparts; the use of partitioned schemes that exploit the natural layer structure of neural networks; the enhanced robustness of specific aspects of neural network training; multirate training of neural networks; and re-evaluation of existing metrics for neural network loss landscape visualization and model evaluation. Unless stated otherwise, for clarity of presentation the respective chapters have the same structure as the accepted/submitted papers. This means that in some cases the mathematical contribution of the paper can be found in the supplementary discussion section of the chapter.

The preliminaries behind these contributions are provided in Chapter 1. Section 1.1 provides a high-level overview of deep learning, including a discussion on neural network training schemes and concepts from deep learning theory such as generalization and regularization of neural networks. Section 1.2 discusses relevant concepts from molecular dynamics, which build the foundations for several contributions in this thesis. We use discretized stochastic differential equations for neural network parameterization in Chapter 3 and Chapter 4. Intuitions from molecular dynamics also provide further context for the constrained algorithms described in Chapter 4 and the multirate schemes provided in Chapter 5. Although we predominantly focus on applications in vision and natural language processing throughout this thesis, we believe the strong connection with molecular dynamics will make our methods easily transferable and successful for datasets that arise in molecular dynamics and the physical sciences. Finally, since we take a Bayesian perspective on deep learning in Chapter 3, we shall introduce the Bayesian

framework, Markov Chain Monte Carlo methods, and Bayesian neural networks in Section 1.3. The original contributions of this thesis are summarized in Section 1.4.

Chapter 2 is dedicated to the foundations of stochastics, where Section 2.1 covers basics of stochastic processes. The properties and discretization of our main stochastic differential equations of interest, Langevin dynamics and adaptive Langevin dynamics, are discussed in Section 2.2 and Section 2.3, respectively. Section 2.4 discusses how stochastic differential equations arise in the context of neural network training. Finally, Section 2.5 lays the foundation for Chapter 4, where we use discretized constrained stochastic differential equations for neural network parameterization.

The main contributions of the thesis can be found in Chapters 3 to 6, followed by a conclusion and discussion for future work. Our contributions shed light on the nature of neural network optimization, and use mathematics to pave the way towards making neural networks more robust and generalizable.

Chapter 1

Introduction

This thesis focuses on methods for model parameterization. The main application on which we test these methods is for the training of deep neural networks, which are introduced in Section 1.1. For the development of these methods we often draw inspiration from molecular dynamics (Section 1.2). Section 1.3 discusses the Bayesian framework, Markov Chain Monte Carlo methods, and Bayesian neural networks. Finally, the contributions of this thesis are summarized in Section 1.4.

1.1 Deep Learning

In this section we provide a high-level introduction to deep learning. This thesis focuses on model parameterization and generalization for supervised learning settings (Section 1.1.1). The specific neural network architectures and training algorithms used in this thesis will be discussed in Section 1.1.2 and Section 1.1.3, respectively. Finally, Section 1.1.4 will discuss generalization and regularization of neural networks.

1.1.1 Supervised Learning

This thesis considers questions in optimization and generalization for supervised neural networks for classification problems. The network is given as data: inputs x and corresponding labels y, and as adjustable parameters θ. For each input pair (x, y) the network outputs a prediction $\hat{y} = f(x, \theta)$, which after training and updating θ should be close to the correct target y when measuring the discrepancy using a pre-defined loss function (see Figure 1.1). Performance is measured on a held out (unseen) test dataset. Neural networks are function approximators and can be trained efficiently with the backpropagation algorithm [240, 239]. The dawn of the era of big data, hardware advancements, and enhanced computational resources, combined with the success of deep (overparameterized) neural networks [147] has lead to their widespread use in recent years.

1.1.2 Neural Network Architectures

A neural network provides a non-linear transformation of inputs x into an output $\hat{y} = f(x; \theta, \mathcal{A})$ that depends on the model parameters θ and the architecture of the network $\mathcal{A}$. Knowledge about the properties of a dataset can be used to build in inductive biases into the architecture. This section covers all the neural network architectures used in this thesis.

1.1.2.1 Multilayer Perceptrons

A multilayer perceptron (MLP) is a feedforward neural network which is composed of an input layer (which takes in the input data), one or more hidden layers, and an output layer (which outputs the prediction) (see Figure 1.2). Each layer is parameterized by a weight matrix W_i and bias vector b_i and consists of a certain number of units –neurons–, which take in a weighted and biased signal vector and produce a scalar output, called their activation, which was passed

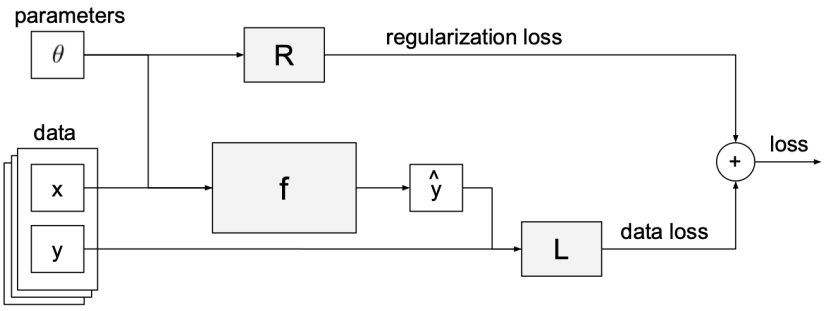

Figure 1.1: Figure adapted from [132]. A typical workflow for supervised learning problems. The function f that maps the inputs x to a prediction $\hat{y}$ is chosen to be a neural network parameterized by θ. The discrepancy between the true target y and the network prediction $\hat{y}$ is measured using a loss function L. A regularization term $R(\theta)$ (typically L_1 or L_2 regularization [268, 288, 106]) is often added to the loss to control the complexity of the learned function (see Section 1.1.4.2 for a discussion). Neural network training involves updating the neural network parameters θ such that the loss is minimized.

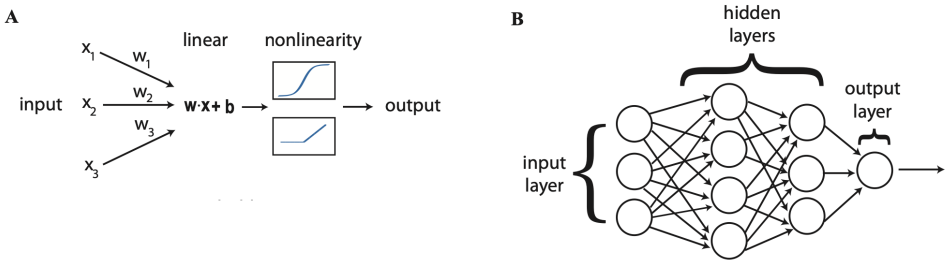

Figure 1.2: Figure adapted from [197]. (A) The inputs x_i are weighted and put through a non-linear activation function. (B) This is a standard feedforward fully-connected neural network architecture. The output of each layer is the input for the next layer.

through a (typically) non-linear activation function σ. For fully connected networks every neuron in each layer is connected with every neuron in the next layer. The outgoing activations of neurons are the input signals to neurons in the next layer. Connections between the neurons can be described by a directed acyclic graph [235].

The number of layers is called the depth of the neural network, while the number of neurons in each layer is called the width of each layer. Deep learning in this context therefore simply refers to using architectures with several hidden layers. The depth and width form hyperparameters of the architecture. Increasing the number of parameters increases the expressivity of the neural network, which means that the network can represent increasingly complex functions. The role of overparameterization and depth will be discussed in Section 1.1.4.

The operations performed in a multilayer perceptron with one hidden layer given some input

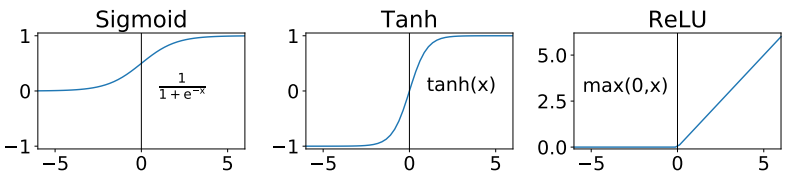

Figure 1.3: Possible choices for non-linear activation functions for neural networks.

data $x \in \mathbb{R}^N$ are

$$\hat{y} = \sigma(W_1 x + b_1)W_2 + b_2, \tag{1.1}$$

with weight matrices $W_1 \in \mathbb{R}^{N \times H}$ and $W_2 \in \mathbb{R}^{H \times Z}$, bias vectors $b_1 \in \mathbb{R}^H$ and $b_2 \in \mathbb{R}^Z$, and element-wise non-linearity σ. Various choices of non-linearity σ are possible, including a sigmoid, tanh, and rectified linear unit (ReLU) [205, 84] as shown in Figure 1.3. We will predominantly use the popular ReLU activation in this thesis. For other examples of activation functions we refer to [295]. For a classification problem, the output of the final layer of the neural network is passed through a softmax, which transforms the output for class label c to

$$\mathbb{P}(\hat{y} = c|x) = \frac{\exp(\hat{y}_c)}{\sum_{c'} \exp(\hat{y}_{c'})}. \tag{1.2}$$

The neural network then aims to minimize the loss, which is usually composed of a cross-entropy loss term (for a classification problem) and an (optional) regularization term

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c}\frac{\exp(y_c)}{\sum_{c'}\exp(y_{c'})}\log\left(\mathbb{P}(\hat{y} = c_i|x_i)\right) + \beta_R R(W). \tag{1.3}$$

The regularization term $R(W)$ penalizes the size of the norm of the weights with a regularization strength β_R. Typically one leaves the biases unregularised, as regularizing them could lead to significant underfitting [88]. Regularization is further discussed in Section 1.1.4.2.

1.1.2.2 Normalization Layers

Deep neural network architectures often incorporate normalization layers. Batch Normalization (BatchNorm) [121] is a popular choice, which adds BatchNorm layers to the architecture that standardize the inputs to the layer by the mean and variance of each batch of data. BatchNorm has an implicit regularization effect (Section 1.1.4.2) and essentially imposes a constraint on the neural network parameters. BatchNorm is widely used, but explanations for its success remain elusive [247, 297]. A variation is Layer Normalization (LayerNorm) [11], which removes the dependency to the data batch size and instead normalizes along the feature direction.

1.1.2.3 Convolutional Neural Networks

The use of other neural network architectures is typically motivated by the nature of the training problem. For applications in computer vision convolutional neural networks (CNNs) have achieved widespread success by building in properties such as locality and translational invariance.

To allow a multi-layer perceptron (MLP) to classify an input image with size $w \times h \times d$ (w width, h height, d number of color channels), the image has to be converted to a one-dimensional vector of size whd, which leads to unmanageable sizes of the network for larger image sizes and discards valuable information. In contrast to MLPs, CNNs take as input volume the pixels $w \times h \times d$ and introduce parameter sharing. For a MLP there is no difference between images whose pixels have been scrambled. CNNs, unlike MLPs, do not assume that the ordering of the inputs has no bearing on the results. Instead they assume that the inputs have some form of inbuilt structure, where the important patterns to pick up on are local, not global.

To build a CNN architecture we need convolutional layers, pooling layers, and fully-connected layers. Convolutional layers consist of learnable filters that are convolved with local regions as they slide across the input volume [172]. Pooling layers provide coarse-grained spatial information and enforce invariance to small translations of the input. Standard CNNs typically stack several convolutional layers (with ReLU activation), followed by a pooling layer. Fully connected layers usually make up the final layer(s) of the network (the same type as in MLPs). There exist many variations of convolutional architectures and over the last decade the architectures have been iteratively improved upon to perform better on popular vision datasets. The two architectures that are most important for this thesis are VGG-Net and ResNet.

VGG. A VGG-Net[1] [256] is a type of convolutional architecture that consists of several blocks, each consisting of convolutional and pooling layers (see Figure 1.4). VGG-Net is important, because it illustrated the importance of depth to obtain good performance on the popular ImageNet Large Scale Visual Recognition Challenge [241] in 2014. The most commonly used VGG-Nets use either 16 (VGG-16) or 19 (VGG-19) layers. The use of more layers leads to vanishing/exploding gradient problems. This problem was addressed by residual neural networks.

Residual Neural Networks. A Residual Network (ResNet) [97] is a widely used deep convolutional architecture with batch normalization [121] (Section 1.1.2.2), which uses skip connections that allow information to skip intermediate layers (Figure 1.4). After AlexNet [147] popularized the use of deep convolutional neural networks in 2012 by lowering the classification error from 26% to 16% on the ImageNet Large Scale Visual Recognition Challenge [241], He et al. [97] were able to achieve a 3.56% error rate using a deep residual network three years later. ResNet architectures will appear various times throughout this thesis in the form of: ResNet-18 or ResNet-34, which means a 18-layer or 34-layer ResNet architecture, respectively.

Popular Vision Datasets. Commonly used datasets in computer vision are the MNIST [152], Fashion-MNIST [292], CIFAR-10 [146], CIFAR-100 [146], and ImageNet [52] datasets of increasing complexity and image size.[2] We will focus on the first four in this thesis: MNIST images ($60,000$ train, $10,000$ test) are 28×28 gray-scale images of hand-written numbers of 0-9 (10 classes). The Fashion-MNIST dataset has the exact same format as standard MNIST, but uses Zalando's article images in 10 classes (T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot) to increase the complexity of the data. CIFAR-10 consists of 60,000 ($50,000$ train, $10,000$ test) 32×32 color images in 10 classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks). CIFAR-100 equivalently has 100 classes, with 600 images per class. Examples are given in Figure 1.5.

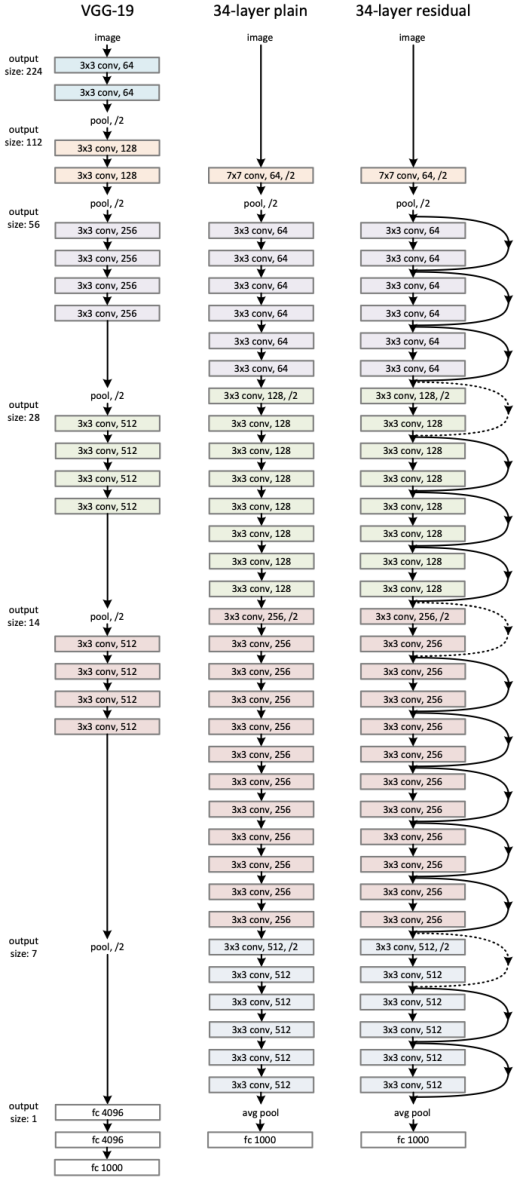

Figure 1.4: Figure adapted from He et al. [97]. Left: a VGG-19 model [256], middle: a 34-layer plain convolutional network, right: a 34-layer residual net, where dotted lines indicate shortcuts.

[1] Named after the research group at the University of Oxford who developed it, the "Visual Geometry Group".

[2] The MNIST acronym stands for the Modified National Institute of Standards and Technology dataset. The CIFAR-10 and CIFAR-100 datasets are named after the Canadian Institute For Advanced Research, where the number indicates the number of classes in the dataset.

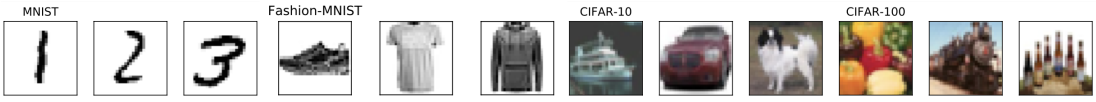

Figure 1.5: Example images from the MNIST [152], Fashion-MNIST [292], CIFAR-10 [146], and CIFAR-100 [146] datasets.

1.1.2.4 Recurrent Neural Networks and Long Short-Term Memory Cells

Recurrent Neural Networks (RNNs) are a useful tool for processing time-series data (for an introduction see e.g., [88]). In contrast with traditional neural networks they contain loops, which allow information from one step of the network to be passed on to the next step. Unfortunately, the memory that the network has of previous timesteps diminishes as the timesteps occurred further in the past. To resolve this, Long Short-Term Memory (LSTM) cells can be used, which were first introduced by Hochreiter and Schmidhuber [104]. LSTM cells contain additional gates, which control which memories are added to the cell state and which memories are no longer needed and can therefore be forgotten. LSTM cells allow the network to store information from far back in time and may therefore be more capable of capturing long-term behaviours.

1.1.2.5 Transformers

For Natural Language Processing (NLP) data and other sequential inputs the transformer architecture is widely used [273]. Transformers not only obtain large performance improvements compared to Recurrent Neural Networks on NLP applications, but are also more efficient to train. The Transformer uses a concept called self-attention to capture information between one element and every other element of a sequence. The encoder part of the Transformer consists of stacks of layer blocks that contain a multi-head self-attention part and a fully connected feedforward part both with residual connections [97] and layer normalization [11]. Before going through the encoder blocks, words are first transformed into vectors using learned word embeddings, i.e. using an embedding matrix, which associates each element in the transformer's vocabulary with an embedding that aims to capture both the semantic (meaning-based) and syntactic (grammar-based) relationships of the words.

BERT. A specific language representation model named DistilBERT appears in Chapter 5. BERT (Bidirectional Encoder Representations from Transformers) [53] is a Transformer architecture with several encoder blocks that uses bi-directional encoding. BERT masks part of the input and asks the model to predict the masked words, which provides additional context for a word in a sequence. BERT also introduces a new task that asks the model to predict whether a sentence is likely to follow another sentence to try to to improve the relationships between sentences. Especially due to the use of masking, BERT has found widespread success and is currently used for the Google search engine. In this thesis we will use DistilBERT, a distilled version of BERT obtained using knowledge distillation [35, 103]. For a small cost in performance, DistilBERT is 60% faster to train than BERT [246].

Vision Transformer. When Transformers recently became hugely popular for natural language processing, the community started to explore applications in other areas as well, in particular for images. Even though transformers do not possess any of the inductive biases that CNNs possess (locality and translational invariance), a paper by Dosovitskiy et al. [57] showed that a Vision Transformer architecture could achieve comparable performance as state-of-the-art CNN architectures on image data with significantly reduced cost. The caveat for this result is that large dataset sizes are required, i.e. when pre-training on the ImageNet Large Scale Visual Recognition Challenge dataset which has 1.3M images ResNet architectures still outperform Vision Transformers, but this changes for larger datasets (14M-300M images) [57]. There is a clear trend of larger dataset sizes and model sizes in the recent literature, this will be discussed more in Section 1.1.4.

CLIP. Finally, we want to highlight a breakthrough multimodal model named CLIP (Contrastive Language–Image Pre-training) that came out in 2021 [230]. CLIP is an image-text bi-encoder trained on a large corpus of image-text pairs. It bridges the fields of natural language processing and computer vision (it uses a Vision Transformer type model for the visual features). CLIP is capable of classifying images without requiring training the model on the specific dataset, and has been shown to possess remarkable generalization capabilities. Although not featured in this thesis, we recommend the reader keep an eye on this research direction.

1.1.3 Neural Network Training

Neural network training consists of updating the neural network parameters using specific update rules that involve the gradients of the loss with respect to the parameters, where the loss $\mathcal{L}(y, \hat{y})$ is a measure of discrepancy between the output of the neural network $\hat{y}$ and the true target y. The gradients are obtained through the backpropagation algorithm [239, 240]. To give a simple example of how the backpropagation algorithm works, consider a small feedforward network with a tanh activation for the input layer, no hidden layers, and no activation or bias for the output layer

$$\begin{aligned} \text{pass input } x \text{ through first layer:} \quad y' &= \tanh(W_1 x + B_1), \\ \hat{y} &= W_2 y'. \end{aligned}$$

The gradients of the loss with respect to the parameters are then given by

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W_2} &= \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} = \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} y', \\ \frac{\partial \mathcal{L}}{\partial W_1} &= \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y'} \frac{\partial y'}{\partial W_1} = \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} W_2 \frac{x}{\cosh^2(W_1 x + B_1)}, \\ \frac{\partial \mathcal{L}}{\partial B_1} &= \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y'} \frac{\partial y'}{\partial B_1} = \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} W_2 \frac{1}{\cosh^2(W_1 x + B_1)}, \end{aligned}$$

where, for example, for a MSE (mean square error) loss $\partial \mathcal{L}(y, \hat{y})/\partial \hat{y}$ is simply $2(\hat{y} - y)$. The specific update rule depends on the choice of optimizer. We describe relevant existing optimizers below. The model parameterization schemes contributed by this thesis are described in Chapters 3, 4, and 5.

1.1.3.1 Gradient Descent

The starting point for most neural network training schemes is a system of ordinary differential equations of the form

$$\mathrm{d}\theta = G(\theta)\mathrm{d}t, \tag{1.4}$$

where θ are the neural network parameters and $G(\theta)$ is the negative gradient of loss function $\mathcal{L}(\theta|\mathcal{D})$ defined in terms of the entire training dataset $\mathcal{D}$. The most common numerical method used for solving the system is the explicit Euler method

$$\theta_{n+1} = \theta_n + hG(\theta_n), \tag{1.5}$$

for a choice of discretization stepsize $h > 0$. This technique is called gradient descent (GD) [38]. For a sufficiently small stepsize h gradient descent will decrease the loss

$$\mathcal{L}(\theta_{n+1}) - \mathcal{L}(\theta_n) = \left(\mathcal{L}(\theta_n) + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta_n}(\theta_{n+1} - \theta_n) + \dots \right) - \mathcal{L}(\theta_n) = -h \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta_n} \right)^2 + \mathcal{O}(h^2) \tag{1.6}$$

as shown using a Taylor expansion and Eq. (1.5). Gradient descent finds local minima by iteratively updating the model parameters in the direction of the negative gradient of the loss. For rugged energy landscapes which exhibit many local minima (see e.g., Zwanzig's model of molecular diffusion on a rough energy landscape [314]), adding stochasticity in the form of a energetic fluctuations can aid to overcome energy barriers blocking movement between states.

The gradient descent method is sensitive to the choice of stepsize and the choice of initialization and can take exponential time to escape saddle points [59], which are common in high-dimensional non-convex optimization [49]. Further, it treats all directions, whether flat or steep, the same, which slows down convergence. Ideally one would use information from the Hessian (as in Newton's method [214]), but this is too computationally expensive and memory intensive for typical-sized neural networks. Finally, to compute the gradient one needs to sum over the full dataset for every gradient descent step, which becomes costly for bigger datasets. This is particularly undesirable for machine learning applications, because a large training dataset is typically required to obtain good generalization performance. These limitations of gradient descent motivated the development of other optimization schemes.

1.1.3.2 Stochastic Gradient Descent

In stochastic gradient descent [234, 137, 153, 29] one updates the parameters of the model using the negative gradient $G_i(\theta)$ corresponding to a single randomly selected data sample x_i (with corresponding label y_i) with index i

$$\theta_{n+1} = \theta_n + hG_i(\theta_n). \tag{1.7}$$

Each iteration is therefore cheap to compute, because the gradient no longer needs to be computed with the full N data points, which reduces the cost per parameter update from $\mathcal{O}(N)$ for gradient descent to $\mathcal{O}(1)$. For e.g. applications in vision, there is a lot of redundancy in the data, which renders using all of the data in each optimization step inefficient.

Approximating the gradient in this manner introduces stochasticity in the system. The stochastic gradient $G_i(\theta)$ is an unbiased estimate of the full gradient $G(\theta)$

$$\mathbb{E}[G_i(\theta)] = \frac{1}{N}\sum_{i=0}^{N} G_i(\theta) = G(\theta). \tag{1.8}$$

Although each direction $G_i(\theta_n)$ may not be a descent direction from θ_n, if it is a descent direction in expectation, then the method will still find a minimizer. Stochastic gradient descent is highly sensitive to the choice of stepsize. For small enough and slowly decreasing learning rate $\{h_n\}$ stochastic gradient descent exhibits similar convergence behaviour as gradient descent [149], but the use of small learning rates significantly slows down convergence.

Minibatch Stochastic Gradient Descent. A popular neural network training scheme is minibatch stochastic gradient descent (which we shall refer to as SGD), where the gradient is evaluated on a randomly selected subset of the training data [234]. This subset is called a minibatch and the subset size is called the *batch size*. A complete pass through the full training dataset is called an *epoch*. Typically, for every epoch the dataset is divided into randomly selected minibatches.

Consider a minibatch B of size $S < N$ and form an estimate of the gradient based on this subset

$$G_{(S)}(\theta) = -\frac{1}{S}\sum_{n\in B} \nabla_\theta \mathcal{L}(\theta, x_n). \tag{1.9}$$

The SGD parameter update is then given by

$$\theta_{n+1} = \theta_n + hG_{(S)}(\theta_n). \tag{1.10}$$

The use of minibatch stochastic gradient descent allows for some degree of parallelization (depending on the chosen batch size). Because the data samples in the minibatch are drawn uniformly at random from the full dataset, the expectation of the gradient is still the same, but the variance of the stochastic gradient estimates is reduced significantly compared to standard stochastic gradient descent. This makes the method less sensitive to the choice of stepsize and aligns the updates more with the full gradient. However, a new hyperparameter is introduced, the batch size. To set this hyperparameter, Goyal et al. [89] proposed a linear scaling rule,

which suggests multiplying the initial learning rate with a factor k if the minibatch size is k times larger. Although popular, this rule does not hold across all batch sizes [252], so should be used with caution. For larger batch sizes it does intuitively make sense that the learning rate should be increased, as you have a smaller number of iterations to update the weights. However, this appears suboptimal for generalization. The use of SGD with smaller batch sizes was empirically shown to lead to enhanced generalization performance compared to using SGD with large batch sizes or GD (in the moderate learning rate regime) [136, 125, 313, 290]. Small batch SGD noise is thought to cause regularization effects [259, 40] and to help with avoiding "sharp" minima [104, 136, 173, 313, 98]. Although subsequent work showed that sharp minima can also generalize well [55] (more on the disputed notion of "sharpness" in Section 1.1.4.1) and Hoffer et al. [107] found that in some settings the generalization gap between small-batch and large-batch settings can be closed by adjusting the learning rate and number of iterations, the generalization issue of large-batch training (in particular in the absence of a large initial learning rate) is still an active area of research. It formed the motivation for the development of the layer-wise training methods discussed in Section 1.1.3.5. It also motivated interesting work by Wen et al. [286], who proposed to modify large-batch SGD updates by injecting Gaussian noise with mean zero and a suitable covariance matrix. This is related to the Stochastic Gradient Langevin Dynamics method [285], which will be introduced in Section 2.4.1.

The remarkable generalization capabilities of SGD (even in the absence of explicit regularization) are attributed to its implicit bias (or implicit regularization) [210, 310]. We discuss the role of implicit regularization of training algorithms and its importance in generalization in Section 1.1.4.2.

1.1.3.3 Add Momentum

For neural network training gradient methods are typically enhanced with momentum [226, 209, 263]. Momentum serves as a memory of past directions and allows the algorithm to gain speed in the presence of stochasticity and when facing small but persistent gradients.

We consider the following update rule for SGD with momentum

$$p_{n+1} = \gamma p_n + hG_{(S)}(\theta_n), \tag{1.11a}$$
$$\theta_{n+1} = \theta_n + p_{n+1}, \tag{1.11b}$$

where momentum variable p is a running average of recent gradients and momentum hyperparameter γ sets the timescale for which memory is retained [88]. This clearly reduces to standard SGD, Eq. (1.10), when $\gamma = 0$. SGD with momentum has found widespread use for the training of neural networks [155, 263, 88, 30] and is available as the standard SGD implementation in PyTorch [219] and Tensorflow [1].

1.1.3.4 Adaptive Methods

There is a range of methods that take into account the second moment of the gradient to adaptively change the learning rate for each parameter throughout training, such as AdaGrad [62], RMSProp [269], and Adam [139]. AdaGrad (Adaptive Gradient algorithm) [62] scales the learning rates inversely proportional to the square root of the accumulated sum of squared gradients. This causes larger learning rates (and more progress) for parameters with smaller associated gradients. Although AdaGrad is well suited to obtain rapid convergence for convex functions, it can be too aggressive in reducing learning rates for deep learning applications [88]. RMSProp (Root Mean Square Propagation) [269] instead uses an exponentially weighted moving average of the square of the gradients, which discards distant history. RMSProp can be interpreted as performing a Hessian-based preconditioning using an approximated diagonal equilibration matrix [50, 119]. Adam (Adaptive moment estimation) [139] is currently one of the most popular methods for neural network training and builds upon RMSProp in combination with momentum. Adam stores moving averages as estimates of the first moment and the second raw moment of the gradient, which are used to adaptively update the effective learning rate for

different parameters. Adam has the update rule

$$m_{n+1} = \beta_1 m_n - (1-\beta_1) G_{(S)}(\theta_n), \tag{1.12a}$$

$$v_{n+1} = \beta_2 v_n + (1-\beta_2) G_{(S)}(\theta_n)^2, \tag{1.12b}$$

$$\hat{m}_{n+1} = m_{n+1}/(1-\beta_1^{n+1}), \tag{1.12c}$$

$$\hat{v}_{n+1} = v_{n+1}/(1-\beta_2^{n+1}), \tag{1.12d}$$

$$\theta_{n+1} = \theta_n - h\, \hat{m}_{n+1}/(\sqrt{\hat{v}_{n+1}} + \epsilon), \tag{1.12e}$$

where multiplication/division are performed element-wise, $\beta_1, \beta_2 \in [0,1)$ are exponential decay rates of the moving averages of the gradient and the squared gradient, respectively, and ϵ is a small constant to prevent divergences. Typical choices are: $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 10^{-8}$, which are often left unaltered. The moving averages m_n and v_n are initialized to be zero vectors. This leads to biased estimates of the moments, which the bias-correction (1.12c)-(1.12d), resulting in $\hat{m}_n$ and $\hat{v}_n$, aims to address. In contrast, RMSProp may have a bias early on in training for the second moment estimate [88]. When using Adam in combination with weight decay, it is recommended to use the AdamW method by Loshchilov and Hutter [183], which decouples weight decay from the gradient, i.e. we use (1.12) but the parameter update in (1.12e) becomes

$$\theta_{n+1} = \theta_n - h\left(\hat{m}_{n+1}/(\sqrt{\hat{v}_{n+1}} + \epsilon) + \beta_R \theta_n\right), \tag{1.13}$$

with regularization strength β_R.

Despite Adam's popularity and its rapid training time there has been empirical evidence of a generalization gap with SGD [135, 289, 186, 312]. Adam also needs to store the second moment, which increases the memory requirement. Although Adam remains popular for certain tasks (in particular for the training of Transformers (Section 1.1.2.5) [308]), SGD with momentum is typically still the method of choice for vision models, see e.g. [115, 254].

1.1.3.5 Layer-wise Methods

LARS (Layer-wise Adaptive Rate Scaling) [299, 300] and LAMB (Layer-wise Adaptive Moments optimizer for large Batch training) [301] are both layer-wise adaptive methods developed for large batch training settings (see the discussion in Section 1.1.3.2). The LARS paper [299] stated that the ratio between the L_2 norm of the weights and of the gradients strongly affects training stability and speed and varies significantly between layers. The layer that is most prone to instability limits the overall learning rate for the full model. The LARS algorithm therefore normalizes the learning rate per layer by this ratio before applying a gradient update in the manner of SGD with momentum (the resulting algorithm is a special case of Lafond et al. [150]). While LARS was shown to do well on the ImageNet dataset using convolutional architectures, it performed badly for attention-based models [300]. To address this the LAMB paper [301] combined the ideas from LARS and AdamW [183]. The LAMB algorithm is very similar to AdamW, where instead of Eq. (1.13), the update rule per layer i with gradients $G^{(i)}$ is

$$\theta_{n+1}^{(i)} = \theta_n^{(i)} - h \frac{\phi(\|\theta_n^{(i)}\|)}{\|r_{n+1}^{(i)} + \beta_R \theta_n^{(i)}\|}(r_{n+1}^{(i)} + \beta_R \theta_n^{(i)}), \tag{1.14}$$

where $r_{n+1} = \hat{m}_{n+1}/(\sqrt{\hat{v}_{n+1}} + \epsilon)$ and ϕ is a scaling function. Similar algorithms that utilize layer-wise gradient normalization were proposed by Ginsburg et al. [83] and Yu et al. [302]. Recent work by Nado et al. [204] claims that for certain settings the performance of LARS and LAMB for large batch size settings can be matched using standard optimizers without using any layer-wise normalization, but with extensive hyperparameter tuning of the standard optimizers. We will explore training algorithms that train different neural network layers differently throughout this thesis.

1.1.3.6 Transfer Learning

Finally, we want to discuss transfer learning, as this will form an important application for our work in Chapter 5. The domain called transfer learning involves the use of pre-trained deep neural networks as initialization [53, 298]. These pre-trained networks are readily available through popular machine learning libraries, such as PyTorch [219], and have been trained on large datasets, such as ImageNet for vision applications [116] or large text corpora for natural language processing [110]. As initialization one chooses a pre-trained network that was trained on data that is in some way related to the dataset you want to train on, e.g. a pre-trained ResNet-34 architecture [97] that was trained on ImageNet would be a good initialization if you want to train a ResNet-34 on CIFAR-10 data (see Section 1.1.2.3). Using such a pre-trained network as initialization for your model can significantly speed-up training and typically improves the generalization performance of your model [298, 98, 229].

For the training procedure one starts with the pre-trained model and removes any task-specific layers. Then (part of the) network is re-trained on the new target task. To accelerate training (and in low target-data scenarios to prevent overfitting) one often only re-trains (fine-tunes) the later layers of the network [110, 47]. The motivation behind this is that the later layers of a neural network typically capture more task-specific knowledge, while early layers capture more general features, which can be shared across tasks [298, 94, 213, 231]. However, "fine-tuning the whole network usually results in better performance" [174], so there exists a balance between the generalization performance and the computational cost of the fine-tuning procedure.

1.1.4 Generalization and Regularization

An important performance measure of a trained neural network is to analyze its capacity to generalize to unseen (test) data. Although a neural network can perform extremely well on the provided training data, it may easily have ended up in a configuration which does not generalize well. A gradual loss in generalization performance as the error on the training data decreases is referred to as overfitting. Several factors appear to influence the generalization capacity of a neural network, such as the number of parameters, architecture, initialization, learning rate, stopping criterion, and optimizer, and no clear consensus has been reached on how these concepts interplay with one-another. In this section we will focus on those aspects that are needed to place the contributions in this thesis into context. We will first introduce some complexity measures from statistical learning theory in Section 1.1.4.1. Unfortunately, these traditional complexity measures are unable to explain the enhanced generalization performance of overparameterized deep neural networks. Overfitting is traditionally dealt with using explicit regularization approaches (Section 1.1.4.2), but in practice overparameterized neural networks often perform well even in the absence of explicit regularization [210, 20]. This observation leads us to also introduce the concept of implicit regularization [210] in Section 1.1.4.2.

1.1.4.1 Complexity Measures

The "no free lunch" theorem postulates that there is no universal learner that will perform well for all problems [251]. For every learner there will be a task for which it fails, while another learner would have succeeded. Because there is no universal learner, one wants to limit the possible candidates (the hypothesis class), which can be interpreted as introducing some sort of prior to the system. In neural networks the hypothesis class corresponds to choosing a specific architecture and activation functions. Different hypotheses in the hypothesis class are then determined by different values of the neural network's weights. So the learning task is to find the best weights, given the neural network's architecture (the "prior").

There are two kinds of error: the approximation error and the estimation error [251]. The approximation error is determined by the expressive power of the chosen neural network configuration. The estimation error represents the difference between the weight configuration that would have been ideal (given this specific neural network architecture) and the weight configuration found by the learning algorithm. The estimation error therefore depends on the training dataset size, because to have a good estimator a certain amount of training data is required.

One can reduce the approximation error by allowing a wider class of possible candidates (i.e., a larger neural network size)[3], but this is traditionally expected to lead to overfitting, which increases the estimation error. There therefore exists a trade-off between these two errors.

Traditional complexity measures of the neural network's architecture are the VC-dimension (Vapnik–Chervonenkis dimension) [272] and the Rademacher complexity [19]. The VC-dimension is a measure of the capacity of the model and can be used to bound the estimation error. It is defined as the size of the largest set of points that one can still obtain zero training error on using the model. The VC-dimension depends on the neural network size. However, it does not depend on the data distribution, which means that the bounds obtained for the generalization error are very loose. Tighter bounds can be obtained using the Rademacher complexity, which measures the ability of the neural network's architecture to fit randomly chosen labels.

Role of Overparameterization. Zhang et al. [306] showed that these traditional complexity measures are incapable of explaining several features of the generalization behaviour of deep neural networks (DNNs). In particular, they showed that deep neural networks can easily fit random labels. Neural networks turn out to have such a high capacity that they can memorize the training data (hence ruling the VC-dimension ineffective as a generalization bound) and can obtain zero training error on random labels (when using an architecture that gave good generalization properties when training with real labels). Additionally, they find that explicit regularization techniques are unable to attenuate this phenomenon. Explicit regularization, which adds a parameter norm penalty term to the loss function of neural networks, is a standard approach to prevent overfitting. Regularization does reduce the Rademacher complexity, but is found to not necessarily affect the generalization error. The findings of Zhang et al. [306] therefore illustrate that the traditional complexity measures from statistical learning theory are insufficient to explain the good generalization capabilities of overparameterized DNNs.

Figure 1.6A shows the traditional notion from statistical learning theory that models with increasingly higher capacity at some point reach a critical threshold, after which they start to overfit and the generalization performance is reduced (also called the bias-variance trade-off). In this setting capacity can be controlled by the choice of model architecture or through the use of regularization. However, Figure 1.6B shows the reality for modern neural networks [21]. For modern neural networks increasing the capacity can actually further improve the generalization performance. This corresponds to widespread observations by practitioners that "bigger is better" [147, 265, 36]. In fact, recent work on neural scaling laws suggest that the returns to scale might continue for multiple more orders of magnitude than currently used, provided the dataset size is large enough [130].

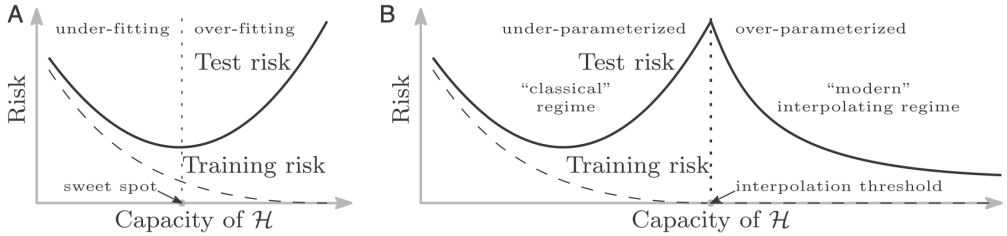

Figure 1.6: Figure adapted from [21]. $\mathcal{H}$ denotes the function class. Figure A shows the traditional notion that after a certain capacity is reached, larger models start to overfit and the performance on unseen test data is reduced. Figure B shows the double descent curve that holds for modern neural networks [21, 79, 198]. Despite intuitions from statistical learning theory, we see that the use of overparameterized neural networks can further improve the generalization performance of the model.

In recent years there has been an active search for an appropriate complexity measure that captures the different generalization properties exhibited by neural networks, which conflict with our traditional and (for neural networks) incorrect notion that the complexity and the

[3]Increasing the set of possible candidates can also be interpreted as making the prior more uninformative, which can be desirable for generalization performance.

number of parameters are inversely proportional [210]. One line of research is on norm-based complexity measures, which was first introduced by Bartlett [18], who proposed that the size of the weights is a more useful indicator of generalization ability than the number of weights. This motivates the use of e.g., weight decay (also known as L_2 regularization) as an explicit regularization technique, as this can be used to keep the weights small. Neyshabur et al. [212] proposed to combine this norm-based approach (see e.g., [211]), with the concept of expected sharpness. The notion of using the sharpness of obtained minima as a complexity measure for neural networks was most notably introduced by Keskar et al. [136] and Chaudhari et al. [41] and is determined by the eigenvalues of the Hessian matrix of the minimum. The idea is that flatter minima are more robust to small perturbations of the parameters and are therefore more generalizable. Additionally, flatter minima have lower description lengths (i.e., less precision is necessary to describe flatter minima, because moving slightly away from a flat minima does not result in much error), which is favorable according to the minimum description length principle [101, 105]. Flaws in the use of sharpness as a complexity measure were pointed out by Dinh et al. [55] and Neyshabur et al. [212], because sharpness is not a scale invariant measure. Neyshabur et al. [212] therefore propose to use the concept of "expected sharpness", which depends on the difference between the perturbed loss and the empirical loss, and combine this with the norm-based approach proposed by Bartlett [18] and others. However, this complexity measure is still incapable of explaining why overparameterized neural networks generalize well.

1.1.4.2 Regularization

As discussed above, overparameterized neural networks have such a high capacity [272] that they can easily memorize the training data and thus obtain zero training error on random labels [306], but such methods do not always generalize well to unseen test data. The traditional approach to mitigate overfitting is explicit regularization, where the loss is modified by adding a parameter norm penalty term. Examples are L_1 [268, 288] and L_2 [106] regularization. Although these explicit regularization techniques remain widely used, Neyshabur et al. [210] found that overparameterized neural networks can often perform well in absence of explicit regularization. We will therefore also discuss a different form of regularization, namely implicit regularization, which we define as regularization imposed by the training algorithm, without modifying the loss. Techniques such as dropout [102, 262] are thought to have both implicit and explicit regularization effects [284]. We will discuss these below.

Explicit Regularization. It is common practice to add a regularization term (a.k.a. a parameter norm penalty term) to the loss function of neural networks (see Eq. (1.3) and Figure 1.1), as this is expected to prevent overfitting and improve the neural network's ability to generalize [88]. Common types of regularization are: L_1 regularization, which attempts to minimise the sum of the absolute values of the network's parameters and encourages sparsity, and L_2 regularization, which minimises the sum of the square of the parameters [268, 288, 106]. The most common choice for the training of deep neural networks is L_2 regularization or weight decay, which in a Bayesian context corresponds to Maximum A Posteriori (MAP) Bayesian inference with a Gaussian weight prior [88].

The use of explicit regularization introduces an additional hyperparameter to tune: the regularization strength. The strength of the regularization term determines if the model will over/under-fit the data and therefore serves an important role. To illustrate the effect of L_2 regularization let us consider a basic example of polynomial regression, in which we attempt to obtain the best polynomial fit for a set of datapoints. A low degree polynomial will not fit the datapoints well enough, whereas a high degree polynomial will have the tendency to overfit. Overfitting will cause the obtained polynomial model to perform well on provided data, but poorly on unseen data. Regularization allows one to mediate these effects and to make the model more generalizable, as illustrated in Figure 1.7. In Figure 1.8 we show that higher values of the L_1 regularization strength cause minima corresponding to smaller neural network weight values to be favoured. For this example the potential $U(x)$ plays the role of the loss in neural networks and there is an additional L_1 regularization term $\beta_R|x|$. The potential originally has four equivalent minima, but as the regularization strength increases, minima closer to $x = 0$ are

favoured. However, if the regularization strength becomes too large, a new minima at $x = 0$ is introduced, which does not minimize the actual potential (a.k.a. loss).

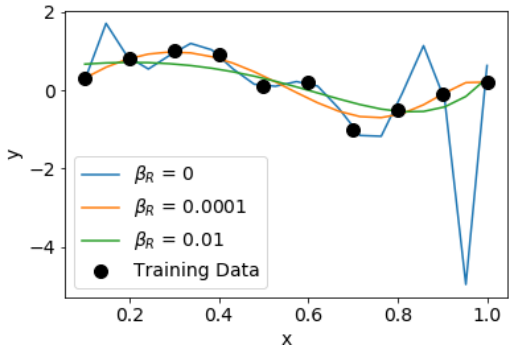

Figure 1.7: The effect of the L_2 regularization strength β_R on a polynomial regression problem using a 10th order polynomial. This figure was obtained by solving the normal equations solution $c = (x^T x + \beta_R \mathbf{I})^{-1} x^T y$, where c are the polynomial coefficients.

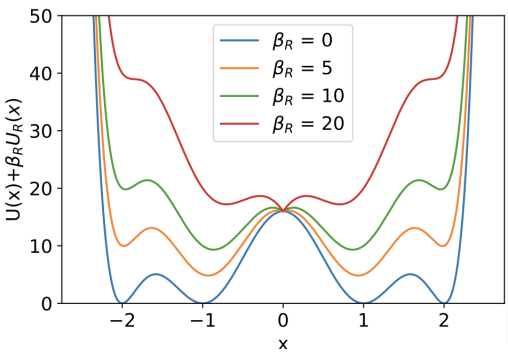

Figure 1.8: Effect of different L_1 regularization strengths β_R on $U(x) + \beta_R U_R(x)$ for potential $U(x) = (x-2)^2(x-1)^2(x+1)^2(x+2)^2$ and regularization term $\beta_R U_R(x)$ with $U_R = |x|$.

Implicit Regularization Implicit regularization is the type of regularization that is imposed not by modifying the objective, but by the choice of training algorithm [210]. It is widely believed that stochastic gradient descent outperforms standard gradient descent due to the implicit regularization effect imposed by the stochasticity of SGD (small-batch vs. large-batch training) [136]. Other examples of implicit regularization effects include the use of initial large learning rates [178], early stopping [296], and batch normalization [121]. Practitioners often use a combination of these techniques when training, but how all these different techniques work together to improve generalization performance is an ongoing area of research. For example, batch normalization is widely used, but explanations for the method's success remain elusive. Claims that it would reduce internal covariance shift [121] or smooth the loss landscape [247] have been disputed [247, 297]. Although batch normalization and dropout (see below) are both thought to cause a regularization effect and enhance generalization performance, their combination can actually reduce performance [187, 177, 42]. To offer a more transparent approach towards controlling parameter space, we propose a constained framework in Chapter 4 that allows us to draw a direct connection between the constraints imposed and the resulting generalization performance of the neural network.

Dropout. Dropout is an example of a technique that has both an explicit and implicit regularization effect [284]. The dropout technique was designed to combat overfitting a few years ago [102, 262] and is now commonly used for neural network training. For each training sample the dropout technique randomly sub-samples part of the network and temporarily ignores the neurons that were not selected, meaning that these neurons do not contribute to the forward pass and are not updated during the backward pass. Neurons are dropped out randomly with probability p and at the end of the procedure the values of the weights should be rescaled by the dropout rate. In this way dropout collects information from a whole ensemble of thinned neural networks, which all share many weights. Randomly switching off some weights at each iteration prevents the neural network from memorising the full training set (and thus prevents overfitting) and forces each of the weights to take more responsibility for the final prediction, which is thought to improve the generalization capacity of the neural network. Dropout modifies the expected loss (explicit regularization) and also adds noise to the gradient updates (implicit regularization) [284]. Gal and Ghahramani [74] provided a probabilistic interpretation of dropout and proposed a way to obtain an uncertainty measure for neural networks on which dropout was applied before each weight layer, a procedure also known as Monte-Carlo dropout. However, Osband [217] has argued that Monte-Carlo dropout actually measures the risk rather than the uncertainty, i.e., they showed that the measure computed by Monte-Carlo

dropout does not decrease with more data. We discuss Bayesian approaches to obtaining uncertainty estimates for neural networks in Section 1.3.4. In Chapter 5 we shall discuss ideas for developing a multirate version of dropout.

1.2 Molecular Dynamics

In this thesis we shall discuss several new approaches to neural network parameterization. For these, we often draw inspiration from molecular dynamics. We therefore provide an overview of the relevant concepts in molecular dynamics in this section.

Molecular dynamics (MD) models the movements of atoms, which are guided by Newton's equations of motion. In Section 1.2.1 we introduce the statistical ensembles we consider for molecular simulation. In Section 1.2.2 Hamiltonian dynamics is discussed, where the total energy of the system is conserved. Unfortunately, an analytical solution of the equations of motion is typically impossible, which requires MD simulations to obtain an approximate numerical solution. The vast number of particles in molecular systems leads to computational challenges. The accuracy of the numerical solution is dependent on the choice of time discretization, i.e. the stepsize. Further, there exists a timestep threshold above which the solutions become unstable. Systems frequently exhibit a separation of time scales, where the timestep threshold is determined by the fastest frequencies in the system. This therefore severely limits the computational efficiency of systems with multiple timescales. Multiple time-stepping strategies to address this are presented in Section 1.2.3. These formed part of the original inspiration for our work on multirate training of neural networks (Chapter 5). Alternatively, one can use constraints to increase the usable timestep of the full system (Section 1.2.4), which partly inspired our work on weight-constrained stochastic dynamics (presented in Chapter 4).

1.2.1 Statistical Ensembles

In this section we describe two statistical mechanical ensembles: the microcanonical ensemble and the canonical ensemble. The ensemble describes the circumstances for the "laboratory environment" of the molecular dynamics simulation. Typically, we fix both the number of particles N and the volume of the system V in which the particles can evolve.

For the **microcanonical ensemble** the energy E, alongside the number of particles N and the volume of the system V, is conserved. This ensemble is referred to as the NVE-ensemble. For such constant-energy, isolated systems purely deterministic Hamiltonian dynamics can be used, as discussed in Section 1.2.2.

To model a more realistic system which exhibits interactions with the ambient environment, we consider embedding the system of interest into a larger "heat bath". The weak coupling between the two systems allows for exchange of energy, although the energy remains constant on average. We assume the system is in thermal equilibrium so that the temperature is fixed. This **canonical** ensemble is known as the NV$\mathcal{T}$-ensemble, where the temperature τ (alongside N and V) is constant, but the energy is allowed to fluctuate. The positions q and momenta[4] p of the particles are described by the associated *Boltzmann-Gibbs distribution*

$$\rho = \frac{1}{Z} e^{-H(q,p)/\tau}, \text{ where } Z = \int_{\Omega} e^{-H(q,p)/\tau}, \tag{1.15}$$

where $H(q,p)$ is the Hamiltonian (the energy of the system), τ is the temperature, Ω is the phase space of the system (which represents all possible states), and Z is the partition function which normalizes the density and ensures that ρ is a probability measure. For the aim of sampling this distribution the use of stochastic differential equations arises (Chapter 2).

1.2.2 Hamiltonian Dynamics

For a system of N particles existing in d-dimensional space, where the particles have masses $m_i \in \mathbb{R}, i = 1, \ldots, N$, positions $q \in \mathbb{R}^{dN}$ and momenta $p \in \mathbb{R}^{dN}$, the evolution of the system is prescribed by Newton's equations of motion

[4]The momentum of a particle can be interpreted as its mass times velocity.

$$\begin{aligned}\frac{\mathrm{d}q}{\mathrm{d}t} &= \quad \nabla_p H(q,p) = M^{-1}p,\\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\nabla_q H(q,p) = -\nabla U(q),\end{aligned} \tag{1.16}$$

with initial conditions $(q(0), p(0)) = (q^0, p^0)$, mass matrix $M = \mathrm{diag}(m_1 I_d, \ldots, m_N I_d)$, identity matrix I_d, and Hamiltonian

$$H(q,p) = K(p) + U(q), \tag{1.17}$$

where the kinetic energy has been set to $K(p) = p^T M^{-1} p/2$ and $U(q) : \mathbb{R}^{dN} \to \mathbb{R}$ is the potential energy function, which describes interactions between particles. The force is described by the negative gradient of $U(q)$. Hamiltonian dynamics is time-reversible and the Hamiltonian (the total energy of the system) is conserved by the dynamics, i.e. using (1.16)

$$\frac{\mathrm{d}H(q,p)}{\mathrm{d}t} = \nabla_p H \cdot \frac{\mathrm{d}p}{\mathrm{d}t} + \nabla_q H \cdot \frac{\mathrm{d}q}{\mathrm{d}t} = 0. \tag{1.18}$$

The state of the system at any time instant is fully determined by the set of particle positions and momenta. System (1.16) can be rewritten as the system of ordinary differential equations

$$\frac{\mathrm{d}z}{\mathrm{d}t} = J\nabla H(z), \quad \text{with } z = (p,q) \text{ and } J = \begin{pmatrix} 0 & I_{dN} \\ -I_{dN} & 0 \end{pmatrix}, \tag{1.19}$$

where I_k denotes the $k \times k$ identity matrix [169]. A mapping $\phi : \mathbb{R}^{2dN} \to \mathbb{R}^{2dN}$ is symplectic if

$$\phi'(z)^T J \phi'(z) = J, \tag{1.20}$$

where ϕ' denotes the Jacobian matrix of ϕ. The flow map $\{\phi_t\}_{t\in\mathbb{R}}$ of a twice differentiable Hamiltonian system is symplectic for all $t \in \mathbb{R}$ [160].

Example 1.2.1. Consider a single particle with mass $M = 1$ attached to the origin through a linear spring with spring constant Ω^2. The potential energy is given by

$$U(q) = \frac{1}{2}\Omega^2 q^2. \tag{1.21}$$

Using Eq. (1.16) the dynamics are described by

$$\begin{aligned}\frac{\mathrm{d}q}{\mathrm{d}t} &= p,\\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\Omega^2 q,\end{aligned} \tag{1.22}$$

with initial conditions $(q(0), p(0)) = (q^0, p^0)$. The energy is conserved

$$H = \frac{1}{2}p(t)^2 + \frac{1}{2}\Omega^2 q(t)^2 = \frac{1}{2}\left(p^0\right)^2 + \frac{1}{2}\Omega^2\left(q^0\right)^2. \tag{1.23}$$

The solution of system (1.22) is

$$\begin{aligned}q(t) &= q^0 \cos(\Omega t) + \frac{p^0}{\Omega}\sin(\Omega t),\\ p(t) &= -q^0\Omega\,\sin(\Omega t) + p^0\cos(\Omega t).\end{aligned} \tag{1.24}$$

Unfortunately, obtaining an exact solution is typically not possible due to the non-linearity induced by considering more complex potential energy functions and the high-dimensionality

of the system. We will therefore briefly discuss numerical schemes for Hamiltonian systems, for which energy conservation is a primary concern.

1.2.2.1 Numerical Methods for Hamiltonian Dynamics

System (1.16) can be rewritten as

$$\mathrm{d}\begin{bmatrix} q \\ p \end{bmatrix} = \underbrace{\begin{bmatrix} M^{-1}p \\ 0 \end{bmatrix}\mathrm{d}t}_{A} + \underbrace{\begin{bmatrix} 0 \\ -\nabla U(q) \end{bmatrix}\mathrm{d}t}_{B}, \tag{1.25}$$

where parts A and B can be solved separately to generate flows $\mathcal{F}_t^A$ and $\mathcal{F}_t^B$. A popular symmetric second order method is "velocity Verlet" [274], which using $\mathcal{F}_{h/2}^B \circ \mathcal{F}_h^A \circ \mathcal{F}_{h/2}^B$ gives

$$p_{n+1/2} = p_n - \frac{h}{2}\nabla U(q_n), \tag{1.26a}$$

$$q_{n+1} = q_n + hM^{-1}p_{n+1/2}, \tag{1.26b}$$

$$p_{n+1} = p_{n+1/2} - \frac{h}{2}\nabla U(q_{n+1}). \tag{1.26c}$$

The Verlet scheme is symplectic and approximately conserves the energy. The positions obtained obey

$$M\frac{q_{n+1} - 2q_n + q_{n-1}}{h^2} = -\nabla U(q_n), \tag{1.27}$$

which is simply the finite difference approximation applied to Newton's second law (mass $\times$ acceleration = force): $M\frac{\mathrm{d}^2 q}{\mathrm{d}t^2} = -\nabla U(q(t))$. The computation of the forces $-\nabla U$ acting on each particle is a computational bottleneck for any MD simulation. Luckily, the Verlet scheme only requires a single evaluation of $\nabla U(q)$ per iteration (because $\nabla U(q_{n+1})$ computed in (1.26c) can be re-used at the next iteration).

1.2.2.2 Stability Threshold

Using the Verlet scheme (1.26) for the dynamics described by Eq. (1.22) gives

$$\begin{aligned} q_{n+1} &= q_n + hp_n - \frac{h^2}{2}\Omega^2 q_n = \left(1 - \frac{1}{2}h^2\Omega^2\right)q_n + hp_n, \\ p_{n+1} &= p_n - \frac{h}{2}\Omega^2 q_n - \frac{h}{2}\Omega^2 q_{n+1} = h\Omega^2\left(\frac{1}{4}h^2\Omega^2 - 1\right)q_n + \left(1 - \frac{1}{2}h^2\Omega^2\right)p_n. \end{aligned} \tag{1.28}$$

Or equivalently

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = A\begin{pmatrix} q_n \\ p_n \end{pmatrix}, \quad \text{where } A = \begin{pmatrix} 1 - \frac{1}{2}h^2\Omega^2 & h \\ h\Omega^2\left(\frac{1}{4}h^2\Omega^2 - 1\right) & 1 - \frac{1}{2}h^2\Omega^2 \end{pmatrix}. \tag{1.29}$$

The eigenvalues of A are

$$\lambda_\pm = \frac{1}{2}\left(2 - h^2\Omega^2 \pm h\Omega\sqrt{h^2\Omega^2 - 4}\right). \tag{1.30}$$

For $h\Omega > 2$ the eigenvalues are both real, with one outside of the unit circle. Under condition

$$h \leq \frac{2}{\Omega}, \tag{1.31}$$

the modulus of the eigenvalues is equal to 1 and both eigenvalues are on the unit circle. So condition (1.31) determines an upper bound for the timestep which can be used to maintain stability. This timestep threshold depends on the fastest frequency in the system, motivating the use of multiple time-stepping strategies for systems with multiple timescales.

1.2.3 Multiple Time-Stepping Methods

We will now discuss how one might address the timestep stability threshold for systems with multiple time scales, which are commonplace in molecular dynamics. The threshold is set by the fastest frequencies in the system, which severely limits the computational efficiency. Multiple time-stepping approaches therefore separate fast and slow degrees of freedom and integrate these using different stepsizes.

Many molecular dynamics applications can be described using the following Hamiltonian

$$H(q,p) = \underbrace{p^T M^{-1} p/2 + U_F(q)}_{H_F} + \underbrace{U_S(q)}_{H_S}, \tag{1.32}$$

with fast U_F and slow U_S potentials, and where evaluation of the slow force ∇U_S is typically computationally expensive [157]. The r-RESPA (reversible reference system propagator algorithm) [270, 271] scheme as described in Algorithm 1.1 is a popular technique for this setting.

Algorithm 1.1 r-RESPA Scheme

$p := p - (h/2)\nabla U_S(q)$
repeat ▷ k times
  $p := p - (h/(2k))\nabla U_F(q)$
  $q := q + (h/k)M^{-1}p$
  $p := p - (h/(2k))\nabla U_F(q)$
$p := p - (h/2)\nabla U_S(q)$

The r-RESPA method is similar in spirit to the Verlet algorithm (1.26) and is symplectic. Using r-RESPA the (typically) computationally expensive slow forces ∇U_S only need to be evaluated every large timestep h. Although presented here for two time scales for simplicity, the r-RESPA algorithm can easily be extended to more time scales [271]. Despite its popularity, the r-RESPA method suffers from resonance instabilities [26, 188]. We illustrate this with a simple example, which extends Example 1.2.1 to this setting.

Example 1.2.2. Consider a single particle with mass $M = 1$ attached to the origin through two springs. The Hamiltonian is described by Eq. (1.32) with fast and slow potentials

$$U_F(q) = \frac{1}{2}\Omega^2 q^2 \quad \text{and} \quad U_S(q) = \frac{1}{2}q^2, \tag{1.33}$$

where $\Omega \gg 1$. This gives for the fast dynamics the same system as in Example 1.2.1, namely Eq. (1.22). In this case this can be solved exactly and the solution of the fast system is

$$\begin{aligned} q(t) &= q^0 \cos(\Omega t) + \frac{p^0}{\Omega}\sin(\Omega t), \\ p(t) &= -q^0 \Omega \sin(\Omega t) + p^0 \cos(\Omega t). \end{aligned}$$

Using the flow map $S_{h/2}F_h S_{h/2}$ as in the r-RESPA method, where

$$S_h = \begin{bmatrix} 1 & 0 \\ -h & 1 \end{bmatrix} \text{ and } F_h = \begin{bmatrix} \cos(\Omega h) & \sin(\Omega h)/\Omega \\ -\Omega \sin(\Omega h) & \cos(\Omega h) \end{bmatrix}, \tag{1.34}$$

gives as update

$$\begin{pmatrix} q^1 \\ p^1 \end{pmatrix} = \begin{pmatrix} c - (h/(2\Omega))s & s/\Omega \\ -(h/2)(c - (h/(2\Omega))s) - s\Omega - (h/2)c & c - (h/(2\Omega))s \end{pmatrix} \begin{pmatrix} q^0 \\ p^0 \end{pmatrix}, \tag{1.35}$$

where $c = \cos(\Omega h)$ and $s = \sin(\Omega h)$. The sum of the eigenvalues is thus

$$\lambda_1 + \lambda_2 = 2\cos(\Omega h) - \frac{h}{\Omega}\sin(\Omega h). \tag{1.36}$$

Taylor expanding around $h = \frac{\pi}{\Omega}$ gives $-2 + \frac{\pi}{\Omega}\left(h - \frac{\pi}{\Omega}\right)$. The following condition

$$\left|-2 + \frac{\pi}{\Omega}\left(h - \frac{\pi}{\Omega}\right)\right| > 2 \tag{1.37}$$

holds if $h < \frac{\pi}{\Omega}$. This means that there is an instability in the neighborhood just below $h = \frac{\pi}{\Omega}$, because there the eigenvalues leave the unit circle. In fact, there turn out to be resonances at integer multiples of π/Ω, i.e. at $h = k\frac{\pi}{\Omega}$, where $k \in \mathbb{Z}_{>0}$. This is shown in Figure 1.9 for $\Omega = 3$. Although there exist stable regions to choose from for the timestep, these shrink as the timestep increases. Further, the resonance behaviour becomes very difficult to predict for nonlinear oscillators.

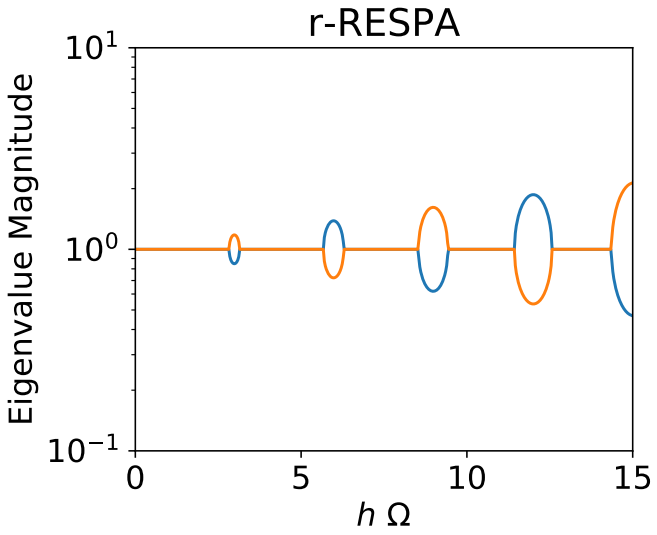

Figure 1.9: Magnitude of the eigenvalues as a function of $h\Omega$ for the r-RESPA method for Example 1.2.2, where we have set $\Omega = 3$.

These resonance effects complicate the use of r-RESPA and other multiple time-stepping methods [26]. There are strategies to overcome this, including the use of stochasticity (Section 1.2.3.1) or reversible averaging with linear drift (Section 1.2.3.2), which we will briefly discuss because they are loosely connected to the contributions of Chapter 5. Alternatively, the use of constraints can completely remove the fastest degrees of motion as discussed in Section 1.2.4.

1.2.3.1 Stochastic Multiple Time-stepping Methods

In this section we will explore whether the use of stochasticity can mitigate the resonance artefacts, which normally destabilize multiple timestepping methods at large timesteps. For this, we follow the discussion presented in Chapter 7 in [157]. Detailed knowledge of Langevin dynamics is not required to understand this section, but we refer the interested reader to Chapter 2 for more details, which is devoted in its entirety to stochastic dynamics.

Example 1.2.3. We consider the same fast and slow potentials as in Example 1.2.2 as described by Eq. (1.33), where again $\Omega \gg 1$. Although the slow dynamics remains the same, for the fast dynamics we shall now consider

$$\mathrm{d}q = p\ \mathrm{d}t, \tag{1.38a}$$

$$\mathrm{d}p = -\Omega^2 q\ \mathrm{d}t - \gamma p\ \mathrm{d}t + \sqrt{2\gamma\tau}\ \mathrm{d}W, \tag{1.38b}$$

where we use the Langevin dynamics system of stochastic differential equations where γ is the friction coefficient, W a vector of independent Wiener processes, and τ is the temperature. The update for the slow dynamics is again described by S_h in Eq. (1.34). Using a r-RESPA type

approach combined with the BAOAB scheme [156] (discussed in detail in Section 2.2.1.2) for the fast dynamics to simulate the dynamics gives

Algorithm 1.2 Langevin r-RESPA type scheme for Example 1.2.3

$p := p - (h/2)q$
repeat ▷ k times
  $p := p - (h/(2k))\Omega^2 q$
  $q := q + (h/(2k))p$
  $p := e^{-\gamma h/k} + \sqrt{\tau(1 - e^{-2\gamma h/k})}R$
  $q := q + (h/(2k))p$
  $p := p - (h/(2k))\Omega^2 q$
$p := p - (h/2)q$,

In Algorithm 1.2 we denote by R a vector of random variables with standard normal distribution. We can then compute the error in the value of q^2 averaged along the trajectory compared to

$$\begin{aligned}\langle q^2\rangle &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} q^2\rho\, dqdp = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} q^2 \frac{1}{Z} e^{-\left(\frac{1}{2}p^2+\frac{1}{2}q^2+\frac{1}{2}\Omega^2 q^2\right)/\tau} dqdp \\ &= \frac{\int_{-\infty}^{\infty} q^2 e^{-\frac{1}{2}q^2\left(1+\Omega^2\right)/\tau} dq}{\int_{-\infty}^{\infty} e^{-\frac{1}{2}q^2(1+\Omega^2)/\tau} dq} = \frac{\int_{-\infty}^{\infty} \frac{-q\tau}{(1+\Omega^2)}\, d\left(e^{-\frac{1}{2}q^2\left(1+\Omega^2\right)/\tau}\right)}{\int_{-\infty}^{\infty} e^{-\frac{1}{2}q^2(1+\Omega^2)/\tau} dq} = \frac{\tau}{(1+\Omega^2)},\end{aligned} \tag{1.39}$$

where $\langle q^2\rangle$ denotes the average of q^2, ρ is the Boltzmann-Gibbs distribution (Section 1.2.1), and we have used integration by parts for the final step. We present this error in Figure 1.10. The instability regions are indicated in white. Although larger values of the friction coefficient γ may slow down sampling, they can mitigate the presence of resonance artifacts.

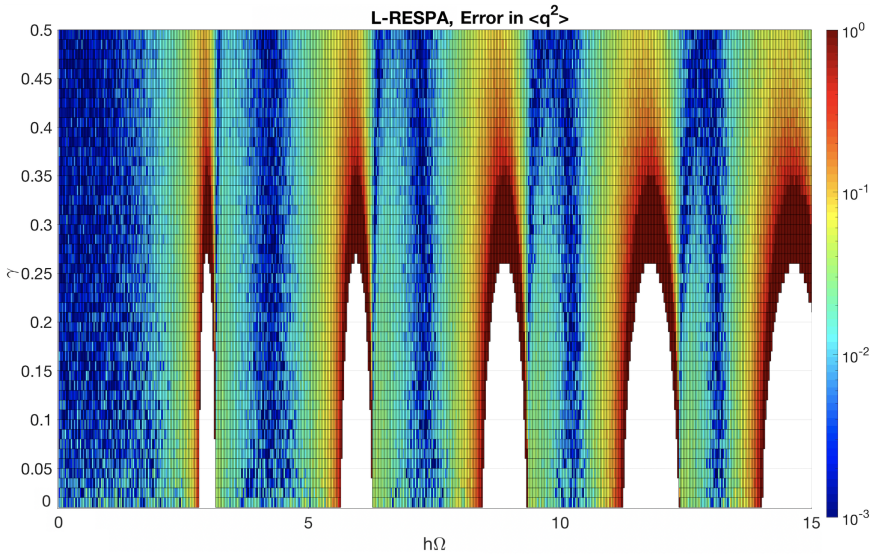

Figure 1.10: The error between $\langle q^2\rangle$ as obtained after running Algorithm 1.2 for 10^6 timesteps and when using Eq. (1.39) using $\tau = 1$ and $\Omega = 3$. Red regions have a high error, whereas blue regions have a low error. White regions indicate regions of instability. Larger values of γ mitigate the resonance artifacts. This mirrors the result obtained in Chapter 7 in [157].

1.2.3.2 Reversible Averaging

The reversible averaging scheme [159] can be used instead of the standard r-RESPA method (Algorithm 1.1) in settings where the variables themselves can be separated into fast (q, p) and slow (Q, P) parts. Although the reversible averaging method has a higher computational cost than standard r-RESPA, it remains comparable in terms of computational complexity if we assume the cost is dominated by the computation of the slow forces. Further, the reversible averaging method may lead to enhanced accuracy and stability compared to r-RESPA as shown in [159]. Here we focus on how it affects the resonance artefacts for a simple linear model (Example 1.2.4). There exist two variants of the reversible averaging scheme, which are presented in Algorithm 1.3 and 1.4. Although for the presentation of this algorithm the fast dynamics are approximated with a numerical approach, in certain cases the fast dynamics can be solved exactly (which is the case for Example 1.2.4).

Algorithm 1.3 Reversible Averaging Type 0

$P := P - (h/2)\mathrm{Av}_+[\nabla_Q U(Q, q_+)]$
$Q := Q + (h/2)M^{-1}P$
repeat ▷ k times
  $p := p - (h/(2k))\nabla_q U(Q, q)$
  $q := q + (h/k)m^{-1}p$
  $p := p - (h/(2k))\nabla_q U(Q, q)$
$Q := Q + (h/2)M^{-1}P$
$P := P - (h/2)\mathrm{Av}_-[\nabla_Q U(Q, q_-)]$

Algorithm 1.4 Reversible Averaging Type 1

$P := P - (h/2)\mathrm{Av}_+[\nabla_Q U(Q, q_+)]$
repeat ▷ k times
  $p := p - (h/(2k))\nabla_q U(Q, q)$
  $q := q + (h/k)m^{-1}p$
  $Q := Q + (h/k)M^{-1}P$
  $p := p - (h/(2k))\nabla_q U(Q, q)$
$P := P - (h/2)\mathrm{Av}_-[\nabla_Q U(Q, q_-)]$

We denote by $\mathrm{Av}_\pm[\nabla_Q U(Q, q_\pm]$ the averaged slow force over the considered time interval, where Q is kept fixed, whereas the fast variable is propagated either forwards (q_+, with all other variables fixed at their initial state) or backwards (q_-, with all other variables fixed at their final state) in time. There are various choices for performing this averaging. For the linear problem we consider here (Example 1.2.4) averaging the force is equivalent to averaging the fast position variable and substituting this into the force. The subject of our study is the difference between the two algorithms. Algorithm 1.3, which we shall denote as RA-0, keeps the slow variable Q fixed during the fast parameter update. In contrast, Algorithm 1.4 (denoted as RA-1) moves Q along a linear path determined by the fixed slow momenta P during the fast parameter update. Algorithm RA-1 has advantageous properties compared to RA-0, which formed part of the inspiration for our novel multirate method for neural network training as described in Chapter 5.

Example 1.2.4.

Consider the following linear model with Hamiltonian

$$H(Q, P, q, p) = \frac{1}{2}M^{-1}P^2 + \frac{1}{2}m^{-1}p^2 + \frac{1}{2}Q^2 + \frac{1}{2}k(Q - q)^2. \quad (1.40)$$

We shall set $M = 1$, $m = 1/\Omega^2$, and $k = \Omega^2$, where $\Omega \gg 1$, so that the system is clearly separated into fast and slow components, namely

$$H(Q, P, q, p) = \frac{1}{2}P^2 + \frac{1}{2}\Omega^2 p^2 + \underbrace{\frac{1}{2}Q^2}_{U_S(Q)} + \underbrace{\frac{1}{2}\Omega^2(Q - q)^2}_{U_F(Q,q)}. \quad (1.41)$$

Figure 1.11: System considered in Example 1.2.4.

For the fast variables one obtains

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \Omega^2 p,$$
$$\frac{\mathrm{d}p}{\mathrm{d}t} = \Omega^2(Q - q). \quad (1.42)$$

The exact solution of this system for fixed $Q = Q^0$ is

$$\begin{aligned} q(t) &= (q^0 - Q^0)\cos(\Omega^2 h) + p^0 \sin(\Omega^2 h) + Q^0, \\ p(t) &= (Q^0 - q^0)\sin(\Omega^2 h) + p^0 \cos(\Omega^2 h), \end{aligned} \tag{1.43}$$

so that initial conditions $q = q^0, p = p^0$ hold. The averaged fast position variable on $[0, h]$ then becomes

$$\begin{aligned} \overline{q} = \frac{1}{h}\int_0^h q(t)\mathrm{d}t &= \frac{1}{h}\left[\frac{1}{\Omega^2}(q^0 - Q^0)\sin(\Omega^2 t) - \frac{p^0}{\Omega^2}\cos(\Omega^2 t) + Q^0 t\right]_{t=0}^{t=h} \\ &= \frac{q^0 - Q^0}{\Omega^2 h}\sin(\Omega^2 h) + \frac{p^0}{\Omega^2 h}(1 - \cos(\Omega^2 h)) + Q^0. \end{aligned} \tag{1.44}$$

Now propagate the slow variables for a half timestep

$$\begin{aligned} Q^{1/2} &= Q^0 + \frac{1}{2}hP^{1/2} \\ P^{1/2} &= P^0 - \frac{h}{2}\nabla_Q U(Q^0, \overline{q}) = P^0 - \frac{h}{2}(Q^0 + \Omega^2(Q^0 - \overline{q})) \\ &= P^0 - \frac{h}{2}\left(Q^0 - \frac{q^0 - Q^0}{h}\sin(\Omega^2 h) - \frac{p^0}{h}(1 - \cos(\Omega^2 h))\right) \\ Q^1 &= Q^0 + hP^{1/2}. \end{aligned} \tag{1.45}$$

Now the fast variables have to be propagated. We will first consider scheme RA-0 (Algorithm 1.3), for which we keep Q fixed at $Q = Q^{1/2}$ during the fast parameter update. Then we obtain using (1.43)

$$\begin{aligned} q^1 &= (q^0 - Q^{1/2})\cos(\Omega^2 h) + p^0 \sin(\Omega^2 h) + Q^{1/2}, \\ p^1 &= (Q^{1/2} - q^0)\sin(\Omega^2 h) + p^0 \cos(\Omega^2 h). \end{aligned} \tag{1.46}$$

Now solve the fast system backward. The method is symmetric so we obtain

$$P^1 = P^{1/2} - \frac{h}{2}\left(Q^1 - \frac{q^1 - Q^1}{h}\sin(\Omega^2 h) + \frac{p^1}{h}(1 - \cos(\Omega^2 h))\right). \tag{1.47}$$

This gives as total update

$$\begin{pmatrix} Q^1 \\ q^1 \\ P^1 \\ p^1 \end{pmatrix} = A_{\text{RA-0}} \begin{pmatrix} Q^0 \\ q^0 \\ P^0 \\ p^0 \end{pmatrix}, \tag{1.48}$$

where

$$A_{\text{RA-0}} = \begin{pmatrix} 1 - \frac{h}{2}s - \frac{h^2}{2} & \frac{h}{2}s & h & \frac{h}{2}(1-c) \\ (1-c)\left(1 - \frac{1}{2}\left(\frac{h}{2}s + \frac{h^2}{2}\right)\right) & c + \frac{1}{2}(1-c)\mathcal{M}_{12} & \frac{1}{2}(1-c)\mathcal{M}_{13} & s + \frac{1}{2}(1-c)\mathcal{M}_{14} \\ \mathcal{M}_{31} & \mathcal{M}_{32} & \mathcal{M}_{33} & \mathcal{M}_{34} \\ s\left(1 - \frac{1}{2}\left(\frac{h}{2}s + \frac{h^2}{2}\right)\right) & -s + \frac{1}{2}s\mathcal{M}_{12} & \frac{1}{2}s\mathcal{M}_{13} & c + \frac{1}{2}s\mathcal{M}_{14} \end{pmatrix}$$

with notation $s = \sin(\Omega^2 h)$ and $c = \cos(\Omega^2 h)$ and

$$
\begin{aligned}
\mathcal{M}_{31} &= -\frac{1}{2}s - \frac{h}{2} - \mathcal{M}_{11}\left(\frac{h}{2} + \frac{1}{2}s\right) + \frac{1}{2}s\mathcal{M}_{21} - \frac{1}{2}(1-c)\mathcal{M}_{41}, \\
\mathcal{M}_{32} &= \frac{1}{2}s - \mathcal{M}_{12}\left(\frac{h}{2} + \frac{1}{2}s\right) + \frac{1}{2}s\mathcal{M}_{22} - \frac{1}{2}(1-c)\mathcal{M}_{42}, \\
\mathcal{M}_{33} &= 1 - \mathcal{M}_{13}\left(\frac{h}{2} + \frac{1}{2}s\right) + \frac{1}{2}s\mathcal{M}_{23} - \frac{1}{2}(1-c)\mathcal{M}_{43}, \\
\mathcal{M}_{34} &= \frac{1}{2}(1-c) - \mathcal{M}_{14}\left(\frac{h}{2} + \frac{1}{2}s\right) + \frac{1}{2}s\mathcal{M}_{24} - \frac{1}{2}(1-c)\mathcal{M}_{44}.
\end{aligned} \tag{1.49}
$$

The magnitude and argument of the corresponding eigenvalues are shown in Figure 1.12 and Figure 1.13, respectively. There are clearly resonance effects present.

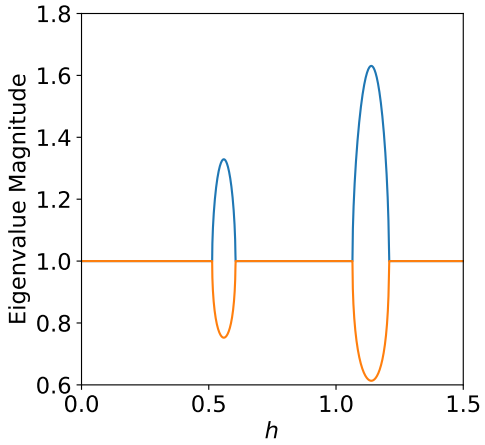

Figure 1.12: The magnitude of the eigenvalues of the RA-0 method (Algorithm 1.3) as a function of the stepsize h for $\Omega^2 = 10$.

A different option is to not keep Q fixed at $Q = Q^{1/2}$ during the evolution of the fast dynamics. One can instead use variant RA-1, as described in Algorithm 1.4, which pushes the slow variables during the fast movement along the path $Q := Q + hP^{1/2}$. This gives as update for Eq. (1.48)

$$
A_{\text{RA-1}} = \begin{pmatrix}
1 - \frac{h}{2}s - h^2/2 & \frac{h}{2}s & h & h(1-c)/2 \\
1 - c - \frac{1}{2}(h - s/\Omega^2)(h+s) & c + \frac{s}{2}(h - s/\Omega^2) & h - s/\Omega^2 & s + \frac{1-c}{2}(h - s/\Omega^2) \\
\mathcal{N}_{31} & \mathcal{N}_{32} & \mathcal{N}_{33} & \mathcal{N}_{34} \\
s - \frac{1}{2\Omega^2}(1-c)(h+s) & -s + \frac{1}{2\Omega^2}(1-c)s & (1-c)/\Omega^2 & c + \frac{1}{2\Omega^2}(1-c)^2
\end{pmatrix}
$$

where

$$
\begin{aligned}
\mathcal{N}_{31} &= -\frac{1}{2}s - \frac{h}{2} - \mathcal{N}_{11}(\frac{h}{2} + \frac{1}{2}s) + \frac{1}{2}s\mathcal{N}_{21} - \frac{1}{2}(1-c)\mathcal{N}_{41}, \\
\mathcal{N}_{32} &= \frac{1}{2}s - \mathcal{N}_{12}(\frac{h}{2} + \frac{1}{2}s) + \frac{1}{2}s\mathcal{N}_{22} - \frac{1}{2}(1-c)\mathcal{N}_{42}, \\
\mathcal{N}_{33} &= 1 - \mathcal{N}_{13}(\frac{h}{2} + \frac{1}{2}s) + \frac{1}{2}s\mathcal{N}_{23} - \frac{1}{2}(1-c)\mathcal{N}_{43}, \\
\mathcal{N}_{34} &= \frac{1}{2}(1-c) - \mathcal{N}_{14}(\frac{h}{2} + \frac{1}{2}s) + \frac{1}{2}s\mathcal{N}_{24} - \frac{1}{2}(1-c)\mathcal{N}_{44}.
\end{aligned} \tag{1.50}
$$

In this case the eigenvalue magnitudes are constantly 1, i.e. the resonances are removed. When we study the argument of the eigenvalues (Figure 1.14) we can see that indeed the eigenvalues do not cross (compared to Figure 1.13), which removes the instability.

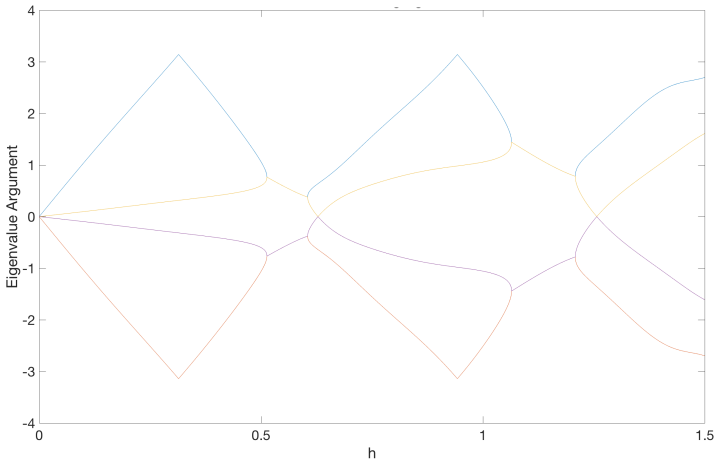

Figure 1.13: The argument of the eigenvalues of the RA-0 method (Algorithm 1.3) as a function of the stepsize h for $\Omega^2 = 10$.

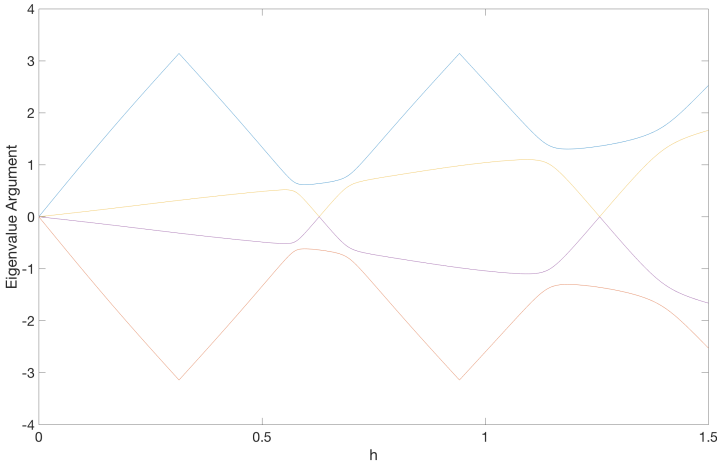

Figure 1.14: The argument of the eigenvalues of the RA-1 method (Algorithm 1.4) as a function of the stepsize h for $\Omega^2 = 10$.

1.2.4 Constrained Dynamics

The stepsize stability threshold (1.31) depends on the fastest frequency in the system. In molecular dynamics simulations the fastest degrees of freedom can sometimes be interpreted as the frequency of the stiff bond stretches. Hence, a more direct approach to address the stability threshold is to simply remove the fastest degrees of freedom using constraints as long as this preserves the properties of interest for the simulation [242, 37, 157]. The constraints we consider here are holonomic, which means that they only depend on the position, not the momentum.

The constrained equations of motions under constraint $g(q) = 0$, can be written as

$$\begin{aligned} \frac{\mathrm{d}q}{\mathrm{d}t} &= M^{-1}p, \\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\nabla U(q) - g'(q)^T\lambda, \\ 0 &= g(q), \end{aligned} \tag{1.51}$$

where λ represents a $\mathbb{R}^m$-valued vector of Lagrange multipliers. The associated constraint manifold is

$$\Sigma = \{q \in \mathbb{R}^{dN} \mid g(q) = 0\}, \quad g : \mathbb{R}^{dN} \to \mathbb{R}^m. \tag{1.52}$$

The co-tangency condition which need to be obeyed is

$$g'(q)\frac{\mathrm{d}q}{\mathrm{d}t} = g'(q)M^{-1}p \equiv 0. \tag{1.53}$$

The co-tangent bundle, denoted as $T^*\Sigma$ is then the set of (q, p) which obey the constraint $g(q) = 0$ and the co-tangency condition (1.53).

Example 1.2.5. Consider a single particle with mass $M = 1$ subject to potential $U(q)$ and a stiff restraint $U_{\text{stiff}} = \Omega(\|q\| - r_0)^2/2$, where $\Omega > 1$ and r_0 is the original distance between the particle and the origin [157]. Then imposing the constraint

$$g(q) = \|q\|^2 - r_0^2 = 0 \tag{1.54}$$

exactly fixes the distance between the particle and the origin for all times. The corresponding constrained system becomes

$$\begin{aligned} \frac{\mathrm{d}q}{\mathrm{d}t} &= p, \\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\nabla U(q) - q\lambda, \\ 0 &= \|q\|^2 - r_0^2. \end{aligned} \tag{1.55}$$

1.2.4.1 Numerical Methods with Constraints

Two commonly used algorithms for the constrained equations of motion (1.51) are SHAKE [242] and RATTLE [5]. These are both Verlet-like algorithms (Section 1.2.2.1). The main distinction between the two is that RATTLE preserves both the constraint and the co-tangency condition (1.53), whereas SHAKE only preserves the former. We will therefore only present the RATTLE scheme here

$$\begin{aligned} p_{n+1/2} &= p_n - (h/2)\nabla U(q_n) - g'(q_n)^T\lambda, \\ q_{n+1} &= q_n + hM^{-1}p_{n+1/2}, \\ 0 &= g(q_{n+1}), \\ p_{n+1} &= p_{n+1/2} - (h/2)\nabla U(q_{n+1}) - g'(q_{n+1})^T\mu, \\ 0 &= g'(q_{n+1})M^{-1}p_{n+1}, \end{aligned} \tag{1.56}$$

where λ and μ are the Lagrange multipliers related to the constraint $g(q) = 0$ and the co-tangency condition (1.53), respectively.

Here we have focused on the Hamiltonian case. In Section 2.5 we will discuss constrained stochastic differential equations (SDEs). In Chapter 4 we propose the use of discretized constrained SDEs for neural network training. The type of constraints we impose are naturally very different from the molecular dynamics setting and aim to address specific issues for neural network training.

1.3 Bayesian Framework and Sampling Methods

Although neural networks are expected to play an important role in science and policy decision making in the coming years, they are currently mostly used as a black box. The most commonly used neural networks are overconfident and can be easily fooled using adversarial attacks [86]. Bayesian inference techniques mitigate these problems by providing an uncertainty distribution over the parameter(s) of interest, rather than a single point estimate. This is especially important for practical situations in which researchers only have access to partial, noisy observations. Bayesian inference techniques are therefore widely used throughout the scientific community for the purpose of e.g., parameter estimation and data assimilation [232]. We introduce Bayes' rule in Section 1.3.1. Section 1.3.2 is dedicated to Markov Chain Monte Carlo methods. We then

discuss uncertainty quantification for machine learning (Section 1.3.3) and Bayesian Neural Networks (Section 1.3.4).

1.3.1 Bayes' Rule

Central to the theory of Bayesian inference is Bayes' rule, which allows us to update our knowledge about the parameters θ which we want to infer, using the provided data $\mathcal{D}$. Bayes' rule presents us with the posterior $p(\theta|\mathcal{D})$, a measure of our uncertainty about the parameters after observing the data, using the following equation

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta}, \tag{1.57}$$

where $p(\theta)$ is the prior distribution, which reflects our knowledge about θ before we see the data, and $p(\mathcal{D}|\theta)$ is the likelihood, which quantifies the likelihood that this data $\mathcal{D}$ could have been generated by the model with parameter values θ.

Example 1.3.1. To illustrate the use of Bayes' rule, let us use it to infer an unknown mean of the Gaussian distribution (see e.g., [17]). For independent and identically distributed (i.i.d.) data $\mathcal{D} = \{x_1, ..., x_N\}$, the likelihood is given by

$$p(\mathcal{D}|\mu, \sigma^2) = \prod_{i=1}^{N} p(x_i|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2},$$

which can be rewritten as $p(\mathcal{D}|\mu, \sigma^2) \propto \mathcal{N}\left(\frac{1}{N}\sum_{i=1}^{N} x_i \,\middle|\, \mu, \frac{\sigma^2}{N}\right)$. We choose the conjugate prior

$$p(\mu) \propto e^{-\frac{1}{2\sigma_0^2}(\mu_0-\mu)^2} \propto \mathcal{N}(\mu|\mu_0, \sigma_0).$$

The posterior is then proportional to

$$p(\mu|\mathcal{D}) \propto p(\mathcal{D}|\mu)p(\mu) \propto e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{1}{2\sigma_0^2}(\mu_0-\mu)^2}.$$

Following [17] one can then rewrite this expression to try to identify a Gaussian

$$p(\mu|\mathcal{D}) \propto e^{-\frac{1}{2}\left(\mu^2\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right) - 2\mu\left(\frac{\sum_{i=1}^{N} x_i}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right) + \frac{\sum_{i=1}^{N} x_i^2}{\sigma^2}+\frac{\mu_0^2}{\sigma_0^2}\right)} \propto e^{-\frac{1}{2\sigma_N^2}(\mu-\mu_N)^2},$$

where we defined $\sigma_N^2 = 1/\left(\frac{N}{\sigma^2} + \frac{1}{\sigma_0^2}\right)$ and $\mu_N = \sigma_N^2\left(\mu_0/\sigma_0^2 + \left(\sum_{i=1}^{N} x_i\right)/\sigma^2\right)$.

In Figure 1.15 we have plotted the prior and corresponding posterior distribution for the unknown mean of a Gaussian distribution, with true value $\mu = 4$.

In practice the denominator of Bayes' rule (1.57), is usually difficult/impossible to compute analytically. One can therefore not perform inference exactly, which motivates the use of Monte Carlo methods to draw samples from the posterior. Basic Monte Carlo methods require the proposal distribution to be easy to sample from and to be close to the target distribution, which is particularly difficult for high-dimensional spaces. In these kinds of scenarios Markov Chain Monte Carlo (MCMC) methods are more effective. Although basic Monte Carlo methods require the samples to be independent, MCMC methods generate a sequence of samples, where each sample depends on the sample preceding it.

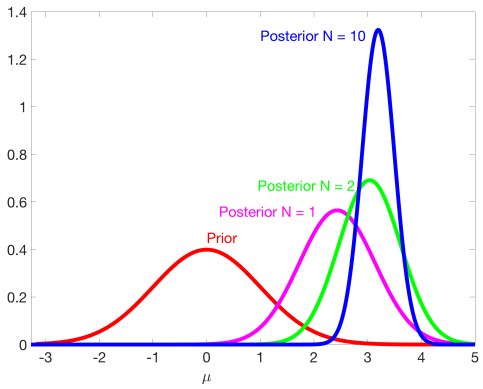

Figure 1.15: Prior and posterior distributions of a Gaussian mean for N data points generated with mean $\mu = 4$ and variance $\sigma^2 = 1$. The actual mean of the simulated dataset consisting of 10 data points was 3.5.

1.3.2 Markov Chain Monte Carlo Methods

In this section we provide a short introduction to Markov Chain Monte Carlo (MCMC) methods (see also e.g., [6] or [191]) and discuss the Metropolis-Hastings algorithm (Section 1.3.2.1), Hamiltonian Monte Carlo (Section 1.3.2.2), and simulated tempering (Section 1.3.2.3). We describe convergence diagnostics for MCMC samplers in Section 1.3.2.4.

A Markov chain is a stochastic process, which is fully specified by an initial probability distribution and a transition probability $T(x'; x)$, which describes the probability of moving from a state x to a state x'. The Markov property prescribes that the transition probability should only depend on the current state x, not on any of the previous states. The idea of MCMC methods is that one does not sample directly from the target distribution $p(x)$ (which is generally impossible) but from a different distribution, which in the large sample number limit should become effectively equivalent to sampling from the target distribution. To achieve this, the transition probability of the Markov chain must be chosen such that the target distribution is equal to its invariant distribution, i.e. $\forall x \in \mathbb{R}^d$

$$p(x') = \sum_x T(x'; x)p(x). \tag{1.58}$$

Additionally we require the chain to be ergodic, which means that regardless of which initial state we start off with, the chain converges to the invariant distribution. To obey the requirement of ergodicity, the chain should be irreducible (which means that we can get from any state to any other state in a finite number of steps) and aperiodic (which means that there are no infinite loops in the chain, i.e. the chain does not only return to certain states at periodic times). If the Markov chain is reversible, i.e. if its transition probability satisfies the detailed balance property

$$T(x'; x)p(x) = T(x; x')p(x') \quad \forall x, x', \tag{1.59}$$

then $p(x)$ is an invariant distribution under the Markov chain [6]. So (1.59) is a sufficient condition to choose appropriate transition probabilities of Markov chains for MCMC methods, but it is not necessary, i.e. one can use transition probabilities which do not satisfy (1.59). Note that for every target distribution there exist multiple transition probabilities which have this target distribution as their invariant distribution. Therefore, there are many different MCMC sampling methods available, where each method has its own pros and cons.

1.3.2.1 Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm [200, 95] iteratively proposes a new state x' using a proposal density $q(x'; x)$, which we can generate samples from (in contrast with the more complicated

target distribution p). The new state is accepted with acceptance probability

$$\alpha = \min\left\{1, \frac{p(x')q(x;x')}{p(x)q(x';x)}\right\}. \tag{1.60}$$

If the move is rejected, the chain remains at its current state. The state of the Markov chain at the end of each iteration is considered a sample of the target distribution (after some burn-in period). Note that the Metropolis-Hastings algorithm obeys the detailed balance property (1.59), i.e.

$$\begin{aligned} T(x';x)p(x) &= \min\left\{1, \frac{p(x')q(x;x')}{p(x)q(x';x)}\right\} q(x';x)p(x) = \min\left\{p(x)q(x';x), p(x')q(x;x')\right\} \\ &= \min\left\{\frac{p(x)q(x';x)}{p(x')q(x;x')}, 1\right\} q(x;x')p(x') = T(x;x')p(x'). \end{aligned}$$

So the target distribution p is an invariant distribution under the Markov Chain.

Some special cases of the Metropolis-Hastings algorithm are Gibbs sampling and the Metropolis-adjusted Langevin algorithm, which are briefly introduced below.

Gibbs Sampling. Consider a multivariate quantity $x \in \mathbb{R}^d$. During a single iteration the Gibbs sampling method [81] updates each element x_i using a conditional distribution $p(x_i|\{x_j\}_{j\neq i})$ (e.g., for 3 variables a single iteration samples $x_1' \sim p(x_1|x_2, x_3)$, $x_2' \sim p(x_2|x_1', x_3)$ and $x_3' \sim p(x_3|x_1', x_2')$). Gibbs sampling can be seen as a special case of the Metropolis-Hastings method, where the proposal distribution is now replaced by a sequence of conditional distributions. The Gibbs sampling method therefore also obeys the detailed balance property (1.59).

Metropolis-Adjusted Langevin Algorithm. The Metropolis-Adjusted Langevin Algorithm (MALA) [238, 236] is another special case of the Metropolis-Hastings method. For MALA the proposal distribution is obtained by an Euler-Maruyama discretization of over-damped Langevin dynamics, which will be introduced in Section 2.2. The use of a Metropolis-Hasting step ensures that the correct invariant distribution is maintained.

1.3.2.2 Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo (HMC) method (also known as hybrid Monte Carlo) [61, 206] uses gradient information to explore the state space faster than when using a simple random walk (for an introduction to HMC see e.g., [207] or [24]). The central idea to the HMC method is to introduce additional auxiliary 'momentum' variables p (one for each target variable). These momentum variables can be thought of as introducing kinetic energy $K(p)$ into the system, which in combination with the gradient information guides the sampler towards states of higher probability. More concretely, imagine we want to sample from

$$\mathbb{P}(\theta) = \frac{1}{Z} e^{-U(\theta)}, \tag{1.61}$$

for some potential energy function $U(\theta)$, partition function Z, and target variables θ, which can be viewed as 'positions'. The joint distribution when adding the momentum variables p can then be described in terms of the Hamiltonian $H(\theta, p) = U(\theta) + K(p)$ as follows

$$\mathbb{P}(\theta, p) = \frac{1}{Z_H} e^{-H(\theta,p)} = \frac{1}{Z_H} e^{-U(\theta)} e^{-K(p)}. \tag{1.62}$$

One can recover the target (1.61) by marginalizing out the momentum variables.

In the HMC method each iteration consists of two stages. In the first stage random values for the momentum variables are drawn from their distribution $e^{-K(p)}/Z_K$. In the second stage a Metropolis step to a new state (θ^*, p^*) is proposed by evolving the current state (θ, p) according

to the Hamiltonian dynamics equations (1.16)

$$\begin{aligned}\frac{\mathrm{d}\theta}{\mathrm{d}t} &= \quad \nabla_p H(\theta, p), \\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\nabla_\theta H(\theta, p),\end{aligned} \tag{1.63}$$

which, if we assume that the kinetic energy takes the form $K(p) = p^T p/2$, reduces to

$$\begin{aligned}\frac{\mathrm{d}\theta}{\mathrm{d}t} &= p, \\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\nabla_\theta U(\theta),\end{aligned} \tag{1.64}$$

where $U(\theta) = -\log\left[\mathbb{P}(\theta)\mathbb{P}(\mathcal{D}|\theta)\right]$ and $\mathcal{D}$ is the data. So the gradient information is transmitted through the auxiliary variables p, which determine the direction of motion of the target variables θ. A proposed state (θ^*, p^*) is obtained after executing Hamiltonian dynamics for a certain number of predefined steps. The chain will then move to this proposed state with probability

$$\min\left\{1, e^{-H(\theta^*,p^*)+H(\theta,p)}\right\}. \tag{1.65}$$

In practice, one must discretize the equations of Hamiltonian dynamics. This can be achieved using the Verlet algorithm (see Section 1.2.2.1). This introduces two new parameters into the computation, namely the stepsize h and the number of steps, which need to be tuned accordingly.

1.3.2.3 Simulated Tempering

In HMC the total energy is conserved. This property makes it very difficult for the algorithm to traverse large energy barriers, as we would observe in multimodal distributions [207]. This motivates the use of tempering methods to move between isolated nodes, which are separated by regions of low probability density. Tempering methods are used to more effectively explore the energy landscape by augmenting the state space with an auxiliary temperature variable [82, 194]. We consider the distribution

$$\mathbb{P}(\theta, p) = \frac{1}{Z} e^{-H(\theta,p)/\tau}, \tag{1.66}$$

with temperature τ. Information from different temperatures is used to improve the simulation of the system at the target temperature. The density is flattened at higher temperatures and energy barriers are lowered (see Figure 1.16). Therefore, the system is less likely to get stuck in local minima, as energy barriers are more easily crossed at higher temperatures. Care must be taken that the temperature differences are not too large, or otherwise the probability of a move between distributions at different temperatures is too low.

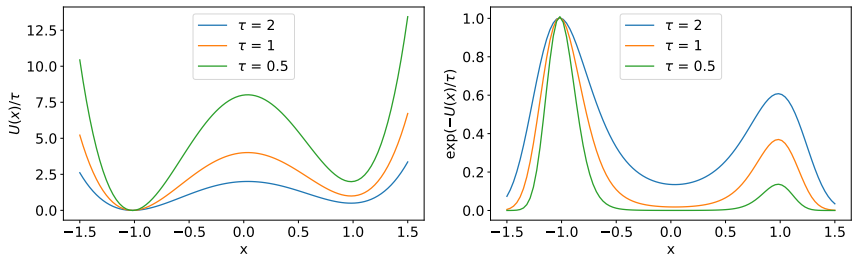

Figure 1.16: For uneven double well potential $U(x) = 3.5(x^2-1)^2 + 0.5(x+1)$, we have plotted $U(x)/\tau$ (left) and $e^{-U(x)/\tau}$ (right) as a function of x and for different temperature values τ.

1.3.2.4 Convergence

Convergence is an important issue that should be considered for all MCMC samplers. To obtain a rough estimate of the rate of convergence of the chain, one can use the integrated auto-correlation time (IAT) [23], which for a function of interest $f(x)$ is defined as

$$\mathbb{T}_f = 1 + 2\sum_{s=1}^{\infty} \rho_f^s, \tag{1.67}$$

where ρ_f^s is the lag-s autocorrelation function defined as $\rho_f^s = \frac{1}{\sigma_f^2}\text{Cov}(f_t, f_{t+s})$ and $\sigma_f^2 = \text{Var}[f]$ is the sample variance. A small value of the IAT indicates that the Markov chain mixes well. The IAT is closely related to the concept of the effective sample size (ESS), which for a total of N MCMC iterations has the following form

$$\text{ESS} = \frac{N}{\mathbb{T}_f}. \tag{1.68}$$

The ESS represents the number of independent samples of the posterior which have the same asymptotic variance as the N (usually correlated) MCMC samples. As one would expect, in the ideal case that all samples are independent, i.e. $\mathbb{T}_f = 1$, then ESS $= N$, i.e. the effective sample size is equal to the total amount of MCMC samples. A smaller value of the IAT corresponds to a larger ESS, which means that the MCMC sampler will be more efficient.

1.3.3 Uncertainty Quantification for Machine Learning

Popular types of modern neural networks have been found to be miscalibrated [93], i.e. events predicted to occur with probability p, do not occur p fractions of the time. Guo et al. [93] think that this miscalibration is closely connected to the increased model capacity and reduced use of weight decay regularization in modern neural networks. Neural networks are most notably also overconfident in predicting inputs that fall outside of the training dataset. This means that the probabilities that neural networks output can not be used as a measure of model confidence. A measure of model uncertainty is however of paramount important in real-world applications, such as self-driving cars, where a correct uncertainty estimate can significantly reduce the occurrence of accidents due to e.g., encountering new or unrecognized objects. Neural networks have also been shown to be particularly susceptible to adversarial attacks, which are small crafted perturbations to the inputs that cause neural networks to misclassify them with a high confidence [264, 86]. This also stresses the need for better uncertainty quantification techniques in neural networks.

There are two types of uncertainty in Bayesian modelling: aleatoric and epistemic uncertainty [142]. Aleatoric uncertainty is caused by noisy data, while epistemic uncertainty is a measure of the model uncertainty. Epistemic uncertainty can be improved by a larger training dataset (this typically makes the model more accurate), but aleatoric uncertainty cannot as it captures the inherent stochasticity of the data (although some data points can be noisier than others). Aleatoric uncertainty does not increase for out-of-training data points, whereas the epistemic uncertainty does increase [134]. Aleatoric uncertainty can be decomposed further into heteroscedastic aleatoric uncertainty, which depends on the data, and homoscedastic uncertainty, which does not depends on the data and can be seen as a task-dependent uncertainty. It is important to distinguish between these types of uncertainty as several methods only tackle one or the other [73]. The aleatoric and epistemic uncertainty combined together form the predictive uncertainty of the model.

1.3.4 Bayesian Neural Networks

Bayesian neural networks (BNNs) provide a way of obtaining uncertainty estimates and are thought to be more robust to overfitting than standard neural networks [189, 190, 206]. Bayesian methods automatically embody Occam's razor (which is closely linked to the minimum description length principle), by preferring simpler models over unnecessarily complex ones, which

is thought to be beneficial for the generalization capacity of the model. BNNs place a prior distribution on the neural network weights and learn a probability distribution, the posterior, using Bayes' theorem (1.57). Since we want our weights to be small and approximately centered around zero, a Gaussian prior $\mathcal{N}(0, \sigma^2 I)$ is a common choice. This gives

$$\mathbb{P}(\theta) \propto e^{-\frac{1}{2}\theta^T r_\theta \theta}, \tag{1.69}$$

where θ are the weights of the neural network, and r_θ is the precision (the reciprocal of the variance). Bayes' rule then allows us to define the desired posterior distribution, but this distribution is intractable. Inference is complicated due to the large amount of weights used in modern neural networks. To address this we will discuss the following two options in this section: Markov Chain Monte Carlo methods (Section 1.3.4.1) and variational inference (Section 1.3.4.2) [27]. The main advantage of MCMC methods is that in the asymptotic limit they sample from the true posterior, which is not the case for the variational inference technique. However, the computational cost of MCMC methods is significant. Variational inference is a more scalable solution and therefore received more attention from the machine learning community until recently.

1.3.4.1 MCMC Methods for BNNs

Izmailov et al. [123] use Hamiltonian Monte Carlo (see Section 1.3.2.2) to sample from the posterior over Bayesian neural network parameters. Recall that in HMC one samples from the distribution $\frac{1}{Z}e^{-H(\theta,p)}$, where $H = U(\theta) + K(p)$ is the Hamiltonian, consisting of a potential term U (dependent on parameters θ) and a kinetic term K (dependent on auxiliary "momentum" parameters p), which is often chosen to be $K = \frac{1}{2}p^T p$. By defining

$$U(\theta) = -\log\left(\mathbb{P}(\theta)\mathbb{P}(\mathcal{D}|\theta)\right) = -\log(\mathbb{P}(\mathcal{D}|\theta)) - \log(\mathbb{P}(\theta)), \tag{1.70}$$

the HMC method will effectively sample our desired posterior distribution, after marginalizing over the auxiliary variables p. In (1.70) the $-\log(\mathbb{P}(\mathcal{D}|\theta))$ term can be interpreted as the loss, and the $-\log(\mathbb{P}(\theta))$ term as the regularization because of the shape of the prior, e.g. for the prior as in (1.69)

$$-\log(\mathbb{P}(\theta)) = -\log\left(e^{-\frac{1}{2}r_\theta \|\theta\|_2^2}\right) = \frac{1}{2} r_\theta \|\theta\|_2^2, \tag{1.71}$$

which is reminiscent of L_2 regularization. Izmailov et al. [123] find that BNNs offer "the best results both in terms of uncertainty calibration and predictive accuracy." Although Izmailov et al. [123] used full-batch HMC as a precise tool to study BNNs posteriors, we note that this is impractical for modern neural network settings due to the huge associated computational requirements. Therefore, MCMC methods with stochastic minibatch gradients (SG-MCMC) are typically used to improve scalability [285, 43, 170, 215].

1.3.4.2 Variational Inference

Variational inference [101, 90, 27] proposes an approximate variational distribution q_φ, which is parameterized by φ. It then learns the parameters φ that minimize the Kullback-Leibler (KL) divergence [148] between this approximate distribution and the true posterior. The optimization objective is known as the variational free energy (which should be minimized) or, if one reverses the signs, as the evidence lower bound (ELBO) (which should be maximized), and which has a minimum description length interpretation [90]. There is a trade-off between using the approximate distribution to explain the data well versus matching the prior well. The prior is used to control the complexity of the learned function.

Stochastic variational inference applies SGD to variational inference to optimize the ELBO [108]. Blundell et al. [28] proposed an algorithm named Bayes by Backprop that is compatible with the backpropagation algorithm. It is commonly used in the machine learning literature due to its scalability to large neural networks. However, MCMC approaches still have the important advantage that they sample in the limit from the true posterior.

1.3.4.3 Cold Posteriors

Consider the tempered posterior distribution over the network parameters

$$p(\theta|\mathcal{D})^{1/\tau} \propto e^{-U(\theta)/\tau}. \tag{1.72}$$

In our 2019 paper on "Thermodynamic parameterization of neural networks" [163] (Chapter 3) we show the advantage of using $\tau \ll 1$ for neural network training. Subsequently, Wenzel et al., 2020 [287] rigorously studied this "cold posterior" effect for Bayesian deep learning using a similar approach, i.e. using Langevin dynamics with minibatch gradients and layerwise preconditioning, but using ensemble averages. They also found that the best test accuracy and predictive performance is obtained using cold posteriors, where $\tau \ll 1$ (Figure 1.17). Although Figure 1.17 shows a plateau for small τ, the specific behaviour at lower temperatures changes for different batchsizes and problem settings [287, 215], but consistently outperforms $\tau = 1$ for a wide range of settings [287]. Several papers have since tried to find the cause of this cold posterior phenomenon in the last two years [4, 123, 71, 215], but at writing time of this thesis no consensus has been reached. The importance of finding suitable priors for deep BNNs is however agreed upon.

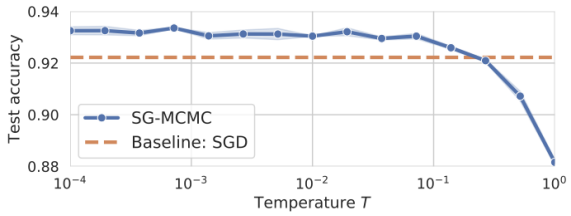

Figure 1.17: Figure was adapted from [287]. The generalization performance of a ResNet-20 architecture on CIFAR-10 data for different values of the temperature.

1.4 Original Contributions

The main contributions of this thesis are:

- The contents of Chapter 3 are based on joint work with Benedict Leimkuhler and Charles Matthews, which was published in [163]. A new section was added that shows further applications of the work and discusses related publications.

  In this work the loss gradient is embedded in a second order Langevin dynamics framework. The additive noise level plays the same role as the temperature in statistical physics and provides the connection between posterior sampling and pure MAP estimation. The work illustrates that using low temperatures in combination with partitioned integrators leads to enhanced diffusion on the loss landscape, increased robustness, and enhanced generalization performance of neural networks on certain classification tasks. Ergodic properties of the idealized stochastic differential equations associated with gradient schemes may help these methods ensure robust exploration of a useful range of parameters.

- The contents of Chapter 4 are based on joint work with Benedict Leimkuhler, Timothée Pouchon, and Amos Storkey, which was published in [165]. Further, a shorter workshop paper was accepted for the *Advances in Neural Information Processing Systems (NeurIPS) Optimization for Machine Learning (OptML) 2020 workshop* and received a best student paper award.

  The work uses constrained stochastic differential equations to train deep neural networks. Constraints provide direct control of the parameter space of a model, which allows for obtaining improved generalization performance. A general framework with accompanying discretization schemes is provided, which allows for user-defined constraints. In the work a statistical guarantee on the convergence of the training is provided, along with detailed implementation schemes for specific constraints –circle and orthogonality constraints–

and testing to show that the approach stands up to scrutiny. Whereas the effect of soft constraints, e.g. adding a penalty term to the loss, on the resulting generalization performance is difficult to measure, the hard constraints proposed in this paper are imposed directly on the parameters themselves, which allows us to directly study their effect on generalization.

- The contents of Chapter 5 are based on joint work with Benedict Leimkuhler, which is currently under submission. The full paper is available as an arXiv preprint [278].

  In this work we illustrate the presence of latent multiple time scales in deep learning applications and introduce the use of multirate techniques for neural network training. Our novel multirate scheme uses linear drift of the slow parameters during the fast parameter update. We analyze the convergence properties of our multirate scheme and draw a comparison with vanilla stochastic gradient descent. By choosing appropriate partitionings of the parameters into "fast" and "slow" parts we show that deep neural networks for transfer learning applications in vision and natural language processing can be trained in half the time, without losing generalization performance.

- The contents of Section 6.1 are based on joint work with Jonathan Frankle, which is currently under submission. The full paper is available as an arXiv preprint [277].

  We study the use of the loss along the linear path between the initial and final parameters of a neural network as a measure of the loss landscape. We show that caution is needed when using linear interpolation to make broader claims on the shape of the landscape and success of optimization. Further we explore the layer-wise sensitivity to the choice of initialization and optimizer hyperparameters and use this to design custom optimization schemes.

- The contents of Section 6.2 are based on a workshop paper, which appeared in [276].

  The work shows a limitation of using test accuracy as the only metric for model evaluation for classification problems by studying both human and network performance on a set of images that capture the transition from one class to another. The paper shows that models with the same test accuracy misclassify (and disagree on) different sets of images. A higher test accuracy corresponds with models becoming more unanimous in their label prediction, which may be undesirable for images that do not have a clear class label associated with them.

All numerical experiments were performed using Python and PyTorch [219]. For published papers the code has been made available on github.com/TiffanyVlaar. The code of papers currently under submission will be added to this repository upon acceptance.

Chapter 2

Background on Stochastics

The aim of this chapter is to provide the background on stochastics that is required to understand the contributions of this thesis. Section 2.1 covers basic concepts from stochastic processes, which can be found in e.g., [75, 131, 144, 220]. In Chapter 3 and 4 we will use stochastic differential equations (SDEs) as a mechanism of model parameterization. The family of SDEs we consider includes the overdamped and underdamped forms of Langevin dynamics (Section 2.2), as well as thermostat methods such as Adaptive Langevin dynamics (Section 2.3). We discuss how SDE-based schemes arise for the training of neural networks in Section 2.4. Finally, we discuss constrained SDEs in Section 2.5 to provide the foundations for our contributions in Chapter 4.

2.1 Basics of Stochastic Processes

Stochastic processes arise when modelling systems that are affected by random fluctuations. A well-known example is Brownian motion or Wiener process W_t, which is a real-valued stochastic process with almost surely continuous paths such that

1. $W_0 = 0$.
2. W has independent increments, so for $0 \leq s < t < u < v \leq T$, $W_t - W_s$ and $W_v - W_u$ are independent.
3. W has Gaussian increments, so $W_{t+h} - W_t$ is normally distributed with mean 0 and variance h.

The generalized derivative of Brownian motion is white noise process $\xi(t)$ with properties

$$\begin{aligned} \mathbb{E}[\xi_i(t)\xi_j(s)) &= \delta_{ij}\delta(t-s), \\ \mathbb{E}[\xi_i(t)] &= 0, \end{aligned} \tag{2.1}$$

where δ_{ij} is the Kronecker delta and $\delta(t-s)$ is the Dirac delta.

2.1.1 Stochastic Differential Equations (SDEs)

Consider stochastic differential equation

$$\mathrm{d}X(t) = b(X(t), t)\mathrm{d}t + \sigma(X(t), t)\mathrm{d}W, \quad X(0) = x, \tag{2.2}$$

for $X \in \mathbb{R}^d$, $t \in [0, T]$, measurable functions $b : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : [0, T] \times \mathbb{R}^d \to \mathbb{R}^{d \times n}$, and vector of n independent Wiener processes W_t. The stochastic integral equation corresponding to SDE (2.2) is

$$X(t) = x + \int_0^t b(X(s), s)\mathrm{d}s + \int_0^t \sigma(X(s), s)\mathrm{d}W(s). \tag{2.3}$$

The Itô stochastic integral is defined as

$$\int_0^t f(s)\mathrm{d}W(s) = \lim_{K\to\infty} \sum_{k=0}^{K-1} f(t_k)\left(W(t_{k+1}) - W(t_k)\right), \tag{2.4}$$

where f is adapted to the filtration $\{\mathcal{F}_t\}$ generated by one-dimensional Wiener process $W(t)$. Further, we assume that f is square integrable, i.e. $\mathbb{E}\left[\int_0^T f(s)^2\mathrm{d}s\right] < \infty$. Then following [220] the stochastic integral satisfies the Itô isometry property

$$\mathbb{E}\left[\left(\int_0^T f(s)\mathrm{d}W(s)\right)^2\right] = \int_0^T \mathbb{E}|f(s)|^2\mathrm{d}s \tag{2.5}$$

and is a martingale, so that

$$\mathbb{E}\left[\int_0^t f(s)\mathrm{d}W(s)\right] = 0 \;\text{ and }\; \mathbb{E}\left[\int_0^t f(u)\mathrm{d}W(u)|\mathcal{F}_s\right] = \int_0^s f(u)\mathrm{d}W(u) \;\; \forall t \geq s. \tag{2.6}$$

For n-dim. $W(t)$ and $f(t) \in \mathbb{R}^{d\times n}$ for each $t > 0$, the Itô isometry (2.5) can be generalized to

$$\mathbb{E}\left[\left(\int_0^t f(s)\mathrm{d}W(s)\right)^2\right] = \int_0^t \mathbb{E}|f(s)|_F^2 \,\mathrm{d}s, \tag{2.7}$$

with Frobenius norm $|f(s)|_F = \sqrt{Tr(f(s)^T f(s))}$ [220].

2.1.2 Kolmogorov Equations

Consider the autonomous SDE

$$dx_t = b(x_t)\mathrm{d}t + \sigma(x_t)\mathrm{d}W_t, \tag{2.8}$$

$t \in [0, T]$. Let us now obtain the associated generator. For solution x_t of SDE (2.8) and continuously twice-differentiable function $\varphi : \mathbb{R}^n \to \mathbb{R}$, Itô's formula gives [220]

$$\mathrm{d}\varphi(x_t) = \left(b(x_t)\cdot\nabla\varphi(x_t) + \frac{1}{2}\sigma(x_t)\sigma(x_t)^T : \nabla^2\varphi(x_t)\right)\mathrm{d}t + (\nabla\varphi(x_t))^T\sigma(x_t)\mathrm{d}W, \tag{2.9}$$

$$= \mathcal{L}\varphi(x_t)\mathrm{d}t + (\nabla\varphi(x_t))^T\sigma(x_t)\mathrm{d}W, \tag{2.10}$$

where $\nabla^2\phi$ is the Hessian matrix, symbol : denotes the Frobenius inner product, and $\mathcal{L}$ is

$$\mathcal{L} = b(x)\cdot\nabla + \frac{1}{2}\sigma(x)\sigma(x)^T : \nabla^2 \tag{2.11}$$

so that for initial condition $x_0 = x$

$$[from\ Itô's\ formula] \qquad \varphi(x_t) - \varphi(x) = \int_0^t \mathcal{L}\varphi\mathrm{d}s + \int_0^t (\nabla\varphi(x))^T\sigma(x)\mathrm{d}W(s), \tag{2.12}$$

$$[taking\ expectations] \qquad \mathbb{E}^x[\varphi(x_t)] - \varphi(x) = \int_0^t \mathbb{E}^x[\mathcal{L}\varphi]\mathrm{d}s, \tag{2.13}$$

$$[taking\ the\ limit] \qquad \lim_{t\to 0}\frac{\mathbb{E}^x[\varphi(x_t)] - \varphi(x)}{t} = \mathcal{L}\varphi(x), \tag{2.14}$$

where expectations are taken with respect to all Brownian paths, we assume that the expectation and the generator commute, and Eq. (2.13) follows from the martingale property (2.6). So $\mathcal{L}$ as defined in (2.11) is the generator of SDE (2.8).

The L^2 adjoint of (2.11) (also known as the Fokker-Planck operator) denoted as $\mathcal{L}^*$ can be

obtained using

$$\int \mathcal{L}fh \, \mathrm{d}x = \int f\mathcal{L}^* h \, \mathrm{d}x, \tag{2.15}$$

for continuously twice-differentiable functions f, h that decay sufficiently fast at infinity. This gives when integrating by parts twice

$$\begin{aligned}
\int \mathcal{L}fh \, \mathrm{d}x &= \int \left[(b(x) \cdot \nabla f)\, h + \left(\frac{1}{2}\sigma(x)\sigma(x)^T : \nabla^2 f \right) h \right] \mathrm{d}x, \\
&= \int f \nabla \cdot \left(-b(x) \cdot h + \frac{1}{2} \nabla \cdot (\sigma(x)\sigma(x)^T \cdot h) \right) \mathrm{d}x, \\
&= \int f \mathcal{L}^* h \, \mathrm{d}x.
\end{aligned}$$

Therefore, the Fokker-Planck operator associated with SDE (2.8) is

$$\mathcal{L}^* = \nabla \cdot \left(-b(x) \cdot + \frac{1}{2} \nabla \cdot (\sigma(x)\sigma(x)^T \cdot) \right). \tag{2.16}$$

Now we are going to present two equations of interest: the backward and forward Kolmogorov equations. Define $u = \mathbb{E}^x[\varphi(x_t)]$. Then the backward Kolmogorov equation

$$\frac{\partial u}{\partial t} = \mathcal{L}u \tag{2.17}$$

with $u(x, 0) = \varphi(x)$, can be directly obtained from Eq. (2.14) using the Markov property. The Fokker-Planck equation (also known as the forward Kolmogorov equation) is given by

$$\frac{\partial \rho}{\partial t} = \mathcal{L}^* \rho \tag{2.18}$$

with $\rho(0) = \rho_0$ [220]. The distribution ρ is a stationary distribution for the stochastic process associated with Fokker-Planck operator $\mathcal{L}^*$ if and only if

$$\mathcal{L}^* \rho = 0. \tag{2.19}$$

Example 2.1.1 (Ornstein-Uhlenbeck process)**.** The Ornstein-Uhlenbeck SDE is given by

$$\mathrm{d}p(t) = -\gamma p(t) \, \mathrm{d}t + \sqrt{2\sigma} \, \mathrm{d}W_t, \tag{2.20}$$

with $p \in \mathbb{R}^d$ and constant $\gamma, \sigma > 0$. Observe that $\mathrm{d}(e^{\gamma t} p) = e^{\gamma t}(\mathrm{d}p + \gamma p \, \mathrm{d}t)$, so by using the Itô's formula applied to the process $e^{\gamma t} p(t)$ we obtain

$$\begin{aligned}
&\mathrm{d}(e^{\gamma t} p) = e^{\gamma t} \sqrt{2\sigma} \, \mathrm{d}W_t, \\
\Rightarrow \;& e^{\gamma t} p(t) = p(0) + \int_0^t e^{\gamma s} \sqrt{2\sigma} \, \mathrm{d}W(s), \\
\Rightarrow \;& p(t) = e^{-\gamma t} p(0) + \int_0^t e^{\gamma(s-t)} \sqrt{2\sigma} \, \mathrm{d}W(s).
\end{aligned} \tag{2.21}$$

We will now use the Itô isometry property (2.5). The stochastic integral $Y(t) = \int_0^t e^{\gamma s} \mathrm{d}W(s)$ is a normally distributed random variable with mean zero and variance (from (2.5))

$$\mathrm{Var}(Y(t)) = \int_0^t e^{2\gamma s} \, \mathrm{d}s = \frac{1}{2\gamma}(e^{2\gamma t} - 1). \tag{2.22}$$

This gives as solution, since the equation has an unique solution in the probability sense, the

distribution

$$p(t) = e^{-\gamma t}p(0) + e^{-\gamma t}\sqrt{2\sigma}\frac{1}{\sqrt{2\gamma}}\sqrt{e^{2\gamma t}-1}R(t),$$
$$= e^{-\gamma t}p(0) + \sqrt{(1-e^{-2\gamma t})\sigma/\gamma}R(t), \tag{2.23}$$

where $R(t)$ should be interpreted as a random process which at every time t evaluates to a normal random variable, i.e. $R(t) \sim \mathcal{N}(0,1)$. We will use expression (2.23) when building discretization schemes for the underdamped Langevin dynamics SDE in Section 2.2.1.2, for a specific choice of $\sigma = \gamma\tau$, where τ is the temperature hyperparameter. To motivate this choice of σ, let us study the stationary distribution of the Ornstein-Uhlenbeck SDE for both a general σ and the specific case.

The generator of (2.20) is

$$\mathcal{L}_{\mathrm{OU}} = -\gamma p \cdot \nabla + \sigma\nabla^2. \tag{2.24}$$

The Fokker-Planck operator is

$$\mathcal{L}^*_{\mathrm{OU}} = \nabla \cdot (\gamma p \cdot + \sigma\nabla\cdot). \tag{2.25}$$

A stationary distribution for the Ornstein-Uhlenbeck SDE (2.20) is given by

$$\rho(p) \propto \exp(-\gamma p^T\sigma^{-1}p/2), \tag{2.26}$$

because

$$\mathcal{L}^*_{\mathrm{OU}}\rho(p) \propto \gamma\left(1-\gamma\sigma^{-1}p^2 - (1-\gamma\sigma^{-1}p^2)\right)\rho(p) = 0. \tag{2.27}$$

For specific choice $\sigma = \gamma\tau$ we recover canonical momentum distribution $\rho_{m_\tau}(p)$ (see Section 1.2.1 for $H(q,p) = p^Tp/2 + U(q)$, with identity mass matrix), given by

$$\rho_{m_\tau}(p) \propto \exp(-p^Tp/2\tau), \tag{2.28}$$

which is a stationary distribution for (2.20) with $\sigma = \gamma\tau$, i.e.

$$\mathcal{L}^*_{\mathrm{OU}}\rho_{m_\tau}(p) = 0. \tag{2.29}$$

2.1.3 Ergodicity

Dynamics are called ergodic if the average properties along the trajectories in the long-time limit converge to averages with respect to the probability measure of interest. A detailed discussion on ergodicity, including Theorem 2.1.1 can be found in [169] and [143].

Theorem 2.1.1. Assume that there exist a (strong) solution for the time-homogeneous SDE

$$\mathrm{d}x = b(x)\mathrm{d}t + \sigma(x)\mathrm{d}W_t, \tag{2.30}$$

at all times with b, σ sufficiently smooth functions. Then the generator can be written as

$$\mathcal{L} = A_0 + \frac{1}{2}\sum_{k=1}^{n} A_k^2, \tag{2.31}$$

$$\text{with } A_0 := \sum_{i=1}^{d} b^i\partial_i - \frac{1}{2}\sum_{i,j=1}^{d}\sum_{k=1}^{n}\sigma^{i,k}\partial_i(\sigma^{j,k}\partial_j) \text{ and } A_k := \sum_{i=1}^{d}\sigma^{i,k}\partial_i \text{ for } k = 1,\dots,n. \tag{2.32}$$

Assume that the Hörmander condition is verified, i.e. the Lie algebra generated by $(A_0,\dots,A_n)$ equals $\mathrm{Span}(\partial_1,\dots,\partial_d)$ [117, 118, 169]. Then the operator $\mathcal{L}$ is hypoelliptic. Given a stationary distribution π, which has a positive density with respect to the Lebesgue measure, then

irreducibility follows. From this pathwise ergodicity is obtained for any initial condition and bounded measurable observable φ

$$\lim_{T\to\infty}\frac{1}{T}\int_0^T \varphi(X)\ \mathrm{d}t = \int_\Omega \varphi\ \mathrm{d}\pi,\ \text{almost surely,} \tag{2.33}$$

where Ω is the state space.

2.2 Langevin Dynamics

For molecular dynamics we extensively discussed the use of constant energy, deterministic Hamiltonian dynamics simulations (Section 1.2.2), which is used for modelling the microcanonical ensemble. To model a more realistic setting one can consider the canonical ensemble, where temperature is conserved (Section 1.2.1). In this context we shall introduce the Langevin dynamics system of SDEs

$$\mathrm{d}q = M^{-1}p\,\mathrm{d}t, \tag{2.34a}$$
$$\mathrm{d}p = -\nabla_q U(q)\,\mathrm{d}t - \Gamma p\,\mathrm{d}t + \Sigma\,\mathrm{dW}_t, \tag{2.34b}$$

where $p_i \in \mathbb{R}^d, i = 1, ..., N$, is the momentum vector (for a system with N particles), $q_i \in \mathbb{R}^d, i = 1, ..., N$ is the position vector, M is a symmetric positive definite mass matrix, $U(q)$ represents the potential, W_t represents a vector of N independent and uncorrelated Wiener processes in $\mathbb{R}^d$ with as time derivative a vector of independent white noise processes obeying (2.1), and Γ and Σ are symmetric positive definite matrices, which we shall assume to be position-independent in the remainder of this thesis. A generalized version of Langevin Dynamics, where the friction matrix Γ varies with position was discussed in [243] and can be used to e.g. study flocking phenomena.

In the special case where

$$\Sigma\Sigma^T = 2\tau\Gamma, \tag{2.35}$$

for scalar $\tau > 0$ the dynamics obeys a fluctuation-dissipation theorem, and under some mild assumptions is provably ergodic. This ensures that solutions of the Langevin dynamics SDE (2.34) effectively sample the distribution $\rho_\tau(q, p)$, where

$$\rho_\tau(q, p) \propto e^{-(p^T M^{-1} p/2 + U(q))/\tau}. \tag{2.36}$$

is the Boltzmann-Gibbs distribution, which we recall from Section 1.2.1 on the canonical ensemble (Eq. (1.15)), where τ plays the same role as the temperature in statistical physics. If we use $\tau = 1$ we obtain a fully Bayesian approach. If we take $\tau \to 0$ we confine the posterior probability mass more and more to the mode(s) of the distribution, obtaining in the limit a maximum a posteriori (MAP) approach. In Chapter 3 we shall explore how considering $\tau \ll 1$ can be used to enhance the neural network parameterization procedure.

As the stationary distribution (2.36) does not depend on the friction term Γ, a common simplification is to simply choose $\Gamma = \gamma I$ and $\Sigma = \sqrt{2\gamma\tau}M^{1/2}$ in (2.34b), where $\gamma > 0$ is the friction coefficient (also known as the collision rate) and τ is the temperature. In what follows, we will make use of this convention. The Langevin dynamics equations then become

$$\begin{aligned}\mathrm{d}q &= M^{-1}p\,\mathrm{d}t,\\ \mathrm{d}p &= -\nabla_q U(q)\,\mathrm{d}t - \gamma p\,\mathrm{d}t + \sqrt{2\gamma\tau}M^{1/2}\,\mathrm{dW}_t,\end{aligned} \tag{2.37}$$

In the limit $\gamma \to 0$ Hamiltonian dynamics (1.16) is recovered. We refer to system (2.37) as *underdamped* Langevin dynamics. In the *overdamped* (infinite friction) limit, where $\gamma \to \infty$, time is rescaled as γt, and the mass is assumed to be the identity matrix, we obtain

$$\mathrm{d}q = -\nabla U(q)\ \mathrm{d}t + \sqrt{2\tau}\ \mathrm{d}W_t. \tag{2.38}$$

Overdamped Langevin dynamics is ergodic with respect to

$$\rho_{q_\tau}(q) = \frac{1}{Z} e^{-U(q)/\tau}, \text{ where } Z = \int e^{-U(q)/\tau}, \tag{2.39}$$

and has generator [168]

$$\mathcal{L}_{\text{OD}} = -\nabla U \cdot \nabla + \tau \Delta. \tag{2.40}$$

Underdamped Langevin dynamics (2.37) with identity mass matrix has generator

$$\mathcal{L} = \quad \mathcal{L}_H \quad + \quad \mathcal{L}_{OU} \tag{2.41}$$

$$= p \cdot \nabla_q - \nabla_q U \cdot \nabla_p + \gamma(-p \cdot \nabla_p + \tau \Delta_p), \tag{2.42}$$

where $\mathcal{L}_H$ and $\mathcal{L}_{OU}$ are the generator of Hamiltonian dynamics and the Ornstein-Uhlenbeck process ((2.24) where $\sigma = \gamma\tau$), respectively. Underdamped Langevin dynamics has Fokker-Planck operator [168]

$$\mathcal{L}^* = -p \cdot \nabla_q + \nabla_q U \cdot \nabla_p + \gamma(\nabla_p p \cdot + \tau \Delta_p). \tag{2.43}$$

It can easily be shown that $\mathcal{L}^* \rho_\tau(q,p) = 0$ for Boltzmann-Gibbs distribution $\rho_\tau(q,p)$ as defined in (2.36), so the canonical measure is invariant. The generator of underdamped Langevin dynamics can be shown to be hypoelliptic [117, 118]. We will first show this for a simplified case in Example 2.2.1 and will then treat the general case.

Example 2.2.1. Consider underdamped Langevin dynamics in 1 dimension, $(q,p) \in \mathbb{R}^2$, and set $\gamma = \tau = 1$ to study Hörmander's condition. The generator of underdamped Langevin dynamics SDE can then be written as

$$\mathcal{L} = \mathrm{A}_0 + \mathrm{A}_1^2,$$
$$\text{where } \mathrm{A}_0 = p\partial_q - \partial_q U(q)\partial_p - p\partial_p \text{ and } \mathrm{A}_1 = \partial_p. \tag{2.44}$$

Then

$$[\mathrm{A}_1, \mathrm{A}_0] = \mathrm{A}_1\mathrm{A}_0 - \mathrm{A}_0\mathrm{A}_1 = (\partial_q - \partial_p). \tag{2.45}$$

Therefore, $\text{span}(\mathrm{A}_1, [\mathrm{A}_1, \mathrm{A}_0])$ equals the tangent space of $\mathbb{R}^2$. So Hörmander's condition holds.

For the more general case we can rewrite the generator (2.42) of underdamped Langevin dynamics as

$$\mathcal{L} = A_0 + \sum_{k=1}^{d} A_k^2, \tag{2.46}$$

where

$$A_0 = \mathcal{L}_H - \gamma p \cdot \nabla_p \text{ and } A_k = \sqrt{\gamma\tau}\partial_{p_k},\ 1 \le k \le d. \tag{2.47}$$

Then

$$[A_k, A_0] = A_k A_0 - A_0 A_k = \sqrt{\gamma\tau}\left(\partial_{q_k} - \gamma\partial_{p_k}\right). \tag{2.48}$$

The Lie algebra generated by $\{A_k\}_{k=1,\dots,d}$ and $\{[A_k, A_0]\}_{k=1,\dots,d}$ has maximal rank $2d$ for every configuration (q,p) in the state space. So Hörmander's condition holds

$$\text{span}\left(A_0, \dots, A_d, [A_1, A_0], \dots, [A_k, A_0]\right) = \text{span}\left(\partial_{q_1}, \dots, \partial_{q_d}, \partial_{p_1}, \dots, \partial_{p_d}\right) \tag{2.49}$$

and $\mathcal{L}$ is hypoelliptic. Equivalently, it can be shown that the Fokker-Planck operator of Langevin dynamics (2.43) is hypoelliptic.

2.2.1 Numerical Methods for Langevin Dynamics

Langevin dynamics is the main stochastic differential equation of interest for this thesis. We will discuss numerical methods for both overdamped (2.38) and underdamped (2.37) Langevin dynamics in Section 2.2.1.1 and Section 2.2.1.2, respectively.

2.2.1.1 Euler-Maruyama

Consider a first order SDE

$$dX = a(X)dt + b(X)\mathrm{d}W_t,$$

where W_t is a Wiener process. This can be solved numerically using the Euler-Maruyama method (an extension of the well-known Euler method)

$$X_{n+1} = X_n + ha(X_n) + b(X_n)\sqrt{h}R_n, \tag{2.50}$$

where h is the timestep and R_n is a normal random variable with zero mean and unit variance [220]. For overdamped Langevin dynamics (2.38) one obtains using the Euler-Maruyama method

$$q_{n+1} = q_n - h\nabla U(q_n) + \sqrt{2\tau h}R_n, \tag{2.51}$$

where R_n is a vector of i.i.d. standard normal random variables drawn at timestep n.

2.2.1.2 Langevin Dynamics Splitting Methods

Several numerical schemes for underdamped Langevin dynamics (2.37) have been proposed, e.g. [2, 31, 34, 156, 267]. To build discretization schemes one can use splitting methods [157, 158]. Such schemes split the vector field for underdamped Langevin dynamics into different parts, which are solved separately. In this thesis we shall split (2.37) into pieces called A, B, and O

$$\mathrm{d}\begin{bmatrix} q \\ p \end{bmatrix} = \underbrace{\begin{bmatrix} M^{-1}p \\ 0 \end{bmatrix}\mathrm{d}t}_{A} + \underbrace{\begin{bmatrix} 0 \\ -\nabla U(q) \end{bmatrix}\mathrm{d}t}_{B} + \underbrace{\begin{bmatrix} 0 \\ -\gamma p\ \mathrm{d}t + \sqrt{2\gamma\tau}M^{1/2}\ \mathrm{d}W_t \end{bmatrix}}_{O}. \tag{2.52}$$

When taken individually, each part can be solved 'exactly' in its evolution of distribution [157]. The corresponding individual update maps are given by

$$\mathcal{F}_h^A(q,p) = (q + hM^{-1}p, p), \tag{2.53a}$$
$$\mathcal{F}_h^B(q,p) = (q, p - h\nabla U(q)), \tag{2.53b}$$
$$\mathcal{F}_h^O(q,p) = (q, e^{-\gamma h}p + \sqrt{\tau(1-e^{-2\gamma h})}M^{1/2}R), \tag{2.53c}$$

where parts A and B correspond to those we introduced for deterministic Hamiltonian dynamics (1.25). The update map for part O in (2.53c) can be obtained by studying the Ornstein-Uhlenbeck SDE, see Example 2.1.1. Schemes are coded by the order in which the updates are applied. The BAOAB scheme, which was shown to have excellent configurational sampling properties by Leimkuhler and Matthews [156], is described by

$$\mathcal{F}_{h/2}^B \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_h^O \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_{h/2}^B, \tag{2.54}$$

where repeated letters indicate half steps of size $h/2$. Using the update rules in (2.53) one obtains for the BAOAB scheme

$$\begin{aligned} p_{n+1/2} &:= p_n - \frac{h}{2}\nabla U(q_n), \\ q_{n+1/2} &:= q_n + \frac{h}{2}M^{-1}p_{n+1/2}, \\ \hat{p}_{n+1/2} &:= \alpha p_{n+1/2} + \sqrt{\tau(1-\alpha^2)}M^{1/2}R_n, \quad \text{where } \alpha = e^{-\gamma h}, \\ q_{n+1} &:= q_{n+1/2} + \frac{h}{2}M^{-1}\hat{p}_{n+1/2}, \\ p_{n+1} &:= \hat{p}_{n+1/2} - \frac{h}{2}\nabla U(q_{n+1}). \end{aligned} \tag{2.55}$$

The BAOAB scheme only requires a single force-evaluation per timestep, which makes it computationally efficient as the force-evaluation tends to form the computational bottleneck for large systems. Construction of higher order schemes typically requires multiple force-evaluations.

In the overdamped limit $\gamma \to \infty$ the exact solution of part O reduces to $p = \sqrt{\tau}M^{1/2}R$, which gives the BAOAB limit scheme

$$\begin{aligned} q_{n+1} &= q_n - \frac{h^2}{2}M^{-1}\nabla U(q_n) + \frac{h}{2}\sqrt{\tau}M^{-1/2}(R_n + R_{n-1}), \\ &= q_n - \tilde{h}M^{-1}\nabla U(q_n) + \sqrt{2\tau\tilde{h}}M^{-1/2}(R_n + R_{n-1})/2, \text{ with } \tilde{h} = h^2/2. \end{aligned} \tag{2.56}$$

As comparison the Euler-Maruyama method gives

$$q_{n+1} = q_n - hM^{-1}\nabla U(q_n) + \sqrt{2\tau h}M^{-1/2}R_n. \tag{2.57}$$

The BAOAB limit scheme is shown to give improved accuracy with respect to the Euler-Maruyama method in Leimkuhler and Matthews [156].

For a more general formulation with a position-dependent friction matrix it can be computationally expensive to compute an exact solution of the O part. Sachs et al. [243] therefore propose to use a multiple timestepping BAOAB scheme to circumvent this. In this case the numerical approximation for the O-step (obtained by further splitting it into separate parts) is performed K times with a smaller stepsize of h/K to reduce numerical errors introduced by the approximation. For the interested reader we refer to Sachs et al. [243] for further reading.

2.3 Adaptive Langevin Dynamics

The Adaptive Langevin dynamics (AdL) method is a modification of Langevin dynamics (2.37), where the friction parameter γ is automatically determined by an isokinetic control law [128]. A similar negative feedback loop control law for a dynamical friction is used in Nosé-Hoover dynamics, which was developed by S. Nosé and W. Hoover in the 1980s to sample from the canonical ensemble using a deterministic system [216, 109]. Adaptive Langevin dynamics was first introduced in molecular dynamics. For challenging force computations, the numerical approximation of the gradient may bias the invariant distribution [128]. AdL allows for adaptive dissipation of the excess noise, so that the correct distribution is maintained. AdL has also been used for Bayesian inference to remove the bias arising from mini-batching for specific cases of the covariance matrix [54, 250]. The method was analyzed from both a numerical and theoretical viewpoint in [253, 161, 99, 164].

Adaptive Langevin dynamics takes the form of a degenerate SDE system

$$\begin{aligned} \mathrm{d}q &= p\ \mathrm{d}t, \\ \mathrm{d}p &= -\nabla U(q)\mathrm{d}t - \xi p\ \mathrm{d}t + \sigma_{\mathrm{A}}\mathrm{d}\mathrm{W}_t, \\ \mathrm{d}\xi &= \varepsilon\left(p^T p - N\tau\right)\mathrm{d}t, \end{aligned} \tag{2.58}$$

where we have set the mass matrix to be the identity matrix for simplicity. AdL has as hyper-

parameters: the coupling coefficient ε, the number of degrees of freedom N, the temperature τ, and the driving noise amplitude σ_{A}. The friction is adjusted using the feedback law so that the thermodynamics average of $p^T p$ is $N\tau$. If $|p|^2 > N\tau$ the kinetic energy is too large, so the friction is increased which in turn decreases p. For the opposite case $|p|^2 < N\tau$, the feedback law effectively acts as a driving and increases p. If the system is subject to constraints one should take care to subtract the number of constraints from the degrees of freedom N.

To build discretization schemes for Adaptive Langevin dynamics one can use splitting methods in a similar vein as for Langevin dynamics (Section 2.2.1.2). Different discretization schemes are obtained by breaking up the AdL system (2.58) into different parts, which are solved separately. The maps $\mathcal{F}_h^A$ and $\mathcal{F}_h^B$ are identical to those described for Langevin dynamics (2.53a)-(2.53b) (we have set the mass to be the identity matrix), but formally need to be supplemented with an identity mapping of ξ. Additionally, we define flow maps $\mathcal{F}_h^C, \mathcal{F}_h^D$, and, $\mathcal{F}_h^E$ by

$$\begin{aligned}
\mathcal{F}_h^A(q,p,\xi) &= (q + hp, p, \xi),\\
\mathcal{F}_h^B(q,p,\xi) &= (q, p - h\nabla U(q), \xi),\\
\mathcal{F}_h^C(q,p,\xi) &= (q, e^{-h\xi}p, \xi),\\
\mathcal{F}_h^D(q,p,\xi) &= (q, p + \sigma_{\mathrm{A}}\sqrt{h}R, \xi),\\
\mathcal{F}_h^E(q,p,\xi) &= (q, p, \xi + h\varepsilon\left[p^T p - N\tau\right]).
\end{aligned} \tag{2.59}$$

A straightforward method (in the same vein as BAOAB (2.54)) is then defined by the composition BACDEDCAB

$$\mathcal{F}_{h/2}^B \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_{h/2}^C \circ \mathcal{F}_{h/2}^D \circ \mathcal{F}_h^E \circ \mathcal{F}_{h/2}^D \circ \mathcal{F}_{h/2}^C \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_{h/2}^B. \tag{2.60}$$

Alternatively, one can combine the components in part C and D, which results in an Ornstein-Uhlenbeck equation, i.e. flow map $\mathcal{F}_h^F$ would represent the weak solution of the SDE

$$\mathrm{d}p = -\xi p\ \mathrm{d}t + \sigma_{\mathrm{A}}\mathrm{dW}, \tag{2.61}$$

where ξ is held constant. The flow map $\mathcal{F}_h^F$ would then replace $\mathcal{F}_h^C$ and $\mathcal{F}_h^D$. At the point of writing this thesis, an extensive study of which method (or choice of ordering) is most advantageous is still an open question. We did not find notable differences between either approach, and therefore use (2.60) as discretization scheme for Adaptive Langevin dynamics in Chapter 3.

2.4 SDE-based Schemes in Machine Learning

Consider for a supervised learning setting (Section 1.1.1), a neural network with parameters θ (Section 1.1.2). The neural network parameters are updated using update rules that involve the gradients of the loss with respect to the parameters (Section 1.1.3). We denote by $G(\theta) := -\nabla_\theta L$ the (full) negative gradient of the loss. The starting point for most neural network training schemes is system of ordinary differential equations

$$\mathrm{d}\theta = G(\theta)\mathrm{d}t. \tag{2.62}$$

Using explicit Euler one obtains the gradient descent method (Section 1.1.3.1)

$$\theta_{n+1} = \theta_n + hG(\theta_n), \tag{2.63}$$

for discretization stepsize (learning rate) $h > 0$. In the stochastic gradient descent (SGD) method one approximates the gradient by evaluating it on a randomized (uniformly sampled) finite subset of the dataset (called a minibatch) at every timestep. This introduces gradient noise into the dynamics. For a minibatch B of size $S < N$, the gradient evaluated on this

subset is

$$G_{(S)}(\theta) = -\frac{1}{S}\sum_{n\in B}\nabla_\theta \mathcal{L}(\theta, x_n). \tag{2.64}$$

Throughout the rest of the thesis we shall refer to $G_{(S)}(\theta)$ as the perturbed gradient $\tilde{G}(\theta)$. SGD has been modelled as a stochastic differential equation with an unknown parameter-dependent covariance matrix in the literature, see e.g. [253, 43], with theoretical justifications (weak approximation) provided in [111, 176] (small learning rate case) and [179] (finite learning rate case). The type of parameterizations found by SGD are directly influenced by the rate of stepsize to batch size [125]. Further discussion of SGD can be found in Section 1.1.3.2.

2.4.1 Stochastic Gradient Langevin Dynamics

The stochastic gradient Langevin dynamics (SGLD) method [285] maintains the computational efficiency of stochastic gradient descent compared to traditional gradient descent, while introducing additional additive noise (with an adjustable, but fixed strength) to stabilize the invariant measure of the stochastic dynamics

$$\mathrm{d}\theta = \tilde{G}(\theta)\mathrm{d}t + \sigma_{\mathrm{A}}\mathrm{d}\mathrm{W}_t, \tag{2.65}$$

with constant σ_A and where we again use $\tilde{G}$ to denote the negative truncated gradient evaluated on a minibatch of the data. SGLD is typically discretized using Euler-Maruyama, resulting in

$$\theta_{n+1} = \theta_n + h\tilde{G}(\theta_n) + \sigma_{\mathrm{A}}\sqrt{h}R_n. \tag{2.66}$$

For a decaying sequence of stepsizes $h_n \to 0$, we eventually expect to generate states from a known invariant distribution using the SGLD method. Unfortunately, using such small stepsizes would significantly slow down the sampling process. Therefore, one typically accepts a small bias in order to be able to more efficiently sample the target distribution for practical applications. The properties of SGLD were studied in [266, 280].

SGLD may suffer from slow mixing rates [3], which is why we consider introducing additive noise within a second order stochastic dynamics framework. We shall use the 'minibatch' variant of underdamped Langevin dynamics.

2.4.2 Relation between Langevin Sampling Methods and Schemes in the Machine Learning Literature[1]

We consider the following model of underdamped Langevin dynamics

$$\begin{aligned} \mathrm{d}\theta &= p\,\mathrm{d}t, \\ \mathrm{d}p &= \tilde{G}(\theta)\,\mathrm{d}t - \gamma p\,\mathrm{d}t + \sqrt{2\gamma\tau}\,\mathrm{d}\mathrm{W}_t, \end{aligned} \tag{2.67}$$

where $\tilde{G}$ again denotes the truncated negative gradient obtained by selecting a randomized finite subset of the data at each timestep. We will show how for specific hyperparameter choices and discretization of Langevin dynamics we arrive at well-known neural network parameterization schemes. Using the flow mappings defined in (2.53), the OBA scheme for (2.67) can be written as Algorithm 2.1.

We can define a family of schemes through specific choices of friction γ and temperature τ. Some schemes correspond to existing schemes in the literature. For example, setting $\tau = 0$ and using finite friction we arrive at a reparameterization of the SGD with momentum scheme, with two variants given in Algorithm 2.2. Choosing the damping parameter $\mu = h\exp(-\gamma h)$ and learning rate $\delta t = h^2$ in type I we recover the OBA scheme with $\tau = 0$. Similarly in type II we reparameterize $\mu = (1 + h\exp(\gamma h))^{-1}$ and $\delta t = h + \exp(-\gamma h)$. In the limit $\gamma \to \infty$ (or equivalently if $\mu \to 0$), we recover the traditional SGD scheme.

[1] This section is adapted from [163], but is provided here instead of in Chapter 3 (where [163] is discussed) for clarity of presentation.

Algorithm 2.1 The OBA Splitting Scheme

procedure OBA(θ,p,γ,τ,T,h)
  for $t \leftarrow 1$ to T **do**
    $R \leftarrow \mathcal{N}(0,1)$ ▷ R is a vector of i.i.d. Gaussian random numbers
    $p \leftarrow \exp(-\gamma h)p + \sqrt{\tau(1-\exp(-2\gamma h))}R$
    $p \leftarrow p + h\tilde{G}(\theta)$
    $\theta \leftarrow \theta + hp$
  return θ, p

Algorithm 2.2 The OBA Splitting Scheme with $\tau = 0$

procedure OBA_TAU_IS_ZERO(θ,p,γ,T,h)
  for $t \leftarrow 1$ to T **do**
    $p \leftarrow \exp(-\gamma h)p + h\tilde{G}(\theta)$
    $\theta \leftarrow \theta + hp$
  return θ, p
procedure SGD_WITH_MOMENTUM_I(θ,v,μ,T,δt)
  for $t \leftarrow 1$ to T **do**
    $v \leftarrow \mu v + \delta t\, \tilde{G}(\theta)$
    $\theta \leftarrow \theta + v$
  return θ, v
procedure SGD_WITH_MOMENTUM_II(θ,v,μ,T,δt)
  for $t \leftarrow 1$ to T **do**
    $v \leftarrow \mu v + (1-\mu)\tilde{G}(\theta)$
    $\theta \leftarrow \theta + \delta t\, v$
  return θ, v

Similarly we may consider the limiting case of infinite friction and positive τ in the OBA scheme, where the momenta are redrawn from their distribution at every step. The resulting scheme (see Algorithm 2.3) matches the SGLD scheme with a reparameterization between temperature and noise strength $\sigma^2 = \tau$ and learning rate $\delta t = h^2$. We may extend SGLD to include momentum by instead using a finite friction parameter γ in Algorithm 2.1.

Thus, with a specific interpretation of the coefficients in SGD with momentum and SGLD we can obtain certain Langevin integrators. In Chapter 3 and Chapter 4 we shall consider underdamped Langevin dynamics as a somewhat general scheme that incorporates the ideas of additive noise and momentum for neural network training.

2.5 Constrained SDEs

In molecular dynamics holonomic constraints can be used to increase the maximum usable timestep for which stability is maintained (Section 1.2.2.2). In Section 1.2.4 we introduced constrained dynamics for the deterministic case. Similarly, constraints can be introduced to the overdamped Langevin SDE (2.38) as follows

$$\begin{aligned} \mathrm{d}q &= -\nabla U(q)\,\mathrm{d}t + \sqrt{2\tau}\,\mathrm{d}W_t - g'(q)^T\,\mathrm{d}\lambda, \\ 0 &= g(q), \end{aligned} \tag{2.68}$$

where λ represents an $\mathbb{R}^m$-valued vector of Lagrange multipliers, $q \in \mathbb{R}^d$, and $g : \mathbb{R}^d \to \mathbb{R}^m$ [157]. Provided the initial configuration q_0 satisfies the constraint, any trajectory q_t of (2.68) remains on the constraint manifold

$$\Sigma = \{q \in \mathbb{R}^d | g(q) = 0\}. \tag{2.69}$$

Algorithm 2.3 The OBA Splitting Scheme with infinite friction

procedure OBA_INFINITE_FRICTION(θ,τ,T,h)
  for $t \leftarrow 1$ to T **do**
    $R \leftarrow \mathcal{N}(0,1)$ ▷ R is a vector of i.i.d. Gaussian random numbers
    $\theta \leftarrow \theta + h^2\tilde{G}(\theta) + \sqrt{\tau}hR$
  return θ
procedure SGLD(θ,σ,T,δt)
  for $t \leftarrow 1$ to T **do**
    $R \leftarrow \mathcal{N}(0,1)$ ▷ R is a vector of i.i.d. Gaussian random numbers
    $\theta \leftarrow \theta + \delta t\tilde{G}(\theta) + \sigma\sqrt{\delta t}R$
  return θ

We show in Chapter 4 that (2.68) is equivalent to an underlying ergodic (unconstrained) SDE with unique invariant measure

$$\mathrm{d}\nu_\Sigma = Z^{-1}e^{-U(q)/\tau}\,\mathrm{d}\sigma_\Sigma,\ Z = \int_\Sigma e^{-U(q)/\tau}\,\mathrm{d}\sigma_\Sigma, \tag{2.70}$$

where σ_Σ is the surface measure on Σ. Ergodicity ensures that averages of observables with respect to ν_Σ can be approximated by time averages of trajectories of (2.68). The ergodic properties of constrained overdamped Langevin dynamics were studied in Lelièvre et al. [169].

Introducing momenta leads to the constrained counterpart of the underdamped Langevin SDE (2.37)

$$\begin{aligned}\mathrm{d}q &= M^{-1}p\,\mathrm{d}t,\\ \mathrm{d}p &= -\nabla U(q)\,\mathrm{d}t - \gamma p\,\mathrm{d}t + \sqrt{2\gamma\tau}M^{1/2}\,\mathrm{dW}_t - g'(q)^T\,\mathrm{d}\lambda,\\ 0 &= g(q),\end{aligned} \tag{2.71}$$

The co-tangency condition which need to be obeyed is

$$g'(q)^T M^{-1}p = 0. \tag{2.72}$$

The corresponding phase space is the cotangent bundle $T^*\Sigma = \{(q,p) \mid q \in \Sigma, p \in T_q^*\Sigma\}$, where $T_q^*\Sigma = \{p \in \mathbb{R}^d \mid \nabla^T g(q)M^{-1}p = 0\}$ is the cotangent space of the manifold Σ for a given $q \in \Sigma$. The ergodic properties of constrained underdamped Langevin dynamics were studied in Lelièvre et al. [166]. Given an initial pair $(q_0,p_0) \in T^*\Sigma$, any trajectory (q_t,p_t) of (2.71) stays on $T^*\Sigma$ for all time. (2.71) is equivalent to an underlying ergodic SDE with invariant measure

$$\mathrm{d}\mu = Z^{-1}e^{-H(q,p)/\tau}\mathrm{d}\sigma_{T^*\Sigma}, \tag{2.73}$$

with normalizing constant Z, Hamiltonian $H(q,p) = U(q) + \frac{1}{2}p^T M^{-1}p$, and Liouville measure of the cotangent bundle $\sigma_{T^*\Sigma}$ [166].

2.5.1 Discretization of Constrained Langevin Dynamics

For any discretization scheme, the initialization should be such that the constraint is obeyed initially. Then for constrained overdamped Langevin dynamics (2.68) the simplest discretization scheme takes an Euler-Maruyama step (as for the unconstrained case, see Section 2.2.1.1), followed by a projection onto the constraint manifold (2.69) [157]. The ideal way to perform this projection is constraint-specific.

Equivalently, to develop an algorithm for constrained underdamped Langevin dynamics (2.71), the most straightforward approach is to first evolve unconstrained Langevin dynamics (2.37) and then follow this by SHAKE/RATTLE (see Section 1.2.4.1) to perform the projection back to the constraint manifold [5, 157, 242]. Alternatively, one can build discretization schemes using the splitting methods introduced in Section 2.2.1.2. In particular, we could split constrained underdamped Langevin dynamics (2.71) into a constrained Hamiltonian part and

a constrained Ornstein-Uhlenbeck part. Using the same A, B, O notation as before, a splitting strategy under the constraint $0 = g(q_t)$ and co-tangency condition $0 = \nabla_q g(q_t) p_t$, then gives

$$\begin{aligned} A: &\ \mathrm{d}q_t = p_t \,\mathrm{d}t, \quad \mathrm{d}p_t = -\nabla_q g(q_t) \,\mathrm{d}\lambda_t, \\ B: &\ \mathrm{d}q_t = 0, \ \mathrm{d}p_t = -\nabla_q U(q_t) \,\mathrm{d}t - \nabla_q g(q_t) \,\mathrm{d}\mu_t, \\ O: &\ \mathrm{d}q_t = 0, \ \mathrm{d}p_t = -\gamma p_t \,\mathrm{d}t + \sqrt{2\gamma\tau} \,\mathrm{d}\mathcal{W}_t - \nabla g(q_t) \,\mathrm{d}\nu_t, \end{aligned} \tag{2.74}$$

where M is set to be the identity matrix. The B and O components can then be solved exactly (in law). For part A one can use SHAKE or RATTLE as discussed in Section 1.2.4.1 for constrained ODEs. Importantly, no evaluation of the gradient is required for part A. One can then proceed using e.g. the OBA scheme or the symmetric BAOAB method (Section 2.2.1.2).

We use discretized constrained SDEs for neural network training in Chapter 4. We provide a general framework, alongside suitable choices for the constraints and discretization schemes.

Chapter 3

Partitioned Integrators for Thermodynamic Parameterization of Neural Networks

The results presented in this chapter appeared in *Foundations of Data Science* in 2019 [163] and are joint work with Benedict Leimkuhler and Charles Matthews. A shorter workshop paper appeared in the *NeurIPS Machine Learning and the Physical Sciences workshop* in 2019 [279]. The presentation has been slightly altered to match the contents in this thesis and additional context is provided where necessary. In particular, Section 3.3.2 has been significantly expanded (with corresponding numerical experiment Figure 3.15) and Section 3.8 (Summary & Outlook) is new.

Abstract

Traditionally, neural networks are parameterized using optimization procedures such as stochastic gradient descent, RMSProp and Adam. These procedures tend to drive the parameters of the network toward a local minimum. In this work, we employ alternative "sampling" algorithms (referred to as "thermodynamic parameterization methods"), which rely on discretized stochastic differential equations for a defined target distribution on parameter space. We show that the thermodynamic perspective already improves neural network training. Moreover, by partitioning the parameters based on natural layer structure we obtain schemes with rapid convergence for certain classification problems.

We describe easy-to-implement hybrid partitioned numerical algorithms, based on discretized stochastic differential equations, which are adapted to feed-forward neural networks, including a multi-layer Langevin algorithm, AdLaLa (combining the adaptive Langevin and Langevin algorithms) and LOL (combining Langevin and Overdamped Langevin); we examine the convergence of these methods using numerical studies and compare their performance among themselves and in relation to standard alternatives such as stochastic gradient descent and Adam. We present evidence that thermodynamic parameterization methods can be (i) faster, (ii) more accurate, and (iii) more robust than standard algorithms used within machine learning frameworks.

3.1 Introduction

Neural networks (NNs) are an important class of complex, hierarchical models which have been used in recent years for a vast range of applications. As impactful examples we mention the exploration of chemical structure [249], medical decision making strategies for palliative care

[9] and AlphaZero which is able to master a complex challenge, e.g. learning to play Go or chess, in the span of a few days [255]. Yet there remain a number of mysteries regarding the performance of neural networks, their generality, and their ultimate reliability. An important practical challenge is that neural networks require considerable computational power for training and, in many applications, re-training. Neural networks are typically parameterized/trained using variants of (stochastic) gradient descent, where the parameters – the weights and biases of the neural network – are updated so that the training loss (the difference between the neural network output and the 'truth') is minimized. In this work, we describe new training methods suited to neural network parameterization which are applicable in a variety of settings. We focus on classification problems and single hidden layer perceptrons. Our methods combine two basic ingredients: (i) the use of additive noise within a framework of second order stochastic dynamics, and (ii) exploitation of layer structure which induces a partitioning of the parameters of the network. The algorithms we present build directly on recent ergodicity results obtained for Langevin and Adaptive Langevin algorithms [162, 243, 164].

An important performance measure of a trained neural network is its capacity to generalize from its training data to unseen (test) data. Although a neural network can perform extremely well on the data on which it was trained, the algorithm used for optimization may easily end up in a minimum which does not generalize well to unseen data, a phenomenon called overfitting. Several factors appear to influence the generalization capacity of a neural network, such as the number of parameters, initialization, learning rate, stopping criterion, activation functions, and numerical method used, and no clear consensus has been reached on how these concepts interplay with one-another. Zhang et al. [306] found that traditional complexity measures from statistical learning theory are incapable of explaining several features of the generalization behaviour of deep neural networks (DNNs). In particular, they demonstrated that neural networks have such a high capacity that they can memorize the training data and can obtain zero training error on random labels (when using an architecture that gave good generalization properties when training with real labels). Explicit regularization techniques are unable to reliably attenuate this phenomenon [306]. Regularization, which adds a parameter norm penalty term to the loss function of neural networks, is a standard approach to prevent overfitting, but does not necessarily affect the generalization error.

So how do we find parameterizations that generalize well? Loss landscapes of deep neural networks are known to possess many low-loss minima [45], but not all of these minima generalize equally well and different optimizers may find different solutions [210, 289, 120]. The loss landscapes of neural networks are difficult to interpret due to their high-dimensionality and non-convexity. Goodfellow et al. [87] used a 1D linear interpolation technique to study the loss along the linear path between the initial and final model state. They showed that a large variety of neural networks never encounter any obstacles along this path, i.e. the loss along this path typically decreases monotonically. They used this result to motivate the success of methods such as stochastic gradient descent (SGD) in optimizing neural networks, despite the non-convexity of the objective functions. However, in this work we argue that the results obtained by Goodfellow et al. [87] do not hold for some common types of problems, for which SGD –as well as improved optimizers such as Adam [139]– can be shown to fail. This failure may be a consequence of the more complex structure of the loss landscape of these problems. This motivates the development of more sophisticated schemes for enhanced exploration of the low loss states in these settings.

3.1.1 Bayesian Perspective on Neural Network Training

In this work, we focus on the training (parameterization) process for neural networks using ideas from statistical mechanics. Neural networks approximate a function $y = f(x)$, $f : \mathbb{R}^m \to \mathbb{R}^n$ by an abstract family of maps having a simple homogeneous form. Here consider a single hidden

layer perceptron network with the structure

$$z_j = \varphi\left(\sum_{i=1}^{m} w_{ij}^{(1)} x_i + b_j^{(1)}\right), \qquad j = 1, 2, \ldots, d,$$
$$\hat{y}_k = \varphi_{\mathrm{o}}\left(\sum_{j=1}^{d} w_{jk}^{(2)} z_j + b_k^{(2)}\right), \quad k = 1, 2, \ldots, n.$$

where $x \in \mathbb{R}^m$ is an input data vector, $w^{(1)} \in \mathbb{R}^{d\times m}$ and $w^{(2)} \in \mathbb{R}^{n\times d}$ are matrices that contain the weights of the various layers, $b^{(1)} \in \mathbb{R}^d$ and $b^{(2)} \in \mathbb{R}^n$ are the biases, $z \in \mathbb{R}^d$ is the networks output after passing the input data through the first layer, and $\hat{y} \in \mathbb{R}^n$ is the neural network's approximation of the (for a classification problem) true labels y. The function φ is taken to be a ReLU activation function [124, 84] in our experiments. The function φ_0 is taken to be either a sigmoid, for a binary classification problem, or a softmax, for the MNIST dataset. The equations define a map $\Phi : \mathbb{R}^m \times \mathbb{R}^q \to \mathbb{R}^n$, where $q = md + d + (d + 1)n$ is the dimension of the parameter space, that is we have

$$\hat{y} = \Phi(x, \theta), \tag{3.1}$$

where θ contains all the parameters of the neural network. The data $\mathcal{D}$ fed into the neural network consists of pairs of input data x and their labels y. The loss function is then determined by the difference between the neural network output $\hat{y}$ and the true labels y. In our experiments we used a binary cross entropy loss function (for binary classification) or cross entropy loss function (for MNIST), but many different loss functions are available [203].

In this work, we take the Bayesian perspective, that the parameters θ are defined by the data $\mathcal{D}$ only in the sense of a probability distribution defined by Bayes' formula,

$$\rho(\theta|\mathcal{D}) \propto \rho(\mathcal{D}|\theta)\rho_0(\theta), \tag{3.2}$$

where $\rho(\mathcal{D}|\theta)$ is the likelihood, which for a cross entropy loss function takes the form $\rho(\mathcal{D}|\theta) = \exp\left[\sum_i \left(y_i \log[\Phi(x_i, \theta)] + (1 - y_i)\log[1 - \Phi(x_i, \theta)]\right)\right]$, and $\rho_0(\theta)$ encodes prior knowledge of θ. The exploration of values of θ that are consistent with Bayes' rule then becomes the outstanding challenge. When $\rho(\theta|\mathcal{D})$ is unimodal and convex it is natural to choose θ as the mode of the target distribution by maximizing the posterior probability density, a technique referred to as "MAP," for "maximum a posteriori probability," but in practice this does not hold for neural networks and it then becomes a challenge to identify all relevant possible parameter values, and to compare different parameter choices in terms of their relative probabilistic weight. This task is referred to as *sampling*, and thus the Bayesian parameterization problem naturally reduces to a sampling problem for the parameters of the model. While the idea of Bayesian modelling is commonplace in all areas where statistics is used, the Bayesian perspective is usually only viewed as the starting point for optimization schemes in the setting of high dimensional neural networks, due to the vast amounts of data and parameters involved [208]. We argue here that the sampling approach provides parameterization candidates with as great or greater efficiency than standard optimization schemes.

A single parameter vector is typically not a meaningful way to characterize a model since it fails to capture the fundamental statistical nature of the relationship between data and model. Assuming that θ is a random variable partially constrained by the knowledge of the training set, the output $\hat{y}$ of the neural network is also a random variable with its own probability distribution directly related to that of θ. We can compute the mean of the output $\hat{y}$ from the parameter distribution

$$\bar{y} = \int \Phi(x, \theta)\rho(\theta|\mathcal{D})\mathrm{d}\theta, \tag{3.3}$$

where ρ is the normalized Bayesian density for θ. We sample parameter space by generating a sequence of discrete values of θ defined by some Markov chain $\theta_0 \mapsto \theta_1 \mapsto \ldots$ Then we simply

approximate $\bar{y}$ by

$$\bar{y} \approx N^{-1} \sum_{i=1}^{N} \Phi(x, \theta_i). \tag{3.4}$$

Depending on the application, it is often enough to perform a draw of a singleton from the distribution of parameters thus generated. Note that the Bayesian approach gives access to the mean $\bar{y}$, as well as other statistics (such as variances) by a similar procedure. It is also possible to rely on the mode (or several modes, in complicated systems) as proposed parameter values and to examine the sensitivities of those parameters using averaging methods.

Several issues are raised by the use of Markov Chain Monte Carlo (MCMC) methods, such as equilibration of the Markov process (the "burn in" phase, in the language of statistics), the problem of high correlation among the samples taken along the sampling path, and the actual computational procedure by which such samples can be generated efficiently. We will not address all the issues here, but we will show that taking a sampling perspective can cast new light on some challenging problems in machine learning.

There is a well known link between posterior sampling and MAP estimation. Introduce the negative log posterior $l(\theta) = -\ln \rho(\theta|\mathcal{D})$, and define

$$\rho_\tau(\theta) = \exp(-\tau^{-1} l(\theta)) = \rho(\theta)^{1/\tau}. \tag{3.5}$$

For $\tau = 1$ we have the posterior density. For $\tau \to 0$ we obtain a sequence of distributions which, although globally supported, have their mass confined progressively closer to the mode of the distribution. Thus we can think of MAP as an extreme form of sampling in which the sampled distribution is more and more confined to the vicinity of the mode or modes. In this setting, τ becomes a parameter of an embedded family of models which may be used to enhance the optimization process. Examples are annealing [141], where τ is gradually driven from higher to lower values, and simulated tempering [82, 194], where it is allowed to increase or decrease. The parameter τ plays precisely the same role as temperature in statistical physics, thus the use of the term *thermodynamic parameterization* to describe methods that rely on this embedding (and the sampling of the associated family of probability distributions) to enhance the parameterization procedure.

We note that the full exploration of the parameter space taken as a region of Euclidean space would be implausible in high dimensions. Neural networks are sometimes used with millions or billions of degrees of freedom and there is no conceivable way to fully explore such a space. On the other hand a very small range of parameter values are likely to be interest (the ones that have relatively large statistical weight with respect to the probability distribution). Moreover, there is often much to be gained by exploring parameters in the vicinity of a local optimum, i.e. by short sampling paths. It is important to recognize that MAP estimation, as normally practiced, is local, not global, optimization. Molecular dynamics [157] provides an obvious illustration of the potential value of the sampling paradigm in very high dimensions.

3.1.2 The Parameterization Process using Stochastic Gradients

In this section, we outline the standard training procedure based on stochastic gradient descent. In Section 3.1.3 we shall discuss the alternative stochastic gradient Langevin dynamics method as an illustration of a sampling method.

The starting point for most neural network training schemes is a system of ordinary differential equations $\mathrm{d}\theta = G(\theta)\mathrm{d}t$, where G is the negative gradient of loss function $L(\theta|\mathcal{D})$ defined in terms of the entire training dataset $\mathcal{D}$. Such gradient systems have the feature that, along their solutions $\theta(t)$, we have

$$\dot{L} = -\|G\|^2, \tag{3.6}$$

implying that the loss decreases monotonically along solutions. Since local minima are stationary points one hopes that this dynamics steadily drives the parameters to such local minima.

The most common numerical method used for solving the system is the explicit Euler method

$$\theta_{n+1} = \theta_n + hG(\theta_n), \tag{3.7}$$

for a choice of discretization stepsize $h > 0$. When the gradient is approximated by evaluating it on a randomly sub-sampled partial dataset we introduce gradient noise into the dynamics. This noise can be approximately modelled by replacing G in each evaluation by $\tilde{G}(\theta) = G(\theta) + \Sigma(\theta)R$ where $\Sigma\Sigma^T$ is the noise covariance matrix and R a standard normal random vector with iid components. We can thus re-interpret the training process as being

$$\theta_{n+1} = \theta_n + hG(\theta_n) + h\Sigma(\theta_n)R_n, \tag{3.8}$$

which we recognize as Euler-Maruyama discretization of the Itô SDE [75]

$$\mathrm{d}\theta = G(\theta)\mathrm{d}t + \sqrt{h}\Sigma(\theta)\mathrm{d}W, \tag{3.9}$$

using a stepsize h. It is an odd feature of the process that the discretization stepsize appears in the right hand side of the SDE itself [161]. The system (3.9) is driven by multiplicative noise. Since the gradient noise defined by $\Sigma(\theta)$ is complicated, this system of SDEs has an unknown invariant distribution which will depend on the subsampling. However, if $h \to 0$ in (3.9) it is clear that we arrive eventually at a local minimum of the loss. One assumes that for a small value of the stepsize the consequence is that we arrive near such a local minimum, or, to be precise, due to the inherent degeneracy of neural networks, near to a manifold of local minima of the loss.

One might worry that the dynamics could be drawn frequently toward saddles. Although such points are unstable –under continual perturbation the dynamics bypasses the saddles and local minima are indeed eventually located– there are other downsides to relying on gradient flow as the foundation for training algorithms. Namely, we can only ever count on gradient dynamics as a local minimization procedure. It has, a priori, no mechanism for global exploration. Introducing ad hoc mechanisms to increase exploration is prone to failure since, in high dimensions, there is no natural way to grid the parameter space.

There exist an increasing number of methods, in the same class as SGD, which are based on accelerating the scheme described above. These include SGD with momentum, RMSProp [269], AdaGrad [62] and Adam [139]. Although the efficacy of adaptive gradient methods in large scale machine learning is an active area of research [289], we have found for the cases considered in this work that Adam gave substantially better results than SGD.

3.1.3 SDE-based Schemes in Machine Learning

Stochastic Gradient Langevin dynamics (SGLD) [285], the Unadjusted Langevin Algorithm (ULA) [236, 63] and the Stochastic Gradient Nosé-Hoover Thermostat (SGNHT) [128, 54] are examples of existing thermodynamic sampling methods. In SGLD one introduces an additional additive noise into (3.9) resulting in

$$\mathrm{d}\theta = G(\theta)\mathrm{d}t + \sqrt{h}\Sigma(\theta)\mathrm{d}W + \sigma_{\mathrm{A}}\mathrm{dW}_{\mathrm{A}}. \tag{3.10}$$

The additive noise is usually taken to have constant variance.[1] SGLD is typically discretized using Euler-Maruyama, resulting in

$$\theta_{n+1} = \theta_n + hG(\theta_n) + h\Sigma(\theta_n)R_n + \sigma_{\mathrm{A}}\sqrt{h}R_n^{\mathrm{A}}. \tag{3.11}$$

At this stage, we see that for small h, the $\sqrt{h}$ term will strongly dominate the noise, and conclude that if we replace the constant stepsize h by a decaying sequence of stepsizes $h_n \to 0$, we would, in the long term, expect to generate states from the stationary distribution, thus the claim that SGLD is a sampling method for a known distribution. The mathematical analysis

[1] At this (formal) level we could of course combine the two Wiener processes, but it is desirable to keep them separate since the first (W) only enters into evaluations of the force and must always be realized in conjunction with force evaluation in the description of a numerical method.

of this method relies on the framework known as "stochastic approximation" [149]. The caveat of course is that such a rigorous procedure requires the use of small stepsizes which would be expected to slow the sampling process. In practice a small bias is accepted in exchange for being able to more efficiently sample the target distribution, although Brosse et al. [33] argue that the high variance of stochastic gradients can limit the usefulness of SGLD in practice.

In this work we propose to use, as in SGLD, additive noise (which has an adjustable but fixed strength) to stabilize the invariant measure of the stochastic dynamics. In contrast to SGLD however, we rely on underdamped Langevin dynamics and apply state-of-the-art discretization methods [157, 164], which introduce additive noise within a framework of second order stochastic dynamics. Additionally, we partition our algorithm based on the natural layer structure of the neural network.

For properties of the partitioned algorithms we draw on three recent works: (i) hypoelliptic properties of Langevin dynamics numerical methods [162], (ii) hypoelliptic properties for Langevin dynamics with configuration-dependent noise [243] and (iii) recent work on weighted-L^2 hypocoercivity of Adaptive Langevin dynamics [164]. To connect our methods with these works recall that our methods use additive noise and that in practice there will also be a second term with an unknown covariance arising from the gradient approximation. Such an approach is close to the systems treated in [243] where the SDEs take the form of a Langevin system where the friction matrix Γ is allowed to vary with position q (which in the results above corresponds to our parameters θ)

$$\begin{aligned} \mathrm{d}q &= p\ \mathrm{d}t, \\ \mathrm{d}p &= G(q)\mathrm{d}t - \Gamma(q)p\ \mathrm{d}t + \Sigma(q)\mathrm{dW}. \end{aligned} \tag{3.12}$$

Nondegeneracy of $\Sigma(q)$ is required for the results of [243] to hold, but we will obtain that by driving the system by additive noise of defined strength (in each momentum equation). In the case of the method AdLaLa (described in Section 3.3.3) which makes use of Adaptive Langevin (AdL) dynamics we have hypocoercivity results for AdL [164], which can be used to justify the method. These methods are based on position-independent noise. We conjecture that these hypocoercivity results can be extended to systems with position dependent noise.

3.1.4 Improving Stability of Neural Network Parameterization using Partitioned Stochastic Methods

In this work we make use of the layer structure of neural networks to obtain partitioned algorithms that use a different optimizer for different parts of the network. We show for certain datasets that these schemes can significantly accelerate training. There have been a number of attempts in recent years to design better training strategies by relying on the detailed structure of neural networks. For example the method AdaDelta [304] attempted to use an adaptive procedure to vary the learning rate (integration stepsize) according to dimension. Singh et al. [257] looked at using different stepsizes to treat the weights and biases in different layers. Although the method developed showed improvements compared to using the same stepsize, the gains were small. An effort which may be potentially more relevant to our work is the paper by Lan et al. [151] which found that freezing the last layer (i.e., fixing the weights and biases in the last layer) results in significant performance gain.

We are not aware of an effort to use differential thermostatting among layers in the design of training algorithms. In several of our experiments we found it advantageous to use low temperatures (even zero temperature) in the output layer but to maintain the hidden layer weights and biases at slightly elevated values. This means that those inner parameters can rapidly explore a wide range of low-loss states. We conjecture that it is this fluidity in the hidden layer which gives the LOL and AdLaLa methods described here their improved convergence speed.

It is well-known that local minima can be very sensitive to small changes in the choice of hyperparameters. This sensitivity has implications for the reliability and stability of training algorithms. Standard methods to improve stability of neural networks include L_1 and L_2 regularization a la Tikhonov [268, 288, 106]. In our experience, these methods cannot be relied upon to improve the test accuracy of a classifier, where the term "test accuracy" indicates how

many of the (during the training process unseen) test data points are correctly classified by the trained neural network.

We suggest that stochastic differential equations impose a different form of regularization, since the SDEs incorporate additive noise. A notable ramification is that thermodynamic parameterizations appear to give rise to classifiers whose level sets are relatively smooth compared to those produced by alternative methods. Thermodynamic parameterization thus effectively controls the distribution of weights–more precisely the distribution of the conjugate momenta associated to the weights, due to the statistical mechanical property known as equipartition of energy. By drawing parameter states from a sufficiently rich distribution of nearby candidate states, we show that the thermodynamic schemes produce smoother classifiers, improve generalization and reduce overfitting compared to traditional optimizers. In our studies of spiral and other datasets herein, we did not make use of any regularization method, which did not appear to affect our obtained test accuracies.

A benefit of using the thermodynamic parameterization approach as outlined here is to reduce the dependence of the training result on the initial conditions or the details of the mechanism of training. Unlike in conventional stochastic gradient descent and other schemes, the methods we advocate are formally ergodic, meaning that they have a unique stationary distribution and (almost all) trajectories converge to sampling paths for the same target distribution. This provides another way in which these schemes can improve robustness. Even if, in practice, we are unable to see the entire distribution due to computational limitations, it is desirable that the process can in principle be improved by continued exploration.

The per-step cost of our methods is (unless otherwise noted) roughly similar to that of the other training methods such as stochastic gradient descent and Adam, assuming the major cost of a timestep is dominated by the computation of the approximate gradient. We examine the relative performance of the different methods in a detailed series of numerical experiments. We also examine, again in numerical experiment, the key question of the variance of the results obtained by different methods, which points to the reliability and robustness of the schemes.

3.2 Langevin and Adaptive Langevin Schemes

In what follows, let $L(\theta)$ represent the overall loss defined in relation to the training dataset $\mathcal{D}$ (where we have suppressed the explicit dependence of the loss $L(\theta|\mathcal{D})$ on $\mathcal{D}$ for simplicity of notation). We suppose the loss to be piecewise smooth, Lipschitz continuous, for example as obtained using mean square error or cross entropy. We may augment the model by a (mild) quadratic regularization to ensure confinement of the parameters.

All algorithms considered here are based on gradients. We let $G(\theta) := -\nabla_\theta L$, i.e. the (full) negative gradient of the loss, and denote by $\tilde{G}$ the truncated negative gradient obtained by selecting a randomized (uniformly sampled) finite subset of the data $\tilde{D} \subset D$ at each timestep of fixed size. In all algorithms, the stepsize (learning rate) is denoted by h. The temperature parameter used in the thermodynamic algorithms is denoted by $\tau \geq 0$. R_n typically represents a vector of i.i.d. standard normal random numbers drawn at timestep n.

In this work we will primarily be concerned with the use of degenerate stochastic differential equations (SDEs) as the mechanism of parameterization. We may write these in the Itô formalism [75] as

$$\mathrm{d}Z = F(Z)\mathrm{d}t + \Sigma(Z)\mathrm{dW}_\mathrm{t}. \tag{3.13}$$

The degeneracy lies in the fact that Σ is not necessarily of full rank. This family of SDEs includes the underdamped and overdamped forms of Langevin (Brownian) dynamics. It also includes various thermostat methods such as Adaptive Langevin dynamics which is based on the stochastic generalization of the (deterministic) Nosé-Hoover thermostat.

3.2.1 Langevin Dynamics

Consider the Langevin dynamics [157] system of SDEs

$$\mathrm{d}\theta = p\,\mathrm{d}t, \tag{3.14a}$$
$$\mathrm{d}p = G(\theta)\,\mathrm{d}t - \gamma p\,\mathrm{d}t + \sqrt{2\gamma\tau}\,\mathrm{d}\mathrm{W}_t, \tag{3.14b}$$

where θ and p are the position and momentum vectors respectively, W_t is a standard N-dimensional Wiener process, γ is the friction hyperparameter, and $\tau > 0$ is the temperature hyperpameter. Under some mild assumptions Langevin dynamics is provably ergodic, which ensures that solutions of the dynamics sample the distribution $\rho_\tau(\theta, p)$ where

$$\rho_\tau(\theta, p) := \rho_\tau(\theta) \times N(p \,|\, 0,\, \tau) \propto \exp[-(L(\theta) + \|p\|^2/2)/\tau]. \tag{3.15}$$

A popular way of building discretization schemes for Langevin dynamics is via the use of splitting methods [157, 162]. Such schemes are developed by writing the vector field as an additive decomposition (a "splitting") into separate parts and solving for each piece in sequence. We shall use a Langevin splitting into pieces denoted A, B and O

$$\mathrm{d}\begin{bmatrix}\theta\\ p\end{bmatrix} = \underbrace{\begin{bmatrix}p\\ 0\end{bmatrix}\,\mathrm{d}t}_{A} + \underbrace{\begin{bmatrix}0\\ G(\theta)\end{bmatrix}\,\mathrm{d}t}_{B} + \underbrace{\begin{bmatrix}0\\ -\gamma p\,\mathrm{d}t + \sqrt{2\gamma\tau}\,\mathrm{d}W\end{bmatrix}}_{O}, \tag{3.16}$$

which, when taken individually, can be solved 'exactly' in its evolution of distribution [157]. Individual update maps of the splitting pieces are given by

$$\begin{aligned}
\mathcal{F}_h^A(\theta, p) &= (\theta + hp, p),\\
\mathcal{F}_h^B(\theta, p) &= (\theta, p + hG(\theta)),\\
\mathcal{F}_h^O(q, p) &= (\theta, e^{-\gamma h}p + \sqrt{\tau(1 - e^{-2\gamma h})}R).
\end{aligned} \tag{3.17}$$

We can code schemes by changing the order in which we apply the updates, giving for example the OBA scheme

$$\mathcal{F}_h^O \circ \mathcal{F}_h^B \circ \mathcal{F}_h^A, \tag{3.18}$$

or the BAOAB scheme

$$\mathcal{F}_{h/2}^B \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_h^O \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_{h/2}^B, \tag{3.19}$$

where repeated letters indicate substeps, i.e. two 'A's indicate that each should be a half step of size $h/2$. Using the update rules in (3.17) one obtains for the BAOAB scheme

$$\begin{aligned}
p_{n+1/2} &:= p_n + \frac{h}{2}G(\theta_n),\\
\theta_{n+1/2} &:= \theta_n + \frac{h}{2}p_{n+1/2},\\
\hat{p}_{n+1/2} &:= \alpha p_{n+1/2} + \sqrt{\tau(1-\alpha^2)}R_n, \quad \text{where } \alpha = e^{-\gamma h},\\
\theta_{n+1} &:= \theta_{n+1/2} + \frac{h}{2}\hat{p}_{n+1/2},\\
p_{n+1} &:= \hat{p}_{n+1/2} + \frac{h}{2}G(\theta_{n+1}).
\end{aligned} \tag{3.20}$$

The OBA splitting scheme (3.18) can be simplified to existing schemes in the machine learning literature through specific choices of the friction γ and the temperature τ hyperparameters. For example, taking $\gamma \to \infty$ gives the SGLD scheme (under reparameterization of the temperature and noise strength), whereas setting $\tau = 0$ (but using finite friction) gives a form of SGD with momentum. All such schemes which are of standard type are of low order of accuracy

and are relatively crude in their construction; in molecular dynamics it has been shown that schemes like BAOAB substantially improve on sampling accuracy. We thus look to the family of splitting-based methods (and further generalizations) to provided enhanced training strategies.

3.2.2 Role of Temperature

To clarify the role of temperature in parameterization of neural networks, we present in Figure 3.1 four classifiers for planar trigonometric data (see Section 3.4 for a full description of this dataset). Each classifier was obtained using Langevin dynamics and a single hidden-layer perceptron (SHLP) for a fixed amount of work, but was parameterized with different temperatures. Both the test accuracy and qualitative features of the classifier change with the temperature parameter, with results significantly improving as temperature increases. In further experiments we observed that further increases of the τ parameter can negatively affect the results, suggesting a 'Goldilocks' temperature region of optimal efficiency.

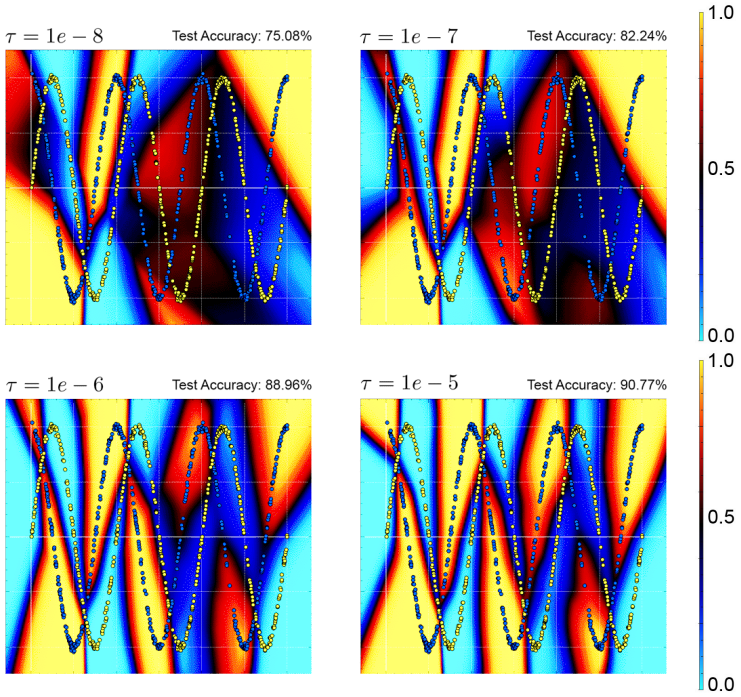

Figure 3.1: The figure shows classifiers computed using the BAOAB Langevin dynamics integrator. Visually, good classification is obtained if the contrast is high between the color of plotted data and the color of the classifier, thus indicating a clear separation of the two sets of labelled data points. The same stepsize ($h = 0.4$) and total number of steps $N = 50,000$ was used in each training run. The friction was also held fixed at $\gamma = 10$. A 500 node SHLP was used with ReLU activation, sigmoidal output and a standard cross entropy loss function. The temperatures were set to τ =1e-8 (upper left), τ =1e-7 (upper right), τ =1e-6 (lower left) and τ =1e-5 (lower right). The figures show that the classifier substantially improves as the temperature is raised. The test accuracies for each run are also shown at the top of each figure. The data is given by Eq. (3.33) with a = 3, b = 2 and c = 0.02. We used 1000 training, 1000 test data points and 2% subsampling.

A hypothetical model for the cause of the performance gain due to elevated temperature might be found by considering molecular diffusion on a rough energy landscape [314, 225]. In a corrugated energy surface and at zero temperature the system will likely get stuck in local minima, lacking the required energy to overcome barriers blocking movement between states. Increasing the temperature allows weak interaction with a heat bath, randomly introducing energetic fluctuations into the system that can move it over barriers and away from local minima. The size of the fluctuations can be related to the temperature parameter: too small

and it will take a long time to cross barriers, whereas too large and the system will not be drawn towards the global minimum.

3.2.3 Adaptive Langevin and the Nosé-Hoover Thermostat

Adaptive Langevin dynamics (AdL) is a method in which the friction parameter of Langevin dynamics is considered as a dynamical variable. The method derives from Nosé-Hoover dynamics developed by S. Nosé and W. Hoover in the early 1980s. Their proposal was to use a deterministic system to sample from the canonical ensemble [216, 109]. The Adaptive Langevin method, which incorporates additive noise, was first elucidated in [128] and has since been employed in a variety of multiscale modelling and noisy gradient settings [54]. Analyses of this method can be found in [161, 99, 164]. The Adaptive Langevin equations are given by degenerate SDE system

$$\begin{aligned} \mathrm{d}\theta &= p\ \mathrm{d}t, \\ \mathrm{d}p &= G(\theta)\mathrm{d}t - \xi p\ \mathrm{d}t + \sigma_{\mathrm{A}}\mathrm{dW}_t, \\ \mathrm{d}\xi &= \varepsilon\left(p^T p - N\tau\right)\mathrm{d}t. \end{aligned} \tag{3.21}$$

Various discretization schemes are obtained by breaking up the AdL system into pieces (as in the discretization of Langevin dynamics) and solving the parts separately. Apart from using the same maps A and B as for Langevin dynamics, we define additional maps $\mathrm{C}, \mathrm{D}, \mathrm{E}$ by

$$\begin{aligned} \mathcal{F}_h^A(\theta, p, \xi) &= (\theta + hp, p, \xi), \\ \mathcal{F}_h^B(\theta, p, \xi) &= (\theta, p + hG(\theta), \xi), \\ \mathcal{F}_h^C(\theta, p, \xi) &= (\theta, \exp(-h\xi)p, \xi), \\ \mathcal{F}_h^D(\theta, p, \xi) &= (\theta, p + \sigma_{\mathrm{A}}\sqrt{h}R, \xi), \\ \mathcal{F}_h^E(\theta, p, \xi) &= (\theta, p, \xi + h\varepsilon\left[p^T p - N\tau\right]). \end{aligned} \tag{3.22}$$

Several choices of ordering are possible. We shall use the composition BACDEDCAB

$$\mathcal{F}_{h/2}^B \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_{h/2}^C \circ \mathcal{F}_{h/2}^D \circ \mathcal{F}_h^E \circ \mathcal{F}_{h/2}^D \circ \mathcal{F}_{h/2}^C \circ \mathcal{F}_{h/2}^A \circ \mathcal{F}_{h/2}^B. \tag{3.23}$$

3.3 Partitioned Discretization Algorithms for Neural Networks

In layered or hierarchical models, e.g. deep neural networks, we have a natural partitioning of the parameter vector according to its role in the hierarchy. It may be useful to treat the parameters at different levels of the hierarchy differently in the parameterization process. In particular, it is possible that, either due to design or some feature of the network, the characteristics of the gradient noise introduced at different layer depths may differ, and it is then natural to design a method that treats the components independently. Lan et al. [151] observed that freezing the last layer of a neural network (while using SGD with momentum for the flexible components) can enhance the performance of training algorithms. We draw on this idea here for motivation in developing a family of partitioned algorithms that can be used to train neural networks.

In this chapter we focus on single hidden layer perceptrons, for which we shall use a two-part partitioning. Let $\theta = (\theta^{(1)}, \theta^{(2)})$ be a partitioning of the full parameter vector, and assume a similar partitioning of the momenta $(p^{(1)}, p^{(2)})$ as well as of the Wiener process $W(t)$. The partitioning can be defined in various ways. The approach we have taken in our experiments is to group together the weights and biases at each layer

$$\theta^{(i)} = (w^{(i)}, b^{(i)}), \qquad i = 1, 2. \tag{3.24}$$

In extending this framework to deep neural networks one could also include several layers (or all hidden layers, say) as one part of the partitioning.

We next describe a number of different families of partitioned integrators which could be used to take advantage of the layer structure.

3.3.1 Langevin in Layers

The simplest idea is to use different Langevin parameters in different layers (or alternatively, Langevin with an anisotropic diagonal friction matrix). Since temperature is purely formal in machine learning, we can, without concern for physical meanings, introduce an artificial temperature gradient by using different temperatures τ_1, τ_2 in the different layers. Meanwhile, we do keep the learning rate fixed at the same value across all layers and throughout training. The equations then become

$$\begin{aligned} \mathrm{d}\theta^{(i)} &= p^{(i)}\mathrm{d}t, \\ \mathrm{d}p^{(i)} &= \tilde{G}^{(i)}(\theta)\mathrm{d}t - \gamma_i p^{(i)}\mathrm{d}t + \sqrt{2\tau_i\gamma_i}\mathrm{dW}^{(i)}, \end{aligned} \tag{3.25}$$

where the indices $i = 1, 2$ represent the different layers. Each subsystem can be propagated using BAOAB or some other Langevin integrator.

3.3.2 Langevin-Overdamped Langevin (LOL) and Dissipated Hamiltonian Stochastic Gradient Dynamics (DHSGD)

Consider a partitioned two-part model on which we use BAOAB. Taking the friction to infinity in the last layer, namely taking the limit $\gamma_2 \to \infty$ results in an alternative method with a strong stabilizing property. The equations become

$$\begin{aligned} p^{(1)}_{n+1/2} &= p^{(1)}_n + \frac{h}{2}\tilde{G}^{(1)}(\theta_n), & \theta^{(1)}_{n+1} &= \theta^{(1)}_{n+1/2} + \frac{h}{2}\hat{p}^{(1)}_{n+1/2}, \\ p^{(2)}_{n+1/2} &= p^{(2)}_n + \frac{h}{2}\tilde{G}^{(2)}(\theta_n), & \theta^{(2)}_{n+1} &= \theta^{(2)}_{n+1/2} + \frac{h}{2}\hat{p}^{(2)}_{n+1/2}, \\ \theta^{(1)}_{n+1/2} &= \theta^{(1)}_n + \frac{h}{2}p^{(1)}_{n+1/2}, & p^{(1)}_{n+1} &= \hat{p}^{(1)}_{n+1/2} + \frac{h}{2}\tilde{G}^{(1)}(\theta_{n+1}), \\ \theta^{(2)}_{n+1/2} &= \theta^{(2)}_n + \frac{h}{2}p^{(2)}_{n+1/2}, & p^{(2)}_{n+1} &= \hat{p}^{(2)}_{n+1/2} + \frac{h}{2}\tilde{G}^{(2)}(\theta_{n+1}). \\ \hat{p}^{(1)}_{n+1/2} &= \alpha p^{(1)}_{n+1/2} + \sqrt{\tau_1(1-\alpha^2)}R^{(1)}_n, & \text{where } \alpha &= e^{-\gamma_1 h}, \\ \hat{p}^{(2)}_{n+1/2} &= \sqrt{\tau_2}R^{(2)}_n, \end{aligned} \tag{3.26}$$

We refer to this method as Langevin-Overdamped Langevin or LOL. The motivation for this scheme is that it gives the possibility to increase the exploration of hidden layer structure (including the weights and biases defining the dependence on the input) while incorporating a strong dissipation in the connection to the output layer (which provides a strong stabilizing property). In most of our experiments we also set the temperature of the secondary partition to be zero, i.e. we set $\tau_2 = 0$. In this scenario one may also interpret the combined method as a sort of free energy minimization of the output layer weights and biases.

An extreme form of LOL arises from relying only on the overdamped component to dissipate the gradient noise. In that case we can set $\gamma_1 = 0$ which removes any thermostatting from the first part and gives it the form of a Hamiltonian system. If we set $\tau_2 = 0$ and after some rearrangements, we obtain the simplified equations

$$p^{(1)}_{n+1/2} = p^{(1)}_n + \frac{h}{2}\tilde{G}^{(1)}(\theta_n), \tag{3.27}$$

$$\theta^{(1)}_{n+1} = \theta^{(1)}_n + hp^{(1)}_{n+1/2}, \tag{3.28}$$

$$\theta^{(2)}_{n+1} = \theta^{(2)}_n + \frac{h^2}{2}\tilde{G}^{(2)}(\theta_n), \tag{3.29}$$

$$p^{(1)}_{n+1} = p^{(1)}_{n+1/2} + \frac{h}{2}\tilde{G}^{(1)}(\theta_{n+1}). \tag{3.30}$$

This method, which we refer to as Dissipated Hamiltonian Stochastic Gradient Dynamics (DHSGD), can be viewed as the combination of the Verlet integrator widely used in molecular dynamics with the stochastic gradient descent method. DHSGD only requires the tuning of a single parameter, the stepsize, as is the case for SGD. It is important to note the appearance of a squared stepsize in the SGD component, reflecting the different timescales associated to first and second order dynamics methods.

It is interesting to ask about the properties of this method in the case that the gradient is exact, i.e. when $\tilde{G} \equiv G$ in (3.27)-(3.30). At a fixed point ($\theta = (\theta^{(1)}, \theta^{(2)})$) of DHSGD we have $\theta_n^{(2)} \equiv \theta_{n+1}^{(2)} \equiv \theta^{(2)}$ and thus

$$G^{(2)}(\theta) = 0.$$

Moreover, from $\theta_n^{(1)} = \theta_{n+1}^{(1)}$ in (3.28), we see $p_{n+1/2}^{(1)} = 0$ and consequently

$$p_{n+1}^{(1)} = p_n^{(1)} + \frac{h}{2}\left[G^{(1)}(\theta_n) + G^{(1)}(\theta_{n+1})\right]$$

which implies

$$G^{(1)}(\theta) = 0.$$

Thus the fixed points of DHSGD are, encouragingly, stationary points of the noiseless gradient system. With respect to their attractive nature, the situation is less obvious. For $h \to 0$, due to the h^2 dependency in (3.29), we can think of $\theta_n^{(2)}$ as approximately fixed at say $\bar{\theta}^{(2)}$ in (3.27), (3.28), (3.30). This is then a symplectic (leapfrog) discretization of a Hamiltonian system with Hamiltonian

$$H(\theta^{(1)}, p^{(1)}) = \|p^{(1)}\|^2/2 + L(\theta^{(1)}, \bar{\theta}^{(2)}),$$

meaning that it is not at all dissipative as $h \to 0$. Even if, for small h, the system would seem to make incremental loss-reducing steps via the slow gradient descent component, it would seem unwise to rely on this weak mechanism of dissipation.

On the other hand, for large h, the dissipative property of (3.27)-(3.30) is entirely unclear since the discretization then differs significantly from the original dynamics.

3.3.2.1 Contractive variant of DHSGD

As a natural alternative which corrects the dissipation issue at small stepsize, one could simply replace the factor $h^2/2$ in (3.29) by h. It is then as if we have directly discretized the dynamics

$$\begin{aligned} \dot{\theta}^{(1)} &= p^{(1)} \\ \dot{p}^{(1)} &= \tilde{G}_1(\theta) \\ \dot{\theta}^{(2)} &= \tilde{G}_2(\theta) \end{aligned}$$

This system dissipates the energy function

$$H(\theta, p^{(1)}) = \|p^{(1)}\|^2/2 + L(\theta).$$

We refer to this method as DHSGDc (contractive). Note that, as described, this becomes again a first order method, as SGD, since we would essentially solve for the second set of variables by the forward Euler method. If desired, the Euler step could be replaced by a 2nd order explicit scheme (e.g. Heun's method).[2] The only potential advantage is that it seems to allow some additional exploration of the first partition, relative to the rapid descent in the second, albeit without benefit of a temperature control to regulate this process.

Although intriguing for their simplicity and lack of parameters, the above deterministic schemes are likely to suffer from similar issues of trapping as SGD or Adam for the problems considered in this work.

[2]The use of implicit schemes such as implicit midpoint, is certainly not realistic in the setting of large scale neural computation.

3.3.3 Adaptive Langevin and Langevin in Layers (AdLaLa)

The Adaptive Langevin method (AdL) (Section 3.2.3) has the property that it can automatically maintain a target temperature in a system driven by Gaussian noise. While the gradient noise encountered in statistical approximation is not, by any means, Gaussian, it may have an important Gaussian component that can be controlled using this device (as observed in practice, see [54, 161]). We therefore consider a modification of the Langevin in layers scheme in which the Adaptive Langevin method is used to manage the sampling of part of the system, thus extracting accumulated heat due to gradient noise.

Applying Adaptive Langevin (AdL) in layers leads to the system, for $i = 1, \ldots, d$:

$$\begin{aligned}
\mathrm{d}\theta^{(i)} &= p^{(i)}\mathrm{d}t, \\
\mathrm{d}p^{(i)} &= \tilde{G}^{(i)}(\theta)\mathrm{d}t - \varepsilon_i \xi^{(i)} p^{(i)}\mathrm{d}t + \sigma_{\mathrm{A},i}\mathrm{dW}_{\mathrm{A}}^{(i)}, \\
\mathrm{d}\xi^{(i)} &= \varepsilon_i \left[\|p^{(i)}\|^2 - N_i\tau_i\right]\mathrm{d}t.
\end{aligned}$$

The parameters for layer i are the coupling coefficient ε_i, the temperature parameter τ_i, and the applied noise amplitude $\sigma_{\mathrm{A},i}$. Discretization then proceeds as for AdL using the variant described in Section 3.2.3 or some other scheme.

It is also possible to have a partitioned algorithm with some components treated using Adaptive Langevin and others using a Langevin scheme. As a simple instance of such a method, consider the two-part "AdLaLa" partitioning:

$$\begin{aligned}
\mathrm{d}\theta^{(1)} &= p^{(1)}\mathrm{d}t, \\
\mathrm{d}p^{(1)} &= \tilde{G}^{(1)}(\theta)\mathrm{d}t - \varepsilon_1 \xi^{(1)} p^{(1)}\mathrm{d}t + \sigma_{\mathrm{A}}\mathrm{dW}_{\mathrm{A}}^{(1)}, \\
\mathrm{d}\xi^{(1)} &= \varepsilon_1 \left(\|p^{(1)}\|^2 - N_1\tau_1\right)\mathrm{d}t, \\
\mathrm{d}\theta^{(2)} &= p^{(2)}\mathrm{d}t, \\
\mathrm{d}p^{(2)} &= \tilde{G}^{(2)}(\theta)\mathrm{d}t - \gamma_2 p^{(2)}\mathrm{d}t + \sqrt{2\tau_2\gamma_2}\mathrm{dW}_{\mathrm{A}}^{(2)}.
\end{aligned} \tag{3.31}$$

Again for the discretization we keep the learning rate fixed at the same value across all layers and throughout training. In the extreme case, where $\tau_2 = 0$ the second part can be viewed as a dissipated gradient system and thus we may think of this as analogous to gradient descent with momentum, but the adaptive control of the first subsystem may provide greater flexibility in the approach to the overall minimum.

3.4 Model Problems for Classification

We examine parameterization of fully connected single hidden-layer neural networks with ReLU activation in the context of binary classification of spiral and trigonometric data, as well as the MNIST dataset. We found that the results were significantly different for the different problem classes, with MNIST data showing fewer substantial differences among schemes. In order to cast some light on this issue, we use the technique of 1D linear interpolation proposed by Goodfellow et al. [87] and a surface plotting technique [120].

The spiral datasets we use in this work are generated from the formulas

$$\begin{aligned}
x_1 &= at^p \cos(2bt^p\pi) + c\mathcal{N}(0,1), \\
x_2 &= at^p \sin(2bt^p\pi) + c\mathcal{N}(0,1).
\end{aligned} \tag{3.32}$$

In these formulas, t is drawn repeatedly from $\mathcal{U}(0,1)$ to generate data points, where $\mathcal{U}$ is the uniform distribution. This creates one arm of the dataset, to which we assign class label 0. The other arm constitutes a shift in the argument of the trig functions by π. We typically set $a = 2, p = 0.5$ and $c = 0.02$, unless otherwise indicated. When we vary b, this directly affects the number of turns of the spiral and therefore the complexity of the problem. In the

trigonometric dataset the data is given for class 0 by

$$x_1 = at, x_2 = \cos(bt\pi) + c\mathcal{N}(0,1). \tag{3.33}$$

Data for class 1 is generated by the same equations but with cosine replaced by sine. Typical classification data are shown in Figure 3.2.

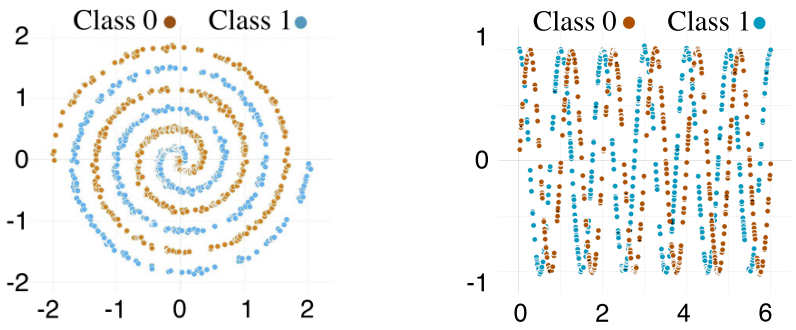

Figure 3.2: Spiral data and trigonometric data typical of those used in our classification studies.

Depending on the choice of parameters, these can be challenging test cases for classification, due to the consequent structure of the loss landscape. Methods such as SGD and Adam, which, up to gradient noise, monotonically decrease the loss, can easily become trapped in unsuitable states or be slowed down by the presence of saddle points for these types of datasets. By contrast MNIST data and related image classification problems may be relatively free of these issues. This is supported by results from Ballard et al. [15] who show (using molecular potential energy landscape visualization techniques) that the obtained landscape for MNIST is single-funnel-like, with only small barriers separating the different local minima from the global minimum. In contrast, they observe large barriers in the landscape for a non-linear regression problem, thus demonstrating that there exist fundamental differences in the structure of the loss landscape for different training problems. In Huang et al. [114] they illustrate this by designing a problem that standard optimizers will find challenging. They set-up a binary classification problem, where they pinch the margin between two rings of datapoints, which causes any good minimizer to be "sharp". The small volume of the corresponding basin makes these minima less likely to be found by standard optimizers. Below we include our approach to demonstrate the difference between the spiral and MNIST datasets, drawing on a method proposed by Goodfellow et al. [87].

1D Linear Interpolation: Denote the initial parameter configuration by θ_0 and the parameter configuration after running the optimizer by θ_f. Define

$$\theta^*(\alpha) = (1-\alpha)\theta_0 + \alpha\theta_f, \quad \alpha \in [0,1]. \tag{3.34}$$

We graph the loss $L(\theta^*(\alpha))$ as a function of α. At $\alpha = 1$ the loss is small, while at $\alpha = 0$ the loss is at a random state.

For MNIST (Figure 3.3) our results are similar to the findings of Goodfellow et al. [87], specifically they observe, "We find that the objective function has a simple, approximately convex shape along this cross-section. In other words, if we knew the correct direction, a single coarse line search could do a good job of training a neural network".

By contrast, in the spiral dataset (with more than 1 turn, i.e., $b > 1$ in Eq. (3.32)) we observe that there is typically a barrier between the loss at the initial parameter configuration θ_0 and the loss at the parameter configuration found by the optimizers (see Figure 3.4). The barrier appears to consistently be significantly higher between θ_0 and the θ_f that AdLaLa finds, than between θ_0 and the θ_f that Adam finds (θ_0 is the same for both methods). This seems to suggest that AdLaLa finds different kinds of minima compared to Adam, which generally have lower test loss. For the trigonometric dataset, the obtained curves were generally similar

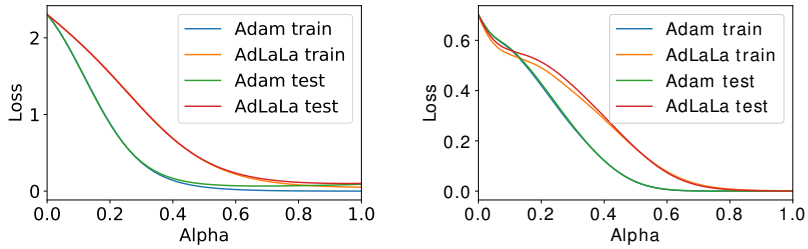

Figure 3.3: Left: graph of the loss along the line (3.34) for the MNIST dataset. It is clear that AdLaLa and Adam converge to different minima, although we used the exact same initialization for both methods. There is no evidence of a loss-barrier. Their final test loss is similar. Right: the same construct for a simple spiral with one turn, i.e., $b = 1$ in Eq. (3.32). As for MNIST there is no evidence of a loss-barrier.

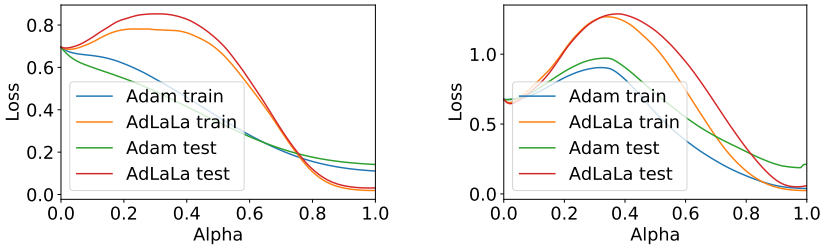

Figure 3.4: The left and right plots are for two runs with the same parameters but different initializations. We train a 20 node SHLP on the two turn spiral dataset, i.e., $b = 2$ in Eq. (3.32), for 20,000 steps, with 500 training and test data points and 5% subsampling. Left: The parameterization that AdLaLa finds gives: 100% train, 99% test. Adam gets: 88% train, 91% test; Right: AdLaLa: 100 % train, 98 % test. Adam: 96% train, 94 % test.

to those for the spirals-2turns problem, although the height of the barrier is typically lower. We emphasize that these plots do not represent the actual path that the optimizer traverses, but do seem to point at a significant difference in the loss landscape structure of the MNIST vs. spiral/trigonometric datasets. We will elaborate on this point by constructing some surface plots.

Surface plots: We will explore 2-dimensional cross-sections in the loss landscape. Denote the initial parameter configuration by θ_0 and now run the optimizer twice to obtain two distinct minima: $\theta_{f,1}$ and $\theta_{f,2}$.

$$\begin{aligned} F_1 &= \alpha(\theta_{f,1} - \theta_0) + \theta_0, \\ F_2 &= \alpha(\theta_{f,2} - \theta_0) + \theta_0, \\ \theta^*(\alpha, \beta) &= \beta F_1(\alpha) + (1-\beta)F_2(\alpha), \alpha \in [0,1], \beta \in [0,1]. \end{aligned}$$

So when

- $\alpha = 0$: $F_1 = \theta_0$ and $F_2 = \theta_0$. This implies that $\theta^* = \theta_0$ if $\alpha = 0$ and $\forall \beta \in [0,1]$. So the loss should be relatively high there, as it is the loss for a random initialization of the neural network parameters.
- $\alpha = 1$: $F_1 = \theta_{f,1}$ and $F_2 = \theta_{f,2}$, so the loss minima are given by $(\alpha = 1, \beta = 0)$ and $(\alpha = 1, \beta = 1)$.

We observe in Figure 3.5 that for the MNIST dataset there is a consistent monotonic decline in loss along the line from the initial parameterization to the final parameterization. However, for the 2-turn spirals we frequently observe loss landscapes with saddle points in the cross-sectional plane. This seems to indicate a fundamental difference in the nature of these problems

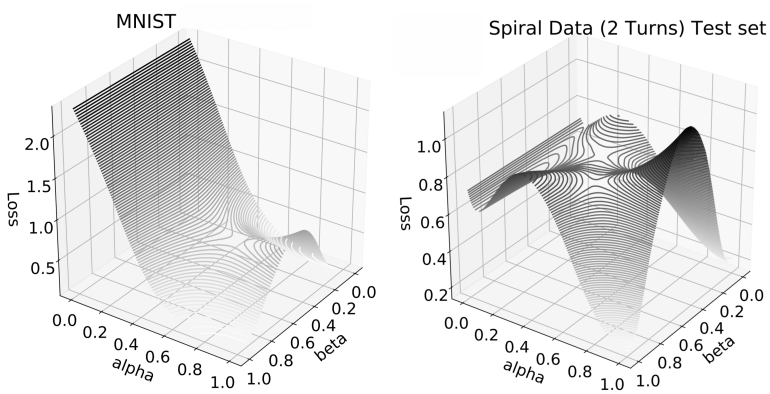

Figure 3.5: MNIST (left) vs. Spirals (2-turn) (right) on test data.

and the flexibility of the optimizers required to tackle them. We note that our low-dimensional intuitions often do not translate to the high-dimensional case: critical points with high error are exponentially likely to be saddle points, rather than local minima, which means that saddle points are thought to be the more likely cause of a possible impediment of optimization [49].

3.5 Properties of the Thermodynamic Parameterization Methods: Ergodicity, Equipartition and Smooth Classifiers

The principles of thermodynamics and the theory of hypoelliptic diffusion underpin the stochastic integrators that we have proposed previously in this chapter. The conditions for an SDE system to be ergodic are discussed in numerous recent works. We summarize these as used in recent studies of ergodic properties of Langevin and generalized Langevin equations.

Consider the Langevin system (3.14). The starting point for analysis of SDEs is the Fokker-Planck equation [75]

$$\frac{\partial \rho}{\partial t} = \mathcal{L}^* \rho, \tag{3.35}$$

where

$$\mathcal{L}^* \rho = -\nabla_\theta \cdot (p\rho) + \nabla_p \cdot ([-G(\theta) + \gamma p]\, \rho) + \gamma \tau \Delta_p \rho, \tag{3.36}$$

where Δ_p is the Laplacian in the momenta components only

$$\Delta_p = \sum_{i=1}^{N} \frac{\partial^2}{\partial p_i^2}. \tag{3.37}$$

Assuming G is smooth it is possible to find conditions which ensure that the system is ergodic in a weighted L^∞ space; this is the usual approach based on Harris chains that one finds described in detail in the excellent book of Meyn and Tweedie [201]. For Langevin dynamics, the analysis was first carried out in detail in [195]. More recently, an alternative framework has become available which is in many ways more directly suited to applications of SDEs to machine learning. This is the method described in the work of Dolbeault et al. [56], which allows the derivation of exponential convergence rates when the Fokker-Planck operator is considered in a suitable subspace of $L^2(\mu_\tau)$, i.e. weighted by the canonical invariant measure μ_τ. The method can be shown to give convergence estimates for underdamped Langevin dynamics.

In recent work, the same framework was applied to the Adaptive Langevin dynamics system [164]. The power of L^2 estimates is that they can be used to establish a Central Limit Theorem which is important in statistical applications.

Although we have not yet looked in detail at hypocoercivity for the more complicated partitioned methods discussed here such as AdLaLa, LOL etcera, we expect that the weighted L^2 approach as used for AdL in [164] could be applied to these systems as well, in order to establish the ergodic property. By contrast, for any of the deterministic schemes mentioned and for schemes relying solely on gradient noise, ergodicity is unlikely to hold and we are unaware of any mathematical technique that could be used for their analysis. When additive noise is combined with gradient noise, assuming enough boundedness, a unique invariant measure still can be shown to exist using weighted L^∞ techniques [243].

With regard to the discretized systems with additive noise, it seems likely that similar ergodic estimates can be formulated and proved. For example, Leimkuhler et al. [162] studied the ergodic properties of Langevin splitting integrators such as BAOAB on weighted L^∞ spaces.

Equipartition Property. One of the most powerful consequences of ergodicity is *equipartition of energy* which simply states that the mean kinetic energy of all degrees of freedom, in thermal equilibrium, is constant. This property can easily be derived by leveraging the uniqueness of the stationary distribution and then through direct integration of the Gibbs density, that is, for each i,

$$\frac{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} p_i^2 \exp\left(-\tau^{-1}\left[\sum_{i=1}^{N} p_i^2/2 + L(\theta)\right]\right) \mathrm{d}^N p \; \mathrm{d}^N \theta}{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \exp\left(-\tau^{-1}\left[\sum_{i=1}^{N} p_i^2/2 + L(\theta)\right]\right) \mathrm{d}^N p \; \mathrm{d}^N \theta} = \tau. \tag{3.38}$$

We confirmed experimentally that the magnitude of the squared momenta are approximately controlled by the set temperature value in AdLaLa and LOL. Because of equipartition we are assured that every weight will be driven directly by a momentum coordinate which has a Gaussian distribution. While we cannot predict the distribution of the weights themselves, since the full complement of weights are coupled intricately through the network structure, we can be sure that they will explore the full available configuration space. Even if small, all weights should be active during training using a thermodynamic method.

Weight Distributions. We observe fundamental differences in the parameterizations obtained by thermodynamic parameterization methods, such as SGLD and AdLaLa, compared to standard optimizers, such as SGD and Adam. We shall illustrate this by plotting the evolution of the obtained weights and biases over time for both the spirals 2-turn dataset (see Figure 3.6) and the complicated spirals 4-turn dataset (see Figure 3.7). We use a SHLP with 500 nodes and ReLU activation, 1000 training data points and 2% subsampling. We distinguish between two sets of weights: those linking the input layer to the hidden layer, weights1 (first row), and those linking hidden layer to output, weights2. We also show the distribution of biases in the hidden layer (second row). We do not show weights2, as their distribution is very similar to those of weights1. Weights2 do typically assume a larger values than weights1, but this is the same for all methods evaluated here.

The thermodynamic parameterization methods rapidly excite a large amount of parameters. This is clearly visible by comparing the obtained weight/bias distributions after a mere 50 steps (dark green colour in the figures) for the different methods. For the easier 2-turn spirals dataset (see Figure 3.6), minima are easier accessible and fewer nodes are required to obtain a good classification, which leads SGD and Adam to be able to find good minima without exciting all the weights and biases. For the complicated 4-turn spirals dataset however, AdLaLa makes much faster headway towards high test accuracies (see Figure 3.7), whereas Adam and SGD appear to be stuck in a parameterization with many small weights/biases. We observe that although SGLD consistently assigns much larger values to the parameters it obtains than AdLaLa, this does not appear to be beneficial for its performance on the test dataset.

As Figure 3.6 and 3.7 only show the obtained parameter distributions of a single run of the optimizers, we will now validate that these results are consistent over many different runs. To do so we plot all the weights obtained over 100 different runs into one histogram (see Figure 3.8). This shows the overall trend of the parameter distributions. To obtain these results we

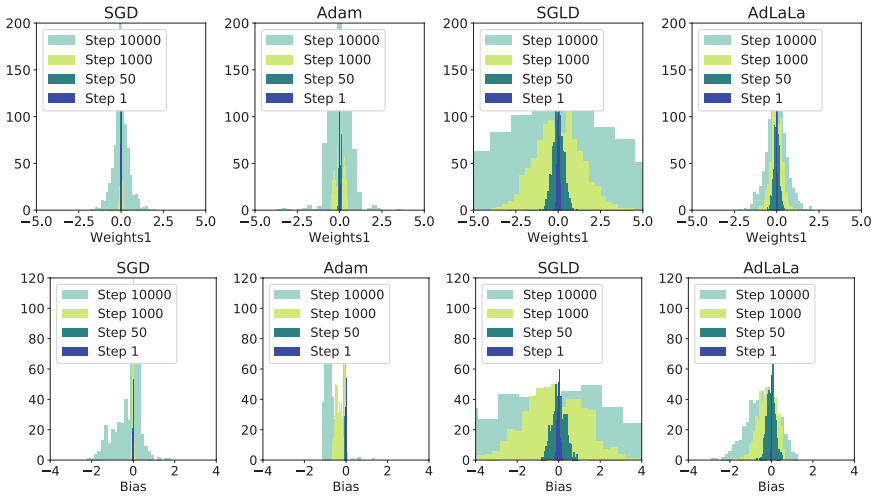

Figure 3.6: Weight and bias distributions for the 2-turn spirals dataset at different times and for different methods. Parameter settings: $h_{\text{SGD}} = 0.2, h_{\text{Adam}} = 0.005$, SGLD: $h_{\text{SGLD}} = 0.1$ and $\sigma_{\text{SGLD}} = 0.01$. AdLaLa: $h_{\text{AdLaLa}} = 0.25, \sigma_A = 0.01$, $\tau_1 = \tau_2 = 10^{-4}, \epsilon = 0.1$ and $\gamma = 0.5$. Test accuracy at step 50: 0.66 (SGD), 0.65 (Adam), 0.61 (SGLD), 0.62 (AdLaLa); at step 1000: 0.66 (SGD), 0.89 (Adam), 0.68 (SGLD), 0.82 (AdLaLa); at step 10000: 0.96 (SGD), 0.99 (Adam), 0.74 (SGLD), 0.99 (AdLaLa).

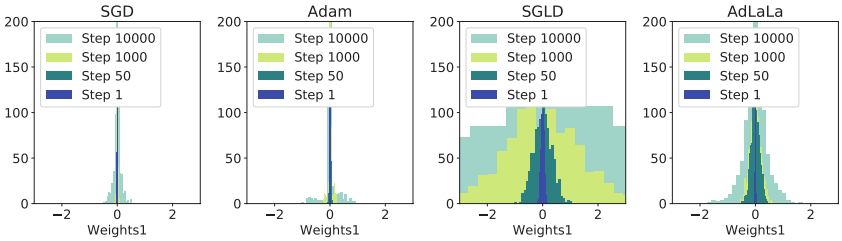

Figure 3.7: Evolution of weights for the 4-turn spiral problem. Same parameter settings as in Figure 3.6, but $\gamma = 0.1$ in AdLaLa. Test accuracy at step 50: 0.5 (SGD), 0.58 (Adam), 0.52 (SGLD), 0.45 (AdLaLa); at step 1000: 0.56 (SGD), 0.55 (Adam), 0.5 (SGLD), 0.62 (AdLaLa); at step 10k: 0.58 (SGD), 0.67 (Adam), 0.54 (SGLD), 0.8 (AdLaLa).

used a SHLP with 20 nodes, 500 training data and 5% subsampling, for the spirals 2-turn dataset.

It is clear that SGD and Adam obtain parameterizations which have many (close to) zero weights and biases. The same was observed for different stepsizes and different batchsizes. In SGLD and AdLaLa most weights and biases appear to be equally activated. We suggest that this is a consequence of the ergodicity and equipartition property of the latter methods. We also note that for most optimizers their obtained layer-2 weights tend to be larger than layer-1 weights, but this changes if one increases the γ parameter in the AdLaLa method. We note that for the LOL method (not shown in the figure) weights can take on both very small and very large values; in particular layer-1 weights and biases can take on values of the order 10^2. This may indicate a possible instability and appears to be linked to larger classifier gradients.

Out of the 100 runs we also compared the parameter distributions for the run with the worst test accuracy vs. the run with the best test accuracy. We observe that Adam performs worse if a larger percentage of the weights and biases are zero. The same holds for LOL, although the difference in accuracies is less dramatic between the worst and best run (10% difference in test accuracy for LOL, 35% for Adam). For AdLaLa there is even less variation in the accuracies

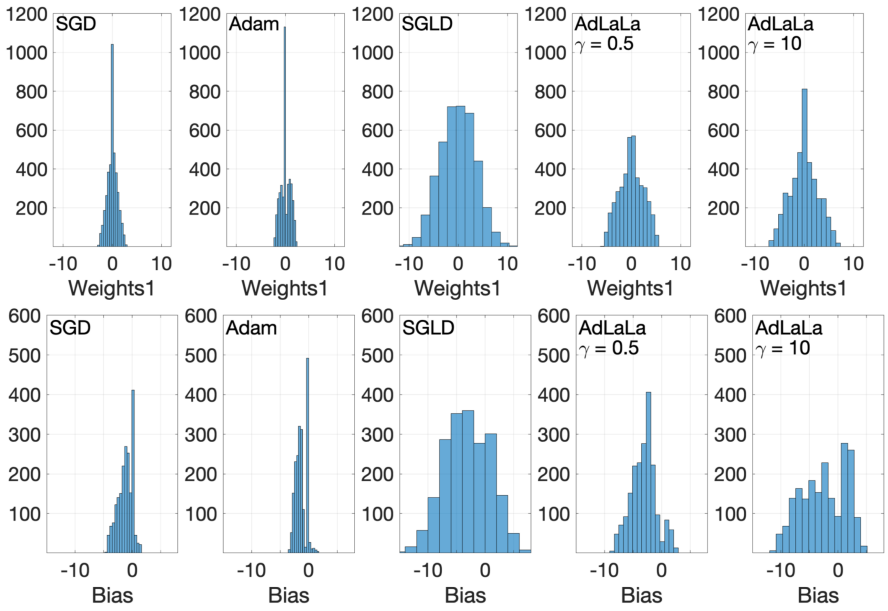

Figure 3.8: Obtained parameter distributions over 100 runs after using different optimizers for the 2-turn spiral problem for 10K steps. Parameter settings: $h_{\text{SGD}} = 0.1, h_{\text{Adam}} = 0.005, h_{\text{SGLD}} = 0.1, \sigma_{\text{SGLD}} = 0.1$, AdLaLa has $h_{\text{AdLaLa}} = 0.25, \tau_1 = \tau_2 = 10^{-4}, \sigma_A = 0.01, \epsilon = 0.1$, $\gamma = 0.5$ (left) and $\gamma = 10$ (right). Average test accuracies: SGD: 79%, Adam: 83.7%, SGLD: 78%, AdLaLa ($\gamma = 0.5$): 93.4%, AdLaLa ($\gamma = 10$): 85.5%.

obtained and the weights appear to be always approximately equally distributed around zero.

3.6 Numerical Studies with Thermodynamic Parameterization Methods

Tests of the various methods were conducted using three separate codes for cross-validation and verification of consistency:

- We used a custom PyTorch-based [219] system [version 1.0.0].
- We implemented the schemes into the latest version of the DLIB package [138] written in C++.
- We created a custom native C++/QT application to perform rapid visual exploration of the training algorithms.

Code that implements the algorithms described in this chapter is available on github.com/TiffanyVlaar/ThermodynamicParameterizationOfNNs.

Choice of Hyperparameters. Despite the high accuracy and rapid convergence of our partitioned schemes (as we illustrate below), a practitioner may consider the relatively large amount of hyperparameters of these methods to be a disadvantage. We wish to emphasize that one can use certain rules of thumb to select the values of these hyperparameters, which significantly reduces the work required in tuning. Additionally, we note that our methods appear less sensitive to the choice of initialization (see Figure 3.14 and corresponding discussion) or the subsampling batchsize, which can be viewed as reducing another aspect of "tuning" and is thus a major performance gain compared to e.g. SGD or Adam. We also do not change the stepsize (learning rate) throughout training and are still able to obtain great performance using our methods.

As rule of thumb for AdLaLa, one can typically set the temperatures of all layers to the range $[10^{-4}, 10^{-6}]$, the coupling coefficient $\epsilon \in [0.05, 0.1]$, additive noise $\sigma_A \in [10^{-2}, 10^{-4}]$ and obtain good performance. In some cases there was an advantage to using a lower temperature for the output layer. The value of γ (associated to the output layer) appears to be linked to the stepsize. We recommend $\gamma \in [0.05, 5]$, in typical cases. In some of our tests much higher values were used with good effect, whereas smaller values typically lead to stepsize restriction. Generally, we can use stepsizes for AdLaLa which are similar to or even larger than those for SGD or SGLD, but for some of the harder problems the stepsize needed to be modestly reduced. For all spirals examples using a SHLP, the following parameter choices work well: AdLaLa: $h = 0.15, \tau_1 = \tau_2 = 10^{-4}, \gamma = 0.1, \sigma_A = 0.01, \epsilon = 0.1$. For LOL, there are fewer parameters to set; good choices appear to be $\gamma_1 = 0.01$ and $\tau_1 = 10^{-3}$ for a SHLP. In experiments with Adam we used the hyperparameters recommended in the original article [139], namely $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon_{\text{adam}} = 10^{-8}$, and did not change the learning rate throughout training. Results are presented for optimal values of the stepsize (for Adam, SGD, and SGLD), momentum (for SGD with momentum), and additive noise strength (for SGLD).

Comparison of Classifiers. The enhanced performance of AdLaLa vs Adam for the difficult 4-turns spiral dataset can be seen by comparing the classifiers they each produce (see Figure 3.9). We observe that the AdLaLa classifiers are far better resolved: they have lower loss and higher test accuracy, and are, moreoever, smoother.

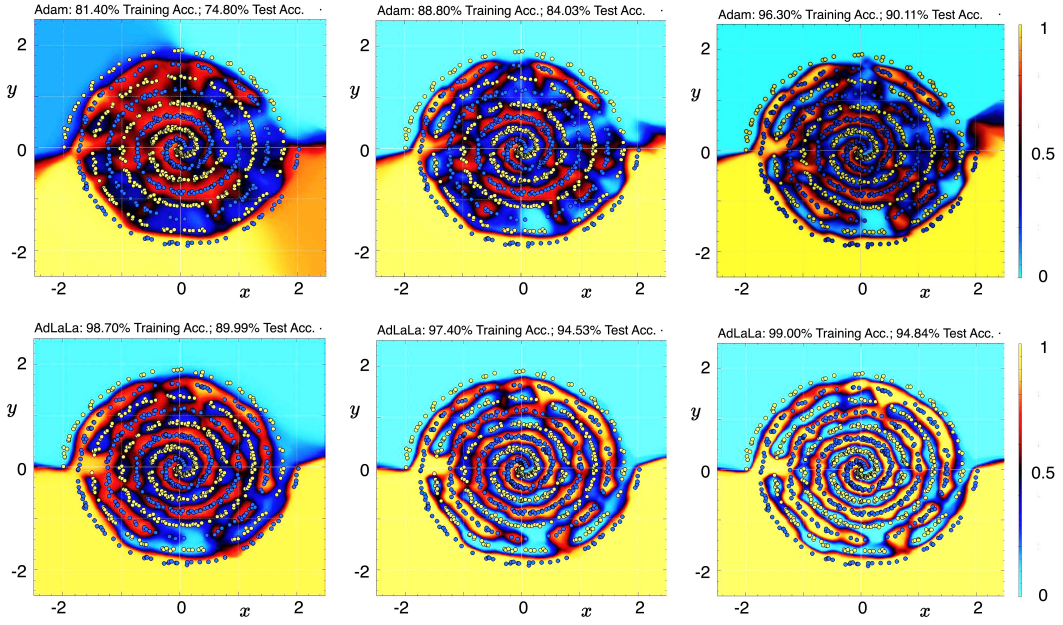

Figure 3.9: Comparison of classifiers for a 500-node SHLP on 4-turn spiral data (with $a = 2, b = 4, c = 0.02, p = 1$ in Eq. (3.32)) generated by Adam (top row) vs AdLaLa (bottom row). For Adam the stepsize used was $h = 0.005$. Adam was initialized with Gaussian weights with standard deviation 0.5. For AdLaLa the parameters were $\epsilon = 0.1$, $\tau_1 = 0.0001$, $\sigma_A = 0.01$, $\gamma_2 = 0.03$, $\tau_2 = 0.00001, h = 0.1$. Weights were initialized as Gaussian with standard deviation 0.01. For both methods we used 2% subsampling per step. From left to right in each row: 20K steps (400 epochs); 40K steps (800 epochs); 60K steps (1200 epochs). For visualization the classifier was averaged over the last 10 steps of training.

Evolution of the Weights and Classifier Boundaries During Training. We studied the development of the classification boundary between the two spiral classes as training progresses. We observe that no matter which optimizer has been used, all of the classifiers tend to distinguish the outer part of the two spirals first, before slowly filling in the classification plane inwards. Early on in the training, there are weights which are assigned a specific role in

fixing the shape of the classification boundary in the outer part of the spirals. These weights typically keep the same role throughout training.

We also isolated the effect of single data points on the training procedure. We distinguish data points from the center of the spirals and data points in the outer part of the spirals. We observed that for SGD data points from the outer part caused a larger change in the weight/bias values than the inner data points (at least initially, it typically changes after 2000 steps or so). For AdLaLa, however, the inner and outer data points affected the weights more or less equally from the outset.

Thermodynamic Parameterization Methods can have High Accuracy and Rapid Convergence. We provide evidence that our methods LOL and AdLaLa are able to converge more rapidly to a low test-loss parameterization than standard optimizers such as SGD, SGLD or Adam, for the spirals and trigonometric datasets. Our methods also perform competitively on the MNIST dataset compared to standard optimizers, but do not significantly outperform other methods for this problem.

In the following experiment we show the superiority of our AdLaLa method on the spirals dataset by fixing the parameters of AdLaLa, but varying the parameters of the other methods (at this point we only varied the stepsize for Adam, not its default parameters, i.e. we did not change the decay rates for the moving averages of the first and second moments). We show that AdLaLa consistently outperforms the other methods in terms of convergence rate. The experiments were performed using a neural network with a single hidden layer consisting of 100 nodes, 1000 test data, 1000 training data and 2% subsampling. We present comparisons for the spirals 4-turn dataset (Figure 3.10). We ran similar comparisons for easier 3-turn spiral data and observed similar trends. The amount of subsampling did not seem to affect the results much.

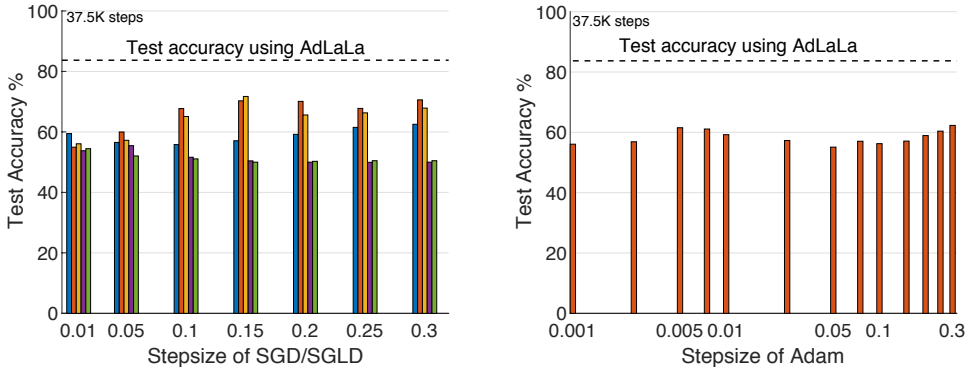

Figure 3.10: AdLaLa (black dotted horizontal line in both figures) consistently outperforms SGD, SGLD (left figure) and Adam (right figure) for the spiral 4-turn dataset. The different bars in the left figure indicate SGLD with different values of σ, namely $\sigma = 0$ (blue, this is standard SGD), $\sigma = 0.005$ (red), $\sigma = 0.01$ (yellow), $\sigma = 0.05$ (purple), $\sigma = 0.1$ (green). Whereas the set of parameter values for AdLaLa is fixed, the parameters of the other methods were varied to show the general superiority of AdLaLa. The results were averaged over multiple runs and the same initial conditions were used for all runs. The parameters used for AdLaLa were $h = 0.25, \tau_1 = \tau_2 = 10^{-4}, \gamma = 0.1, \sigma_A = 0.01, \epsilon = 0.05$.

We also show for the planar trigonometric example with $a = 6, b = 1, c = 0.02$ in Eq. (3.33) that our methods, LOL and AdLaLa, outperform Adam in terms of convergence rate (see Figure 3.11). Even at its (for this example) optimal time stepsize of $h = 0.01$ Adam is almost three times as slow as AdLaLa in obtaining 90% test accuracy.

In our tests on a harder example, which exhibits more crossings of the two data classes, namely $a = 10$ in Eq. (3.33), Adam was never able to reach the accuracy that LOL and AdLaLa obtain (see Figure 3.12). Its progress slows down rapidly and halts completely after

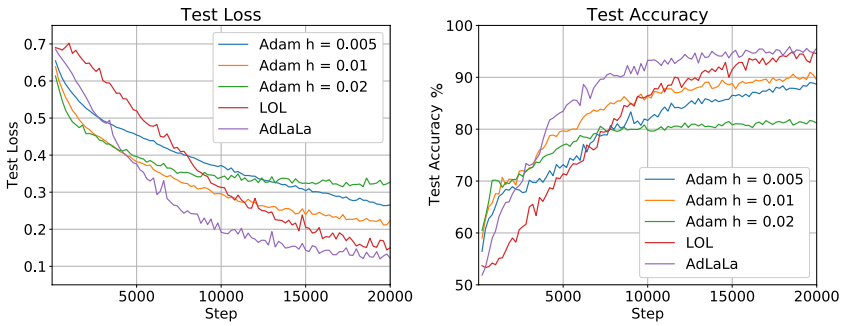

Figure 3.11: Test loss/accuracy obtained for planar trigonometric data (with a = 6 in Eq. (3.33)) using different optimizers and a 100 node SHLP, 1000 test data, 1000 training data and 5% subsampling. The parameters for LOL are set to $h = 0.1, \gamma_1 = 0.01, \tau_1 = 10^{-3}$. For AdLaLa we used parameters: $h = 0.2, \tau_1 = \tau_2 = 10^{-4}, \gamma = 10, \sigma_A = 0.001, \epsilon = 0.1$.

40,000 steps. After 100,000 steps its maximum test accuracy is still around 73%. SGLD is not able to compete at all.

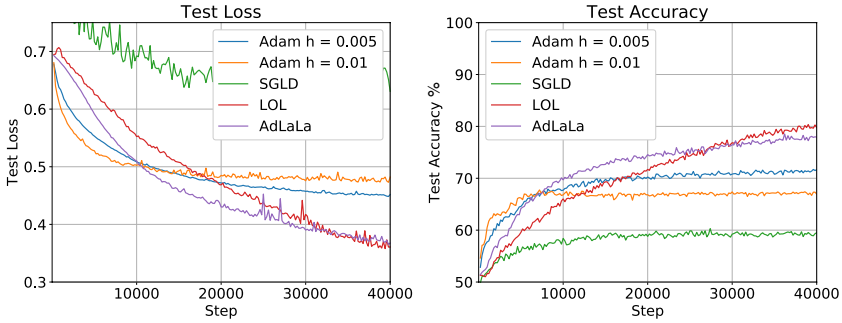

Figure 3.12: Test loss/accuracy obtained for planar trigonometric data (with a = 10 in Eq. (3.33)) with a 100 node SHLP, which was parameterized using different optimizers. The results were averaged over 20 runs. Hyperparameters settings: for LOL: $h = 0.1, \gamma_1 = 0.01, \tau_1 = 10^{-3}$; for AdLaLa: $h = 0.1, \tau_1 = \tau_2 = 10^{-4}, \gamma = 5, \sigma_A = 0.001, \epsilon = 0.1$; for SGLD: $h = 0.1, \sigma = 0.01$.

Thermodynamic Parameterization Methods can Reduce Overfitting. In this section we evaluate the robustness of our algorithms to overfitting. Overfitting is defined as the increase in test loss over time as the optimizer "overfits" on the provided training data and therefore has a reduced generalization performance. To emphasize the overfitting effect, we shall decrease the amount of our training data relative to our test data, namely we shall use 200 training data points vs 4000 test data points. We also increase the noise level in our 2-turn spiral dataset to $c = 0.1$ and use a 500 node SHLP.

In Figure 3.13 one observes that SGD clearly overfits in the sense that after a certain time its test loss monotonically increases with the number of steps. In contrast, LOL with a large enough value of γ_1 can be shown to not exhibit this behaviour. The same can be said for AdLaLa, but only after a careful selection of the method's parameter values. We note that for these parameter settings LOL and AdLaLa are slower in reaching the desired test and training accuracy, but this leads to more stability later on in the training process and limits the need for early stopping techniques. We do not claim that our methods universally counter overfitting, merely that they allow more flexibility which can lead to increased robustness to overfitting.

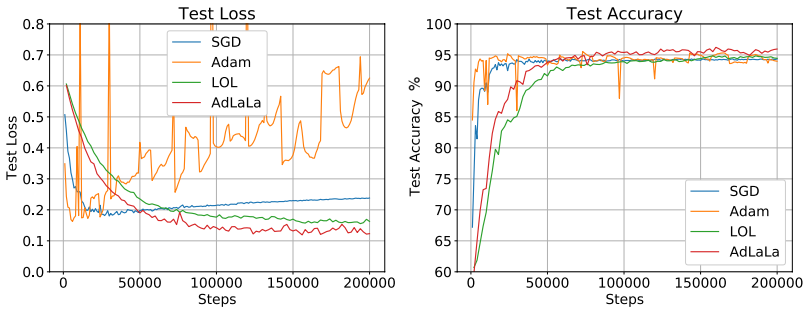

Figure 3.13: Obtained while training a 500-node SHLP on the 2-turn spiral (with $c = 0.1$ in Eq. (3.32)). We used $h_{\text{SGD}} = 0.1, h_{\text{Adam}} = 0.005$, for LOL: $h = 0.1, \gamma_1 = 1, \tau_1 = 10^{-6}$, for AdLaLa: $h = 0.1, \tau_1 = 10^{-4}, \tau_2 = 10^{-8}, \gamma = 1000, \sigma_A = 0.01, \epsilon = 0.1$.

Thermodynamic Parameterization Methods Enhance Robustness. We show that for the two turn spiral problem, Adam and SGD have a larger variance in their test accuracies over different runs than AdLaLa or LOL. We ran each of the optimizers 100 times and plotted the variance of the obtained test accuracies (see Figure 3.14).

The behaviour of Adam is highly dependent on the choice of initialization, while AdLaLa is less sensitive. We illustrate this by using both the standard PyTorch initialization [96, 219] for the weights (Adam is light blue and AdLaLa is green in Figure 3.14) and using a Gaussian initialization (Adam is dark blue and AdLaLa is purple in Figure 3.14). We also use Gaussian initialization for the other methods: SGD (red) and LOL (yellow). We observe that our methods – yellow (LOL) and green/purple (AdLaLa) in Figure 3.14 – have a much lower variance in their obtained test accuracies than Adam (with both initializations) and SGD.

Hybridization of Partitioned Methods: DHSGD with SGD. The DHSGD method (as described in section 3.3.2) only requires the tuning of a single parameter, namely the stepsize. It is therefore straightforward for practitioners to implement this method. We show that by running DHSGD for 2500 steps, we can then using the obtained neural network parameterization significantly improve the performance of SGD on the spiral problem with four turns, which SGD normally is only able to achieve a test accuracy of 0.7 on after 50k steps. In the following plot we have varied the stepsize for both methods and show the resulting test accuracies after 50k steps. The x-axis runs over the stepsize of the DHSGD method, whereas the different bars correspond to different stepsizes of the SGD method.

Studies of the Hessian of the loss function imply that the traditional notion of isolated basins is incorrect, as they show that as optimization progresses there is an abundance of flat directions, with the large majority of the eigenvalues being (close to) zero [244, 245]. We argue that our DHSGD method overcomes the saddle points which are located in the beginning of the optimization process, after which SGD is able to take over and descend into a good minimum.

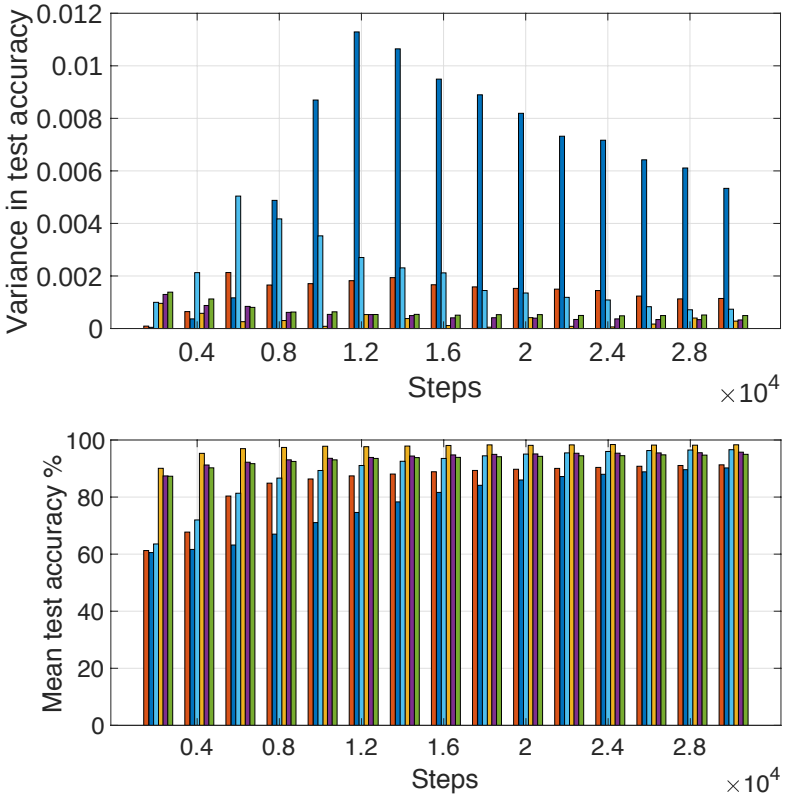

Figure 3.14: Variance (top) and mean (bottom) in test accuracies obtained over 100 runs on the two-turn spiral problem using SGD (red) with $h = 0.25$, Adam (dark blue) with $h = 0.005$ and $0.01 \cdot \mathcal{N}(0, 1)$ initialization for the weights, Adam (light blue) with $\mathcal{U}(-1/\sqrt{N_{in}}, 1/\sqrt{N_{in}})$ (standard PyTorch) initialization for the weights (where N_{in} is the number of inputs to the layer), LOL (yellow) with $h = 0.25, \gamma_1 = 0.01, \tau_1 = 10^{-3}$, and AdLaLa (purple) with $h = 0.25, \tau_1 = \tau_2 = 10^{-4}, \gamma = 0.5, \sigma_A = 0.01, \epsilon = 0.1$ with Gaussian initialization, AdLaLa (green) with standard PyTorch initialization. We used a 20 node SHLP, 500 training data and 2% subsampling.

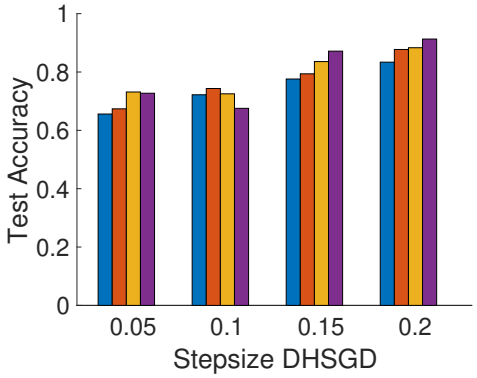

Figure 3.15: Standard SGD with appropriately tuned stepsize only reaches 70% test accuracy after 50k steps on the 4-turn spiral dataset. Here we run DHSGD for 2.5k steps and then SGD for 47.5k steps. We used 1000 training data, 2% subsampling, 1000 test data and initialized the momenta to be zero. The results were averaged over 4 runs. The different bars represent different stepsizes for SGD, namely: 0.05 (blue), 0.1 (red), 0.15 (yellow) and 0.2 (purple).

Role of Additive Noise in AdLaLa. As we can expect that gradient subsampling will introduce noise into the system, it is not immediately clear what the benefit of including additive noise is in the AdLaLa scheme (see Eq. (3.31)). However, we demonstrate in Figure 3.16 that choosing an appropriate noise strength $\sigma_A > 0$ can provide faster convergence to high quality minima.

We run experiments on classifying the four turn spiral problem using an SHLP with 100 hidden nodes. We draw 1000 data points as training data and use 2% subsampling for computing the gradient, with the test accuracy computed from 1000 independently drawn points. The parameters in the second layer are fixed for all experiments at $\gamma_2 = 0.03$ and $\tau_2 = 10^{-8}$, with $\epsilon = 0.1$. We look at the performance of the scheme for different values of τ_1 and σ_A by plotting the test accuracy (averaged over ten independent runs) after 50K steps with $h = 0.1$.

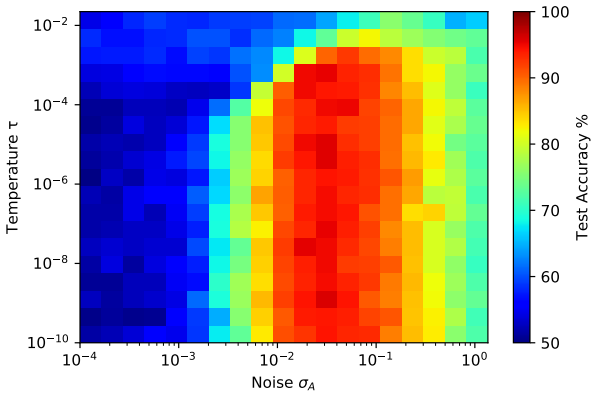

Figure 3.16: We run the AdLaLa scheme on an SHLP with 100 hidden nodes on the four turn spiral problem. Pixels indicate the average test accuracy with corresponding parameters, from ten independent runs, where $\gamma_2 = 0.03$, $\epsilon = 0.1$, $\tau_2 = 10^{-8}$, and $h = 0.1$.

The results in Figure 3.16 demonstrate that there is a broad range (at least an order of magnitude) where using additive noise significantly improves the performance of the classifier. We observe that reducing the strength of the additive noise too much (choosing $\sigma_A < 10^{-3}$ for example), or removing it entirely by setting $\sigma_A = 0$, gives poor results for the overall classification, with results no better than random noise. By contrast, we are able to recover near 100% accuracy for the same computational cost and with the same parameters by including additive noise of sufficient strength (for example choosing $\sigma_A = 0.04$).

The performance of the scheme seems relatively agnostic to the choice of target temperature parameter τ_1, provided it is sufficiently small. At too large a temperature the system is prevented from converging to an energy minima, leading to poor classification accuracy.

Role of Temperature in Partitioned Schemes. In Section 3.2.2, we showed in Figure 3.1 that the Langevin schemes could be more accurate when used with higher temperature. We close this series of numerical experiments with a demonstration using the 4-turn spiral data that the LOL method similarly is more accurate at modest temperatures (i.e. there is a band of temperature for which LOL performance improves), see Figure 3.17.

3.7 Conclusion

We have presented a new approach to parameterization of neural networks which can, in challenging data classification problems, accelerate convergence and provide improved test accuracy. The use of additive noise to supplement gradient noise was already proposed in previous works of other authors. We draw on this, by combining it with state-of-the-art principles for sampling algorithms coming from molecular dynamics and deploy partitioned algorithms that substantially improve on SGD and other optimization procedures. These new methods have other

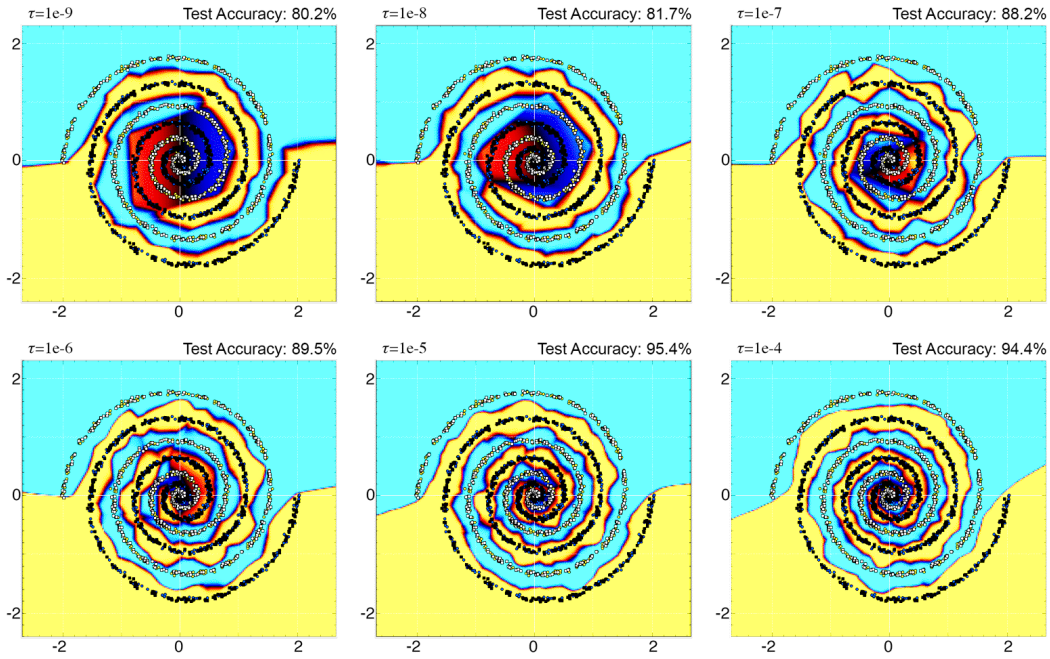

Figure 3.17: Comparison of classifiers for a 200-node SHLP on 4-turn spiral data generated by LOL with different temperature values. The friction was set at 1 in all experiments and 50,000 steps were performed with stepsize 0.8 (similar to large stepsizes used in SGD). Here performance increased with increasing τ until $\tau = 0.00001$ after which it began to decrease. (The method is unusable already for $\tau = 0.001$.)

advantages – for one thing they appear not to require additional regularization to obtain good performance (we did not use regularization in our experiments). Another advantage is that the stochastic methods, namely partitioned Langevin, LOL and AdLaLa, do not require complex initialization in the cases we studied. In fact, we initialized them frequently from zero initial weights and momenta and sometimes using built in training package procedures such as that in DLIB and PyTorch. This did not seem to significantly impact their performance.

In terms of future directions for research, we mention several important challenges. First, the experiments of this article have all focused on a limited collection of toy datasets, specifically classification problems for planar data. These present some difficulty for common training methods, so they are a good first step, but it is natural to look next at some state-of-the-art challenges such as arise in large scale image classification or natural language processing. Second, the power of these methods will not be fully recognized by the field until the results are demonstrated in deep networks, which are increasingly popular for machine learning applications due to better accuracy and generalization capabilities. In contrast to underparameterized networks, the loss landscape of overparameterized networks is not glassy [12, 80].[3] We still expect to see enhanced performance using our approach when extended to deep convolutional networks or transformers by using different optimizers for the fully connected layers versus the convolutional or self-attention layers, respectively.[4] However, further development of appropriate heuristics for the layer-wise partitioning and hyperparameter settings is required to limit the need for extensive tuning. Finally, we highlight the improved generalization properties of the models trained using our methods, as demonstrated in our experiments. Nowhere is the problem of poor generalization more acute than in the study of streaming data, where the continual perturbation of the data leads to aging of parameter sets which can necessitate frequent costly reparameterization. We therefore look to this topic for a rich source of problems to test out our methods in the future.

[3] Although the numerical results provided in this chapter include underparameterized networks, the rest of the thesis predominantly considers overparameterized networks.

[4] Some inspiration for this can be found in [299, 301] as discussed in Section 1.1.3.5.

Acknowledgements

The authors wish to thank John Chodera, Jason Frank, Anton Martinsson, Klaus-Robert Müller, Gabriel Stoltz, Amos Storkey, and Jonathan Weare for helpful discussions during the preparation of this manuscript. The work of Benedict Leimkuhler and Charles Matthews was supported by the Engineering and Physical Sciences Research Council (EPSRC) under EP/P006175/1. Benedict Leimkuhler is also a fellow of the Alan Turing Institute which is funded by grant EPSRC EP/N510129/1 and has benefited from this fellowship in the development of this work. Tiffany Vlaar is supported by The Maxwell Institute Graduate School in Analysis and its Applications, a Centre for Doctoral Training funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016508/01), the Scottish Funding Council, Heriot-Watt University and the University of Edinburgh. This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF).

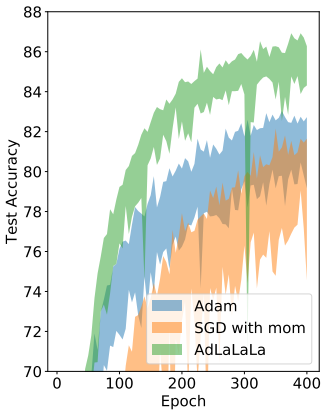

Figure 3.18: The data is the chess endgame (White King and Rook against Black King) dataset from the UCI Machine Learning Repository [60]. There are 6 predictive attributes (the row and column position for the three pieces) and 18 different classes representing in how many moves (0-16) black will lose or draw (1 class). We use a batchsize of 1024 and a 2-hidden layer MLP with 500 and 200 nodes, respectively, with ReLU activation. Hyperparameter settings: Adam: $h = 0.005$; SGD with momentum: $h = 0.05$, $mom = 0.95$; AdLaLaLa: $h = 0.2, \gamma = 0.1, \varepsilon = 0.01, \sigma_A = 0.001, T_1 =$ 1e-5, $T_2 =$ 1e-6.

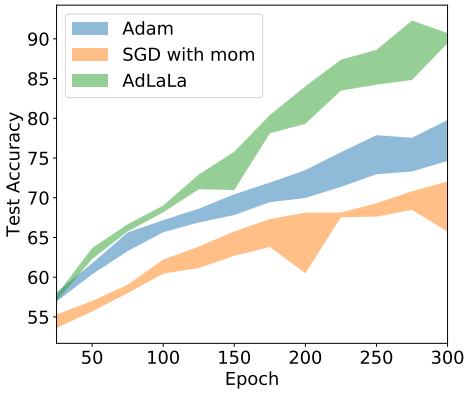

Figure 3.19: The data is the poker hand dataset from the UCI Machine Learning Repository [60]. It has 10 classes representing different poker hands (nothing in hand, one pair, ..., royal flush) and 10 predictive attributes (the suit and rank of each card in the five-card hand). Results are presented for the poker dataset using a batchsize of 500 and a SHLP with 200 nodes with ReLU activation. Hyperparameter settings: Adam: $h = 0.005$, SGD with momentum: $h = 0.005, mom = 0.9$, AdLaLa: $h = 0.05, \gamma = 5, \sigma =$ 1e-3, $\epsilon = 0.01$, $T_1 =$ 1e-4, $T_2 =$ 1e-6. The results were averaged over 5 runs.

3.8 Summary and Outlook

We have provided a second order Langevin dynamics framework for the training of neural networks. The value of the temperature τ can be seen as controlling the amount of exploration. Setting $\tau = 1$ provides a fully Bayesian approach, whereas setting $\tau \to 0$ focuses the posterior probability mass to the mode(s) of the distribution. We observed that setting τ to be very small, but non-zero provided the best performance. We exploit the particular architectural properties to further improve performance by developing a family of partitioned algorithms.

As examples of other datasets on which enhanced performance may be obtained using our partitioned thermodynamic methods we consider the poker hand and the chess endgame datasets, both from the UCI Machine Learning repository [60]. We compare the performance of

the AdLaLa method (Section 3.3.3) with standard optimizers Adam and SGD on these datasets in Figure 3.18 (chess) and Figure 3.19 (poker). We have performed a hyperparameter sweep for the stepsize of the Adam algorithm and the stepsize and momentum hyperparameters of the SGD with momentum algorithm to use the optimal hyperparameters.

We want to briefly highlight here relevant papers by other authors that came out after this work was published, which confirmed and expanded our findings. Wenzel et al., 2020 [287] showed using a similar approach, i.e. using Langevin dynamics with minibatch gradients and layerwise preconditioning, that the use of cold posteriors ($\tau \ll 1$) outperforms a fully Bayesian approach ($\tau = 1$) both in terms of test accuracy and predictive performance for bayesian neural networks (see also Section 1.3.4.3). Other works showed the advantages of being "Bayesian" in only some layers (Murfet et al., 2020 [202], Kristiadi et al., 2020 [145]). Finally, Sekkat and Stoltz, 2021 [250] showed that adaptive Langevin dynamics can be used to reduce (and for certain choices of the covariance matrix completely remove) the error induced by mini-batching.[5]

[5]This was already shown for a constant covariance matrix by Ding et al., 2014 [54].

Chapter 4

Better Training using Weight-Constrained Stochastic Dynamics

The results presented in this chapter appeared in *ICML* in 2021 [165] and are joint work with Benedict Leimkuhler, Timothée Pouchon, and Amos Storkey. A shorter workshop paper appeared in the *NeurIPS Optimization for Machine Learning (Opt-ML)* workshop and received a best student paper award.

Abstract

We employ constraints to control the parameter space of deep neural networks throughout training. The use of customized, appropriately designed constraints can reduce the vanishing/exploding gradients problem, improve smoothness of classification boundaries, control weight magnitudes and stabilize deep neural networks, and thus enhance the robustness of training algorithms and the generalization capabilities of neural networks. We provide a general approach to efficiently incorporate constraints into a stochastic gradient Langevin framework, allowing enhanced exploration of the loss landscape. We also present specific examples of constrained training methods motivated by orthogonality preservation for weight matrices and explicit weight normalizations. Discretization schemes are provided both for the overdamped formulation of Langevin dynamics and the underdamped form, in which momenta further improve sampling efficiency. These optimization schemes can be used directly, without needing to adapt neural network architecture design choices or to modify the objective with regularization terms, and see performance improvements in classification tasks.

4.1 Introduction

We study stochastic training methods based on Langevin dynamics combined with algebraic constraints. Our general framework allows for incorporating constraints into standard training schemes and sampling methods for neural networks. Constraints provide direct control of the parameter space of a model and hence afford a means to improve its generalization performance. As applications, we consider magnitude control and orthogonality of neural network weights.

Current approaches to enhance the generalization performance of overparameterized neural networks consist of both explicit and implicit regularization techniques [210]. Examples of the former are L_1 [288, 268] and L_2 [106] regularization, which modify the loss by adding a parameter norm penalty term. Batch normalization (BatchNorm) [121] is a technique that causes an implicit regularization effect. BatchNorm can be viewed as tantamount to a constraint imposed on the network's parameters during training. Although BatchNorm is widely used, explanations for the method's success remain elusive [247, 297]. The reliance on increasingly complex strategies does little to enhance the explainability of neural networks, so robust simplification

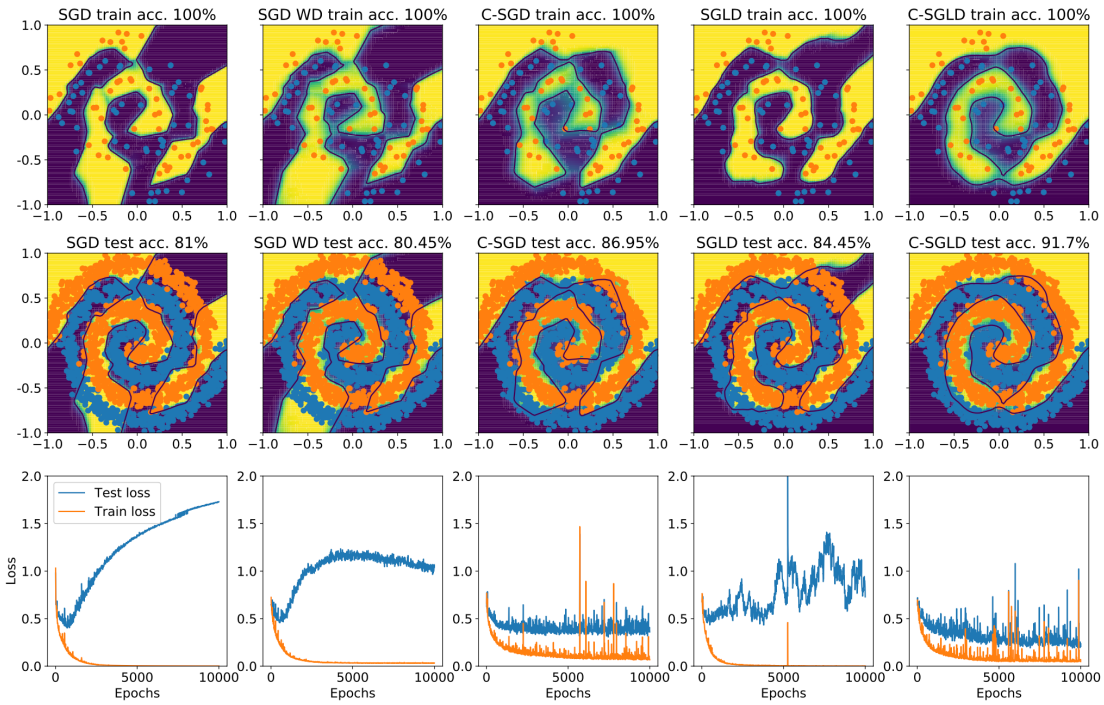

Figure 4.1: Classifiers obtained using different optimizers: SGD (1st column), SGD with weight decay (WD) (2nd column), constrained SGD (C-SGD) (3rd column), SGLD [285] (4th column), constrained SGLD (5th column) using a 500-node single hidden layer perceptron for a spiral binary classification problem. Top and middle row show training and test data points, respectively, and decision boundaries of the trained classifier. Bottom row shows loss curves. Hyperparameter settings: all: $h = 0.05$, 2% subsampling; SGD with WD = 1e-4; C-SGD: $r_0 = 1, r_1 = 5$ (see Eq. (4.2)); SGLD and C-SGLD: $\tau =$ 5e-5 (see Eq. (4.7)). We observe that although the use of WD can stabilize the test loss, it does not improve test accuracy (2nd col.). In contrast, our constrained approach (3rd col.) maintains a stable test loss throughout training and improved generalization performance. The use of additive noise (or low temperature) in combination with the constraints (C-SGLD, 5th col.) strongly outperforms standard SGD: 91.7% vs. 81% test accuracy, respectively, and obtains smoothened classification boundaries.

of all aspects of training is desirable. The constrained approach proposed in this work provides a conceptually straightforward and interpretable framework that offers direct control of parameter spaces, without requiring modifications to the neural network architecture or objective. The transparency of this approach allows for drawing a direct connection between the use of weight constraints and the generalization performance of the resulting neural network.

In neural network (NN) training one aims to minimize the loss $\mathcal{L}_X(\theta)$ for parameters $\theta \in \mathbb{R}^n$ and data X. Constraints can be seen as limiting cases of penalty-based regularization which replaces minimization of the loss $\mathcal{L}_X(\theta)$ by that of the augmented loss $\mathcal{L}^c_X(\theta) = \mathcal{L}_X(\theta) + \frac{1}{\varepsilon^2} g(\theta)^2$, where $g(\cdot)$ is a suitable smooth function of the parameters. In the limit $\varepsilon \to 0$, these penalty terms introduce an undesirable stiffness and consequent stability restriction in gradient-based training, which limits the choice of step size (see Figure 4.5 for an illustration). It is therefore natural to relate the above system to a constrained optimization task subject to $g(\theta) = 0$ (see Section 4.3).

A popular NN training scheme is stochastic gradient descent (SGD). SGD may be improved by incorporating momenta [263] and additive noise [285, 287], or more generally by embedding the loss gradient in a Langevin dynamics (LD) framework [44]. We will combine the resulting discretized stochastic differential equation (SDE) approach with constraints (Section 4.4).

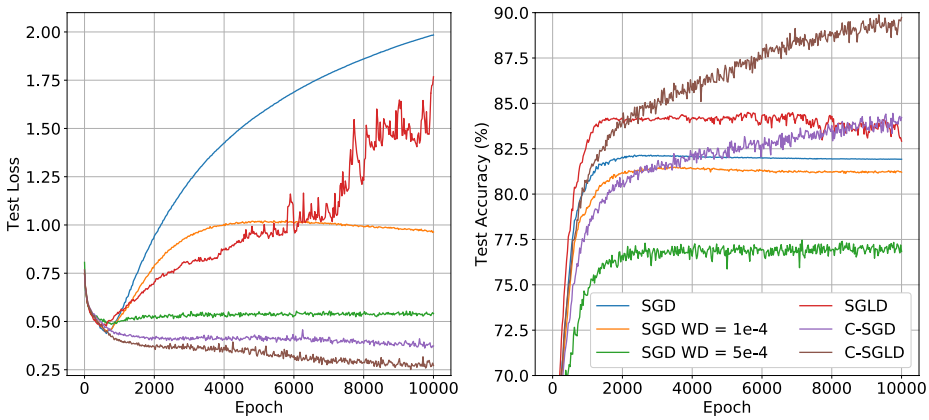

Figure 4.2: Same data and hyperparameter settings as for Figure 4.1, but these results are averaged over 100 runs. Constrained approaches, C-SGD and C-SGLD (with additive noise), clearly outperform standard SGD and SGD with WD in terms of test loss and test accuracy.

Table 4.1: Accompanies Figure 4.1 and 4.2, with same hyperparameter settings. We present estimates of the mean, standard deviation (std), and maximum (max) curvature of classifier boundaries obtained using different optimizers evaluated over 100 runs after training for a fixed number of 10,000 epochs. We computed our curvature estimates using the method described in Section 4.7.3, which we suggest is indicative of the curvature of the locally smoothed classification boundary and allows us to compare the relative curvature estimates of classifiers trained using different optimizers. The combined use of constraints and additive noise (C-SGLD) obtains much lower curvatures compared to SGD with weight decay (WD).

	Curvature Approximation		
Optimizer	Mean	Std	Max
SGD	519	$4.33 \cdot 10^4$	$3.26 \cdot 10^8$
SGD with WD	51.1	$3.80 \cdot 10^3$	$1.14 \cdot 10^7$
C-SGD	9.38	317	$5.58 \cdot 10^5$
SGLD	8.73	189	$6.27 \cdot 10^5$
C-SGLD	6.08	40.8	$1.43 \cdot 10^5$

The benefit of using constrained SDEs for NN training is illustrated in Figure 4.1, where the combination of using additive noise and magnitude constraints (as defined by Eq. (4.2)) leads to smoother classification boundaries and significantly enhanced generalization performance (compare the 5th column, the constrained SDE approach, with column 1, standard SGD). These observations are maintained over 100 runs (see Figure 4.2, Figure 4.3, and Table 4.1). We distinguish between two different types of smoothness of the resulting classifiers: first, the curvature of the classification boundary and second, the sharpness of the transition between prediction regions belonging to different classes. As shown in Table 4.1 and Figure 4.3 the use of magnitude constraints throughout training generates classifiers which exhibit both types of smoothness. The use of additive noise throughout training further reduces the curvature of the classification boundary. In contrast, the use of weight decay is not sufficient for SGD to obtain the same levels of smoothness. See Section 4.7.3 for further numerical details.

Apart from such magnitude constraints, the general framework provided in this work allows for straightforward incorporation of other constraints. Another specific example we consider is orthogonality of the weight matrix. We provide detailed algorithms for both of these purposes and for a general constraint in a Langevin dynamics setting (Section 4.4 and Section 4.7.2) and show improved generalization performance on classification tasks (Section 4.5).

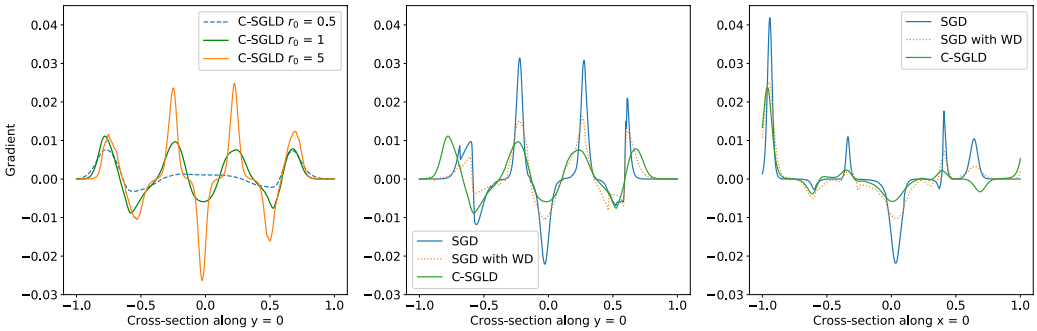

Figure 4.3: Same data and hyperparameter settings as in Figure 4.1 and 4.2. We show gradients of the prediction along horizontal ($y = 0$) and vertical cross-sections ($x = 0$). The results are averaged over 100 runs and evaluated at 10,000 epochs. Our constrained approach C-SGLD exhibits less sharp transitions between classes than SGD (middle/right plot). The size of the constraint directly controls this property (this is illustrated in the left plot for the input layer with constraint size r_0 in Eq. (4.2) for all input layer weights).

Concretely, our contributions are:

- We introduce the use of constrained stochastic differential equations for neural network training.
- We provide a general mathematical framework that allows for implementation of new constraints.
- We propose specific constraints, magnitude control and orthogonality of neural network weights, and provide algorithms to accompany these. The benefit of using these is illustrated for several networks and datasets and is shown to outperform soft constraints (such as weight decay or orthogonal regularization).
- We provide PyTorch code to support our algorithms, which can be found on `https://github.com/TiffanyVlaar/ConstrainedNNtraining`

4.2 Background and Related Work

Neural network loss landscapes are notoriously difficult to characterize rigorously due to their high-dimensionality and non-convexity. Although they appear to contain multiple, roughly equivalent local minima which exhibit nearly zero training loss [45, 133, 136], not all these minima obtain the same generalization performance [41, 291]. The training of deep neural networks is hypersensitive to e.g., the choice of initialization [263], optimizer [289], and hyperparameter settings [125], including learning rate scheduling [182, 258]. Without careful hyperparameter tuning, the loss landscape may not be explored sufficiently by the optimization scheme, thus resulting in a reduced generalization performance of the trained network [309, 136]. Sampling methods, which use small amounts of additive noise [163, 287], have been found to enhance exploration and speed the approach to 'good' minima, which enhance their generalization to nearby datasets. Hence, we incorporate the flexibility to use additive noise to enhance exploration in our optimization schemes by taking a constrained SDE approach to neural network training. We propose a general mathematical framework for this purpose and consider the ergodic properties of the idealized SDEs associated with gradient schemes, which may help these methods to ensure robust exploration of a useful range of parameters (Section 4.4). We further propose specific constraints (Section 4.3) and show that the use of these leads to enhanced performance compared to soft constraints, such as weight decay or orthogonal regularization (Section 4.5).

Magnitude Control of Neural Network Weights. In this work we consider a circle constraint, which limits the magnitude of the size of the weights (we typically leave the biases unconstrained). A corresponding soft constraint, which adds a penalty term to the loss, is weight decay or L_2 regularization [106]. We also propose a sphere constraint, which is analogous to max-norm [261, 262] as used in some regularization procedures. However, applying this constraint in combination with additive noise does yield a distinctive training method.

Orthogonality of the Weight Matrix. The concept of orthogonality has surfaced several times in the recent neural network literature. Orthogonal matrices have properties (norm preservation, unit singular values) which are thought to provide enhanced numerical stability [311, 237]. An orthogonal matrix $Q \in \mathbb{R}^{r\times s}$ (i.e., $Q^TQ = I_s$) is an isometry: $\|Qz\| = \|z\|$ $\forall z \in \mathbb{R}^s$. Orthogonal weight matrices were shown to mitigate the vanishing/exploding gradient problem in RNNs [218, 8, 281] and are developing a growing following in the CNN literature as well [237, 16, 113, 175]. Orthogonal initialization is linked to achieving dynamical isometry [248, 221, 222], which can accelerate training. Xiao et al. [293] were able to train 10,000 layer vanilla CNNs, without learning rate decay, BatchNorm or residual connections, by using initial orthogonal convolution kernels.

Methods for enforcing orthogonality during training include the use of 'soft' constraints which add a restraint term to the loss [32, 294, 16] and hard constraints based on optimization over Stiefel manifolds [113, 127]. The latter requires repeated singular value decomposition of high-dimensional matrices during training, which is costly. In this work we propose a straightforward algorithm to incorporate orthonormality constraints for rectangular matrices within our NN training framework, with manageable additional cost. We make no empirical claims over other manifold optimization methods, but rather provide a framework for network optimization that is theoretically sound, flexible enough to incorporate new constraints, and demonstrates good properties relative to standard SGD training or simple soft constraint approaches.

Constrained SDEs. In this work we focus on optimization schemes for neural networks using constrained Langevin dynamics in both its overdamped and underdamped (with momentum) form. A discussion of the properties of unconstrained Langevin dynamics in its overdamped and underdamped forms was studied in Pavliotis [220]. We consider the specific issues associated to the extension of the standard framework to constrained SDEs. The ergodic properties of constrained Langevin (in the absence of gradient noise) were previously studied in Lelièvre et al. [169] (overdamped) and Lelièvre et al. [166] (underdamped). Exponential convergence to equilibrium for constrained overdamped Langevin is a consequence of a Poincaré inequality. Poincaré inequalities on manifolds and their use in the analysis of diffusion processes are presented in Bakry et al. [14], Chapter 4. Finally, Langevin dynamics discretizations are studied in Faou and Lelièvre [67], Lelièvre et al. [169] (overdamped) and Lelièvre et al. [166], Leimkuhler and Matthews [158] (underdamped).

4.3 Neural Networks with Constraints

Imposing good priors on neural networks is known to improve performance, e.g. convolutional neural networks (CNNs) suit image datasets better than overparameterized fully connected NNs, despite being a subset of the latter [48]. Using constraints also arises naturally in the control of vanishing/exploding gradients [22, 224]. Constraints can be used to control the magnitudes of individual weights and/or to limit the growth of gradients in deep networks. We present various approaches in this section.

We consider a L-layer neural network, which has parameters $\theta \in \mathbb{R}^n$, with a weight matrix $W^\ell \in \mathbb{R}^{d^\ell \times d^{\ell-1}}$ and bias vector $b^\ell \in \mathbb{R}^{d^\ell}$ for each layer ℓ. To allow for inequality constraints, we define slack variables vector $\xi \in \mathbb{R}^{n^\xi}$ and consider the variable $q = (\theta, \xi) \in \mathbb{R}^d$, where $d = n+n^\xi$. The constraint manifold is

$$\Sigma = \{q \in \mathbb{R}^d \mid g(q) = 0\}, \quad g : \mathbb{R}^d \to \mathbb{R}^m. \tag{4.1}$$

We partition $\theta = (\theta^u, \theta^c)$ into unconstrained $\theta^u \in \mathbb{R}^{n^u}$ and constrained $\theta^c \in \mathbb{R}^{n^c}$ parameters.

We typically only constrain the neural network weights, not the biases.

Circle Constraints: In a *circle constraint*, we restrict each parameter in θ^c as $|\theta_i^c| \leq r_i$, where $r_i > 0$ is given. We thus introduce $m = n^c = n^\xi$ slack variables ξ_i and define

$$g_i(q) = |\theta_i^c|^2 + |\xi_i|^2 - r_i^2, \qquad 1 \leq i \leq m. \tag{4.2}$$

If $q \in \Sigma$, then the parameters in θ^c are bounded as desired.

Sphere Constraints: In a similar way, we could opt to restrict the sums of squares of weights associated to the input channels of any node. For layer ℓ, we denote the i-th row of the weight matrix W^ℓ as $\theta^{c,i}$, set $\theta^u = b^\ell$, introduce $m = d^\ell$ slack variables ξ_i, and define as *sphere constraint*

$$g_i(q) = \|\theta^{c,i}\|^2 + |\xi_i|^2 - r_i^2, \qquad 1 \leq i \leq m, \tag{4.3}$$

where $\|\cdot\|$ denotes the Euclidean norm. Sphere constraints are analogous to max-norm [261, 262], but have been unexplored in combination with additive noise. We leave this for future work.

Orthogonality Constraints: We set $\theta^u = b^\ell$, and define as *orthogonality constraint* for layer ℓ with n^ℓ parameters

$$g(q) = \begin{cases} \left(W^\ell\right)^T W^\ell - I_{n^{\ell-1}} & \text{if } n^{\ell-1} \leq n^\ell, \\ W^\ell \left(W^\ell\right)^T - I_{n^\ell} & \text{otherwise.} \end{cases} \tag{4.4}$$

As the matrix equality $g(q) = 0$ is symmetric, it corresponds to $m = s(s+1)/2$ constraints, where $s = \min\{n^{l-1}, n^l\}$.

4.4 Constrained SDEs and their Discretization

In this chapter we describe SDE-based methods for constrained neural network training. We first introduce standard (unconstrained) Langevin dynamics in Section 4.4.1. Then in Section 4.4.2 we discuss properties of constrained Langevin dynamics, such as ergodicity and exponential convergence to equilibrium, which ensures the effectiveness of our schemes as training methods. In Section 4.4.3 we discuss the discretization of constrained Langevin dynamics in both the overdamped and the underdamped case, where the use of momenta allow us to accelerate the training process. The choice of discretization scheme will strongly affect the efficiency and robustness of the resulting training method. Hence, to allow for ease and efficacy of implementation of our methods, we describe the most appropriate discretization schemes in detail in Section 4.7.2 for both the general setting and for the specific constraints we consider in this work, i.e. circle and orthogonality constraints.

4.4.1 Langevin Dynamics

Consider the unconstrained Langevin system of SDEs

$$\begin{aligned} \mathrm{d}\theta_t &= p_t \,\mathrm{d}t, \\ \mathrm{d}p_t &= -\nabla \mathcal{L}(\theta_t)\,\mathrm{d}t - \gamma p_t \,\mathrm{d}t + \sqrt{2\gamma\tau}\,\mathrm{d}\mathcal{W}_t, \end{aligned} \tag{4.5}$$

with momenta p, parameters θ, loss $\mathcal{L}(\theta)$, temperature hyperparameter $\tau \geq 0$, friction hyperparameter γ, and d-dimensional Wiener process $\mathcal{W}$ [157]. Under some mild assumptions, Langevin dynamics is provably ergodic, which means that its solutions sample the distribution

$$\rho \propto \ \exp[-(\mathcal{L}(\theta) + \|p\|^2/2)/\tau]. \tag{4.6}$$

The temperature hyperparameter τ, which controls the additive noise level, provides a direct connection between a pure optimization and sampling approach. The standard Bayes posterior

is recovered for $\tau = 1$, whereas setting $\tau = 0$ will provide maximum a posteriori (MAP) point estimates. The range of values in between corresponds to an artificially sharpened posterior, where as $\tau \to 0$, the posterior probability mass is confined closer and closer to the modes of the distribution.[1] Using low temperatures [163, 287], sampling methods have been found to enhance exploration and speed the approach to 'good' minima, which enhance their generalization to nearby datasets. In this work we therefore consider a constrained SDE approach to neural network training to allow for the incorporation of both constraints and additive noise.

4.4.2 Constrained Langevin: Ergodicity and Central Limit Theorem

The neural network loss function naturally extends to the variable $q = (\theta, \xi) \in \mathbb{R}^d$ taking the form $U(q) = \mathcal{L}(\theta)$ (note that in particular $\nabla_\xi U = 0$). The first continuous training method we consider is the constrained overdamped Langevin[2] system

$$\begin{aligned} \mathrm{d}q_t &= -\nabla U(q_t)\,\mathrm{d}t + \sqrt{2\tau}\,\mathrm{d}\mathcal{W}_t - \nabla_q g(q_t)\,\mathrm{d}\lambda_t, \\ 0 &= g(q_t), \end{aligned} \tag{4.7}$$

where $\mathcal{W}$ is a d-dimensional Wiener process, $\tau \geq 0$ is the temperature hyperparameter, and λ_t is an $\mathbb{R}^m$-valued vector of Lagrange multipliers. Provided the initial configuration q_0 satisfies the constraint, any trajectory q_t of (4.7) remains on the constraint manifold Σ defined in Eq. (4.1). For reciprocal temperature $\beta^{-1} = \tau > 0$, (4.7) is equivalent to an underlying ergodic (unconstrained) SDE (see Section 4.7.1.1) with unique invariant measure

$$\mathrm{d}\nu_\Sigma = Z^{-1} e^{-\beta U(q)}\,\mathrm{d}\sigma_\Sigma,\ Z = \int_\Sigma e^{-\beta U(q)}\,\mathrm{d}\sigma_\Sigma, \tag{4.8}$$

where σ_Σ is the surface measure on Σ.

Ergodicity ensures that averages of observables with respect to ν_Σ can be approximated by time averages of trajectories of (4.7). To ensure the practical use of (4.7) as a training method, we need the convergence to occur in a reasonable time. Thanks to the reversibility of the underlying SDE (see Section 4.7.1.1), exponential convergence to equilibrium occurs as a consequence of a Poincaré inequality for ν_Σ (see Section 4.7.1.2, 4.7.1.3 and Bakry et al. [14]). We provide a summary of the results here and refer to Section 4.7.1 for more details.

A Poincaré inequality holds under a curvature-dimension assumption: there exists $\rho > 0$ such that

$$CD(\rho, \infty): \qquad \mathrm{Ric}_{\mathfrak{g}} + \beta \nabla^2_{\mathfrak{g}} U \geq \rho \mathfrak{g}, \tag{4.9}$$

in the sense of symmetric matrices. The terms in (4.9) rely on the structure of Σ as a Riemannian manifold: $\mathfrak{g}$ is the Riemannian metric, $\mathrm{Ric}_{\mathfrak{g}}$ is the Ricci curvature tensor and $\nabla^2_{\mathfrak{g}} U$ is the Hessian of U on the manifold. Under (4.9) we have the following result (Section 4.7.1.2).

Theorem 4.4.1. Assume that there exists $\rho > 0$ and $N > n$ such that $CD(\rho, N)$ holds. Then ν_Σ satisfies a Poincaré inequality: there exists a constant $L > 0$ such that

$$\int_\Sigma \left|\phi(q) - \langle\phi\rangle_{\nu_\Sigma}\right|^2 \mathrm{d}\nu_\Sigma(q) \leq \frac{1}{2L} \int_\Sigma \left|\Pi(q)\nabla\phi(q)\right|^2 \mathrm{d}\nu_\Sigma, \quad \forall \phi \in H^1(\nu_\Sigma), \tag{4.10}$$

where $\Pi(q)$ is the projection described in (4.23) and $H^1(\nu_\Sigma)$ is the space of functions with square ν_Σ-integrable gradients Eq. (4.22).

Consequences of Theorem 4.4.1 are exponential convergence and a central limit theorem (CLT) for the convergence (see Section 4.7.1.3).

[1] Techniques such as annealing or simulated tempering vary τ throughout training to enhance the optimization process [141, 82].

[2] Unconstrained stochastic gradient overdamped Langevin dynamics is analogous to the algorithm known as SGLD [285] in the machine learning literature. In SGLD one adds an additional additive noise term (typically with constant variance) to the dynamics. For a decaying sequence of stepsizes $h_n \to 0$ one expects to eventually sample from a known stationary distribution.

Corollary 4.4.2. If (4.9) holds then

$$\int_\Sigma \left|\mathbb{E}(\phi(q_t) \mid q_0) - \langle\phi\rangle_{\nu_\Sigma}\right|^2 \mathrm{d}\nu_\Sigma(q_0) \leq C(\phi)e^{-2L/\beta t}, \ \forall \phi \in H^1(\nu_\Sigma), \tag{4.11}$$

where $C(\phi)$ depends only on ϕ. Furthermore we have the following convergence in law:

$$\sqrt{T}\big(\langle\phi\rangle_T - \langle\phi\rangle_{\nu_\Sigma}\big) \to \mathcal{N}(0, \sigma_\phi^2) \quad \text{as } T \to \infty,$$

where the asymptotic variance σ_ϕ^2 is bounded as $\sigma_\phi^2 \leq \frac{\beta}{L}\int_\Sigma \left|\phi - \langle\phi\rangle_{\nu_\Sigma}\right|^2 \mathrm{d}\nu_\Sigma$.

In $\mathbb{R}^n$ assumption (4.9) is equivalent to convexity of U, which is known to be too strong a requirement (a confining assumption is sufficient, see e.g. Lelièvre and Stoltz [168]). Although (4.9) can certainly be weakened, the above results ensure that provided the curvature of the manifold is well behaved, sampling on Σ has similar properties as on a flat space.

Introducing momenta p leads to constrained underdamped Langevin dynamics, the second order counterpart of Eq. (4.7)

$$\begin{aligned} \mathrm{d}q_t &= p_t\,\mathrm{d}t, \quad 0 = g(q_t), \\ \mathrm{d}p_t &= -\nabla U(q_t)\,\mathrm{d}t - \gamma p_t\,\mathrm{d}t + \sqrt{2\gamma\tau}\,\mathrm{d}\mathcal{W}_t - \nabla g(q_t)\mathrm{d}\lambda_t, \end{aligned} \tag{4.12}$$

where γ is the friction hyperparameter. The constraint induces a cotangency condition: $p \in T_q^*\Sigma$, where $T_q^*\Sigma = \{p \in \mathbb{R}^d \mid \nabla^T g(q)p = 0\}$ is the cotangent space of the manifold Σ. The corresponding phase space is the cotangent bundle $T^*\Sigma = \{(q,p) \mid q \in \Sigma, p \in T_q^*\Sigma\}$. Given an initial pair $(q,p) \in T^*\Sigma$, any trajectory (q_t, p_t) of (4.12) stays on $T^*\Sigma$ for all time.

(4.12) is equivalent to an underlying ergodic SDE, whose invariant measure is

$$\mathrm{d}\mu = e^{-\beta H(q,p)}\mathrm{d}\sigma_{T^*\Sigma}, \tag{4.13}$$

with Hamiltonian $H(q,p) = U(q) + \frac{1}{2}p^T p$ and Liouville measure of the cotangent bundle $\sigma_{T^*\Sigma}$ [166]. Based on the result for the unconstrained case, we expect exponential convergence to equilibrium also to hold here, but will leave this technical proof (e.g. based on hypocoercivity [275, 168]) for future work.

4.4.3 Discretization of Constrained Langevin Dynamics

The simplest iteration scheme $q_n \in \Sigma \mapsto q_{n+1} \in \Sigma$ for constrained overdamped Langevin dynamics (4.7) consists of an Euler–Maruyama step followed by projection onto the constraint manifold Σ. The best choice for the projection is constraint-specific.

For circle constraints we suggest orthogonal projection, which is both explicit and robust (we describe this in detail in Section 4.7.2.3). For orthogonality constraints, we derive an efficient quasi-Newton scheme to solve the non-linear system for the projection step (Section 4.7.2.5). We present the resulting training scheme in Algorithm 4.1, where we denote $Q = W^\ell$ if $n^\ell \leq n^{\ell-1}$ and $Q = (W^\ell)^T$ otherwise, and present one training iteration $Q_n \in \Sigma \mapsto Q_{n+1} \in \Sigma$. Further, we denote h as the stepsize, $G(Q) = \nabla_Q U(Q)$ and $\tilde{G}$ the gradient of the loss evaluated on a randomly subsampled partial dataset. R_n is an independent standard random normal matrix of the same size as Q. The initialization must be done with care: the constrained parameters and the potential slack variable must satisfy the constraint initially.

Algorithm 4.1 Orthogonality constraint overdamped Langevin

Every step:
$Q^{(0)} = Q_n - h\tilde{G}(Q_n) + \sqrt{2\tau h}R_n$,
for $k = 0$ **to** $K-1$ **do**
$Q^{(k+1)} = Q^{(k)} - \frac{1}{2}Q_n\big((Q^{(k)})^T Q^{(k)} - I_s\big)$,
$Q_{n+1} = Q^{(K)}$.

For underdamped Langevin dynamics a common way of building discretization schemes is via the use of splitting methods [158]. For the constrained underdamped Langevin system (4.12) an ABO splitting strategy under $0 = g(q_t), \;\; 0 = \nabla_q g(q_t) p_t$ gives:

$$\begin{aligned} &\text{A: } \mathrm{d}q_t = p_t \,\mathrm{d}t, \quad \mathrm{d}p_t = -\nabla_q g(q_t)\,\mathrm{d}\lambda_t, \\ &\text{B: } \mathrm{d}q_t = 0, \;\mathrm{d}p_t = -\nabla_q U(q_t)\,\mathrm{d}t - \nabla_q g(q_t)\,\mathrm{d}\mu_t, \\ &\text{O: } \mathrm{d}q_t = 0, \;\mathrm{d}p_t = -\gamma p_t\,\mathrm{d}t + \sqrt{2\gamma\tau}\,\mathrm{d}\mathcal{W}_t - \nabla g(q_t)\,\mathrm{d}\nu_t, \end{aligned} \tag{4.14}$$

In the specific case $\tau = 0$ and by re-scaling $\mu = e^{-\gamma h}/h$ and $\delta t = h^2$, an OBA sequence is equivalent to the standard PyTorch form of SGD with momentum μ and stepsize δt [219, 163]. As alternative one could use a symmetric splitting method, e.g. the BAOAB method [162].

In (4.14) the B and O components can be solved exactly (in law) while the A component can be approximated using a standard scheme for constrained ODEs (e.g. SHAKE or RATTLE [160][Chapter 7]). Importantly, the A component does not involve the evaluation of the gradient. For circle constraints the A step can be solved explicitly and the corresponding algorithm is provided in detail in Section 4.7.2.4. For orthogonality constraints all details are provided in Section 4.7.2.6, but we will provide the algorithm here. For $Q \in \Sigma$, the projection onto the cotangent space $T_Q^*\Sigma$ is defined as $\Pi_Q : \mathbb{R}^{r\times s} \to \mathbb{R}^{r\times s}$,

$$\bar{P} \mapsto \Pi_Q \bar{P} = \bar{P} - \frac{1}{2} Q(\bar{P}^T Q + Q^T \bar{P}). \tag{4.15}$$

We initialize the parameters and momenta (using projection (4.15)) to obey the constraint. Then the ABO steps $(Q_n, P_n) \in T^*\Sigma \mapsto (Q_{n+1}, P_{n+1}) \in T^*\Sigma$ are given by Algorithm 4.2, where $\tilde{G}(Q)$ is the gradient of the loss evaluated on a subset of the data. More details can be found in Section 4.7.2.

Algorithm 4.2 Orthogonality constraint underdamped Langevin

Every step:

$$\left.\begin{aligned} &Q^{(0)} = Q_n + hP_n, \\ &\textbf{for } k = 0 \textbf{ to } K-1 \textbf{ do} \\ &\quad Q^{(k+1)} = Q^{(k)} - \tfrac{1}{2} Q_n\big((Q^{(k)})^T Q^{(k)} - I_s\big), \\ &Q_{n+1} = Q^{(K)}, \;\bar{P}_{n+1} = P_n + \tfrac{1}{h}\big(Q_{n+1} - Q^{(0)}\big), \\ &P_{n+1} = \Pi_{Q_{n+1}} \bar{P}_{n+1} \end{aligned}\right\} \text{(A)}$$

$$\left.\begin{aligned} &\bar{P}_{n+1} = P_n - h\tilde{G}(Q_n), \\ &P_{n+1} = \Pi_{Q_n} \bar{P}_{n+1}, \end{aligned}\right\} \text{(B)}$$

$$\left.\begin{aligned} &P_{n+1} = e^{-\gamma h} P_n + \sqrt{\tau(1 - e^{-2\gamma h})} R_n, \\ &P_{n+1} = \Pi_{Q_n} \bar{P}_{n+1} \end{aligned}\right\} \text{(O)}$$

4.5 Numerical Experiments

The use of constraints can enhance generalization performance. We support this claim by comparing the performance of neural network architectures trained using the constrained approaches described in this work to nets trained using unconstrained SGD. We typically set $\tau = 0$ and use equivalent learning rates to present a fair comparison between constrained and unconstrained approaches. We denote our circle and orthogonal Constrained *overdamped* Langevin Algorithms as c-CoL*od* and o-CoL*od*, respectively. We compare *underdamped* variants (CoL*ud*) with SGD with momentum (SGD-m).

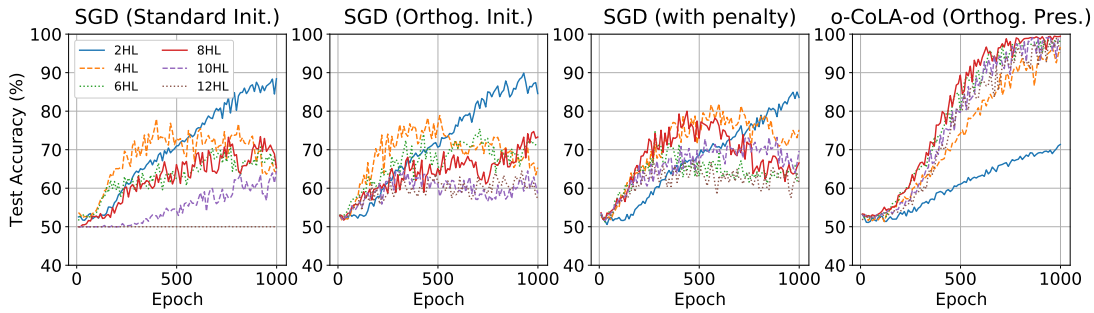

Figure 4.4: Test accuracy of MLPs with p-number of 100-node hidden layers (HL), ReLU activation. The MLPs are trained on a 4-turn spiral dataset (4.58) using SGD with standard initialization (1st column), SGD with orthogonal initialization (2nd column), SGD with orthogonal regularization ('soft constraint') by adding a penalty term with strength λ to the loss (3rd column), and o-CoLod with $\tau = 0$ (4th column). For the orthogonal regularization approach and o-CoLod we constrain weights in all layers, apart from input and output layers. We set stepsize $h = 0.1$ for all methods and use 5% subsampling. We found the optimal penalty strength $\lambda = 0.05$ for the orthogonal regularization method through line search. Results are averaged over 10 runs. We observe that our o-CoLod method significantly outperforms unconstrained SGD and SGD with a soft constraint for MLPs with more than 3 hidden layers.

4.5.1 Orthogonality Constraints

In Figure 4.4 we want to train a multi-layer perceptron (MLP) with p hidden layers on a tightly wound spiral binary classification problem (4.58) and compare the performance of SGD with our orthogonality-preserving overdamped Langevin method o-CoLod. For SGD we show results for i) standard PyTorch initialization, ii) orthogonal initialization, and iii) orthogonal regularization ('soft constraint'), where a penalty term is added to the loss to encourage orthogonality of the NN weight matrices. Our o-CoLod method clearly outperforms all of these variants in terms of test accuracy for MLPs with more than 3 hidden layers. In Section 4.7.3 (Figure 4.8) we show that the use of a small temperature perturbation can speed up training even further and slightly increase the test accuracy. The performance of the soft constraint approach can be somewhat improved by lowering the stepsize, yet cannot match the performance of o-CoLod (see Figure 4.5). This illustrates the undesirable stiffness introduced into the system by using penalty-based regularization. The use of o-CoLod also removes the need to tune an additional parameter (the penalty strength).

For a ResNet-34 architecture with BatchNorm and learning rate (LR) decay on CIFAR-10 [146] data our underdamped orthogonal constrained method, without weight decay (WD) significantly outperforms SGD-m without WD (Figure 4.6). The overdamped case is presented in the supplementary discussion, Figure 4.9. In future work we will explore the nuances of combining orthogonality constraints with BatchNorm, residual connections and LR decay.

4.5.2 Circle Constraints

We evaluate our circle constrained c-CoLud method on the Fashion-MNIST dataset [292]. We reduce the amount of training data to 10K samples and use the remaining 60K samples as test data. c-CoLud clearly outperforms SGD-m in terms of both test accuracy and test loss for a 1000-node single hidden layer perceptron (see Figure 4.7). The lower test loss of c-CoLud is maintained during training and the method shows no signs of overfitting, thus eliminating the need for early stopping. Even with weight decay, SGD-m is outperformed by its constrained counterpart (for more detailed hyperparameter studies see Section 4.7.3). We also show that a small transformer [273] with 2 encoder layers (each with 2-head self-attention and 200-node feed-forward network) trained using c-CoLud achieves a lower validation loss on NLP datasets than its unconstrained counterpart, SGD-m (Table 4.2).

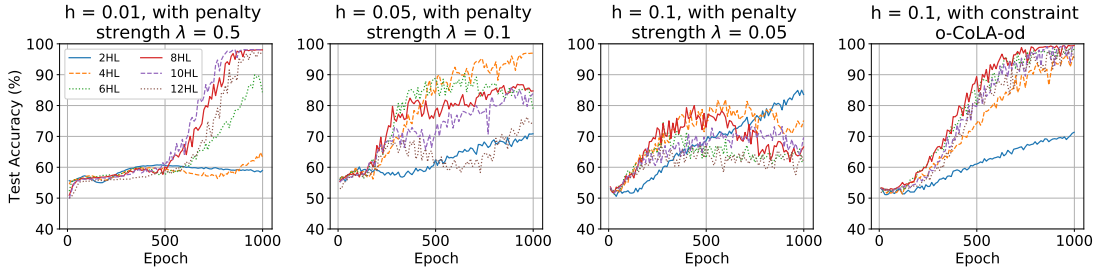

Figure 4.5: Same set-up as for Figure 4.4. MLPs with varying numbers of hidden layers (HL) were trained using o-CoLod with $h = 0.1$ (right-most) and using SGD with a penalty term added to the loss (results are presented in the 1st three columns with varying stepsizes h and penalty strengths λ). Results are averaged over 10 runs. We illustrate that the use of a penalty-based soft constraint introduces an undesirable stiffness into the system, needing the stepsize to be lowered to improve performance and to allow for the use of larger penalty strengths. The soft constraint approach is unable to reach the same performance as our o-CoLod method (right-most) and its performance is heavily dependent on the choice of penalty strength and step size.

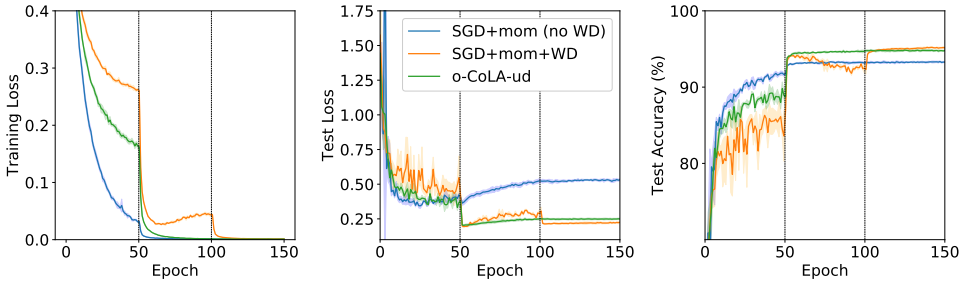

Figure 4.6: Train (left) & test (middle) loss and test accuracy (right) averaged over 5 runs of a ResNet-34 with BatchNorm trained using SGD-m vs. o-CoLud with $\tau = 0$ on CIFAR-10. For SGD we initially use $h = 0.1$ and decay by a factor 10 every 50 epochs (indicated by the vertical black dotted lines). We set momentum $= 0.9$ and present results with and without WD. o-CoLud (with $\gamma = 0.5$) did not use WD. Its learning rate was re-scaled to match the parameters of SGD-m and used the same LR schedule. The o-CoLud method without weight decay strongly outperforms SGD-m without weight decay.

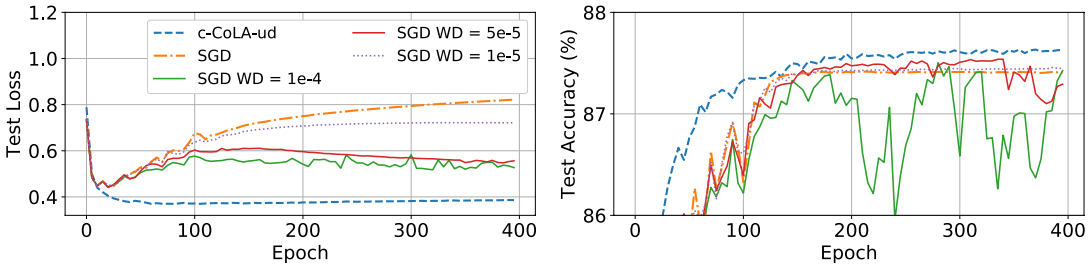

Figure 4.7: Test loss (left) and test accuracy (right) averaged over 5 runs of a 1000-node SHLP trained using SGD-m vs. c-CoLud on Fashion-MNIST (batchsize 128, number of training data samples reduced to 10K). After a line search we chose the best performing hyperparameter setting for SGD, namely $h = 0.1, mom = 0.8$, and varied the amount of weight decay (WD). Standard deviations are provided in the supplementary discussion. Hyperparameters c-CoLud: $h = 0.3, \gamma = 1, r_0 = 0.05, r_1 = 0.1, \tau = 0$. Due to the small training dataset size both methods quickly reached 100% training accuracy, but c-CoLud is superior in its test loss and test accuracy.

Table 4.2: Minimum validation loss on Penn Treebank data (batchsize 1024) [193] and Wikitext-2 (batchsize 128) [199] using a transformer trained using c-CoLud or SGD-m. Hyperparameters c-CoLud: $h = 0.4, r = 0.5, r_L = 0.1, r_N = 1, r_A = 1, \tau = 0, \gamma = 0.5$ (Treebank) and $\gamma = 1$ (Wikitext-2), where the subscripts L, N, A represent the radii belonging to the linear, norm and self-attention layers respectively. The transformer trained using c-CoLud obtains lower validation losses. Studies with weight decay are provided in the supplementary discussion.

Optimizer	Penn Treebank	Wikitext-2
c-CoLud	**4.81**	**5.09**
SGD $h = 0.1$		
$mom = 0.7$	4.87	5.13
$mom = 0.8$	4.83	5.13
$mom = 0.9$	4.84	5.13
SGD $h = 0.2$		
$mom = 0.7$	4.83	5.13
$mom = 0.8$	4.83	5.14

4.6 Conclusion

We provide a general framework that can be used to directly influence the parameter space of deep neural networks. The constrained SDE-based algorithms described in this work allow for the use of additive noise to enhance exploration but can also be used directly in combination with standard SGD approaches. We provide a mathematical framework to study these regularized training methods as discretizations of constrained Langevin dynamics and provide detailed discretization schemes (see Section 4.7.2). As specific examples of constraints we consider circle and orthogonality constraints, which obtain improved generalization performance on classification tasks compared to unconstrained SGD and soft constraint approaches. Further uses of our general framework are left for future work.

Acknowledgements

The authors wish to thank Gabriel Stoltz and Tony Lelièvre for helpful discussions on constrained SDEs. Benedict Leimkuhler is a fellow of the Alan Turing Institute which is supported by EPSRC grant EP/N510129/1. During the creation of this work Timothée Pouchon was supported by the Swiss National Science Foundation, project P2ELP2_188037. Tiffany Vlaar is supported by The Maxwell Institute Graduate School in Analysis and its Applications, a Centre for Doctoral Training funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016508/01), the Scottish Funding Council, Heriot-Watt University and the University of Edinburgh.

4.7 Supplementary Discussion

Section 4.7.1 provides the results necessary to establish exponential convergence to equilibrium of constrained overdamped Langevin dynamics (4.7). Section 4.7.2 provides discretization schemes and implementation details for our constrained training algorithms. The discretization schemes for a general constraint are described in Section 4.7.2.1 for overdamped Langevin dynamics and in Section 4.7.2.2 for underdamped Langevin dynamics. Our circle constrained algorithm is discussed in Section 4.7.2.3 (overdamped) and Section 4.7.2.4 (underdamped). Section 4.7.2.5 and 4.7.2.6 are reserved for our orthogonality constraint Langevin dynamics algorithm (overdamped and underdamped, respectively). Finally, Section 4.7.3 provides further numerical implementation details and results for our constrained methods.

4.7.1 Theory of Constrained Overdamped Langevin Dynamics

We present here the details of the theory summarized in Section 4.4. In particular, we provide the key results to establish the exponential convergence to equilibrium of constrained overdamped Langevin dynamics Eq. (4.7).

In the first part (Section 4.7.1.1), we derive the underlying SDE associated with Eq. (4.7), its generator and the invariant measure ν_Σ defined as

$$\mathrm{d}\nu_\Sigma = Z^{-1} e^{-\beta U(q)}\,\mathrm{d}\sigma_\Sigma, \quad Z = \int_\Sigma e^{-\beta U(q)}\,\mathrm{d}\sigma_\Sigma, \tag{4.16}$$

where σ_Σ is the surface measure on Σ. Ergodicity ensures that averages of observables with respect to ν_Σ can be approximated by time averages of trajectories of Eq. (4.7): for all test functions $\phi \in \mathcal{C}_c^\infty(\Sigma)$

$$\begin{aligned} &\lim_{T\to\infty} \langle\phi\rangle_T = \langle\phi\rangle_{\nu_\Sigma} \quad \text{for a.e. } q_0 \in \Sigma, \\ &\langle\phi\rangle_T := \frac{1}{T}\int_0^T \phi(q_t)\,\mathrm{d}t, \;\; \langle\phi\rangle_{\nu_\Sigma} := \int_\Sigma \phi(q)\,\mathrm{d}\nu_\Sigma(q). \end{aligned} \tag{4.17}$$

Next, in Section 4.7.1.2 we present the Poincaré inequality on a manifold, which holds under a curvature-dimension assumption: there exists $\rho > 0$ such that

$$CD(\rho,\infty): \qquad \mathrm{Ric}_{\mathfrak{g}} + \beta\nabla^2_{\mathfrak{g}} U \geq \rho\mathfrak{g}, \tag{4.18}$$

in the sense of symmetric matrices, where $\mathfrak{g}$ is the Riemannian metric, $\mathrm{Ric}_{\mathfrak{g}}$ is the Ricci curvature tensor and $\nabla^2_{\mathfrak{g}} U$ is the Hessian of U on the manifold. Under Eq. (4.18) the following result holds.

Theorem 4.7.1. Assume that there exists $\rho > 0$ and $N > n$ such that $CD(\rho, N)$ holds. Then ν_Σ satisfies a Poincaré inequality: there exists a constant $L > 0$ such that

$$\int_\Sigma \left|\phi(q) - \langle\phi\rangle_{\nu_\Sigma}\right|^2 \mathrm{d}\nu_\Sigma(q) \leq \frac{1}{2L}\int_\Sigma \left|\Pi(q)\nabla\phi(q)\right|^2 \mathrm{d}\nu_\Sigma(q), \quad \forall\phi \in H^1(\nu_\Sigma), \tag{4.19}$$

where $\Pi(q)$ is projection (4.23) and $H^1(\nu_\Sigma)$ is the space of functions with square ν_Σ-integrable gradients Eq. (4.22).

Consequences of Theorem 4.7.1 are the exponential convergence and a central limit theorem (CLT) for the convergence in Eq. (4.17).

Corollary 4.7.2. If Eq. (4.18) holds then

$$\int_\Sigma \left|\mathbb{E}(\phi(q_t) \mid q_0) - \langle\phi\rangle_{\nu_\Sigma}\right|^2 \mathrm{d}\nu_\Sigma(q_0) \leq C(\phi)e^{-2L/\beta t}, \quad \forall\phi \in H^1(\nu_\Sigma), \tag{4.20}$$

where $C(\phi)$ depends only on ϕ. Furthermore we have the following convergence in law:

$$\sqrt{T}\big(\langle\phi\rangle_T - \langle\phi\rangle_{\nu_\Sigma}\big) \to \mathcal{N}(0,\sigma_\phi^2) \quad \text{as } T\to\infty,$$

where the asymptotic variance σ_ϕ^2 is bounded as $\sigma_\phi^2 \leq \frac{\beta}{L}\int_\Sigma \left|\phi - \langle\phi\rangle_{\nu_\Sigma}\right|^2 \mathrm{d}\nu_\Sigma$.

Section 4.7.1.3 is dedicated to using the Poincaré inequality to proving this.

Notation

We collect here additional notation needed for this discussion.

Given a measure μ in a space $E \subset \mathbb{R}^d$, we associate the space of square integrable functions

$$L^2(\mu) = \left\{\phi : E \to \mathbb{R} \text{ measurable } : \int_E |\phi|^2 \,\mathrm{d}\mu < \infty\right\}.$$

Equipped with the inner product and associated norm

$$\langle\phi,\psi\rangle_\mu = \int_E \phi\psi \,\mathrm{d}\mu, \qquad \|\phi\|_{L^2(\mu)} = \sqrt{\langle\phi,\phi\rangle},$$

$L^2(\mu)$ is a Hilbert space. We further define the subspace $L^2(\mu)$ of functions with zero mean by

$$L_0^2(\mu) = \left\{\phi \in L^2(\mu) : \langle\phi\rangle_\mu = 0\right\}, \quad \langle\phi\rangle_\mu = \int_E \phi \mathrm{d}\mu, \tag{4.21}$$

as well as the space of functions with square integrable gradients

$$H^1(\mu) = \left\{\phi \in L^2(\mu) : \partial_i\phi \in L^2(\mu) \quad 1 \leq i \leq d\right\}. \tag{4.22}$$

For the constraint $g : \mathbb{R}^d \to \mathbb{R}^m$, we denote the Jacobian matrix as $G(q) = \nabla_q^T g(q)$ and denote its right pseudo-inverse by $G^+ = G^T(GG^T)^{-1}$ (GG^T is invertible if G has full row rank). We verify that the map

$$\Pi : \mathbb{R}^d \to \mathbb{R}^{d\times d}, \quad q \mapsto \Pi(q) = I_d - G^+(q)G(q), \tag{4.23}$$

defines for each q the orthogonal projection

$$\Pi_q = \Pi(q) : \mathbb{R}^d \to \mathbb{R}^d, \quad p \mapsto \Pi(q)p.$$

The matrix Π_q is symmetric and idempotent, i.e. $\Pi_q^T = \Pi_q$ and $\Pi_q^2 = \Pi_q$.

4.7.1.1 The Underlying SDE and the Invariant Measure

Although presented differently, the results of this section follow closely the treatment of this issue presented in [169].

We define the mean curvature of the manifold as the vector valued function

$$\mathcal{H} : \mathbb{R}^d \to \mathbb{R}^d, \quad q \mapsto \big(\mathcal{H}(q)\big)_i = \Pi_{jk}(q)\partial_j\Pi_{ik}(q), \quad 1 \leq i \leq d, \tag{4.24}$$

where $\Pi(q) : \mathbb{R}^d \to \mathbb{R}^d$ is the projection defined in Eq. (4.23). We then establish the following result (which is proved below):

Lemma 4.7.3. The constrained system Eq. (4.7) can be rewritten as the following SDE

$$\mathrm{d}q_t = -\Pi(q_t)\nabla U(q_t)\mathrm{d}t + \sqrt{2\beta^{-1}}\,\Pi(q_t)\mathrm{d}\mathcal{W}_t + \beta^{-1}\mathcal{H}(q_t)\,\mathrm{d}t. \tag{4.25}$$

The uniqueness of the invariant measure of Eq. (4.25) and the resulting ergodicity result Eq. (4.17) are proved in [169][Prop. 3.20] (the proof relies on the divergence theorem on manifolds).

The generator associated with Eq. (4.25) is given by

$$\mathcal{L} = -\Pi(q)\nabla U(q)\cdot\nabla + \beta^{-1}\mathcal{H}(q)\cdot\nabla + \beta^{-1}\Pi(q) : \nabla^2.$$

We verify that $\mathcal{L}$ can be written in the following symmetric form

$$\begin{aligned}\mathcal{L}\psi &= \beta^{-1}\operatorname{div}_\Sigma(\nabla_\Sigma\psi) - \nabla_\Sigma U(q)\cdot\nabla_\Sigma\psi \\ &= \beta^{-1}e^{\beta U(q)}\operatorname{div}_\Sigma\big(e^{-\beta U(q)}\nabla_\Sigma\psi\big),\end{aligned} \tag{4.26}$$

where we denote $\nabla_\Sigma\phi = \Pi\nabla\phi$ and $\operatorname{div}_\Sigma\psi = \nabla_\Sigma\cdot\psi = \sum_{i,j=1}^d \Pi_{ij}\partial_j\psi_i$. This expression directly implies that $\mathcal{L}$ is reversible with respect to ν_Σ:

$$\big\langle\mathcal{L}\phi,\psi\big\rangle_{\nu_\Sigma} = -\beta^{-1}\big\langle\nabla_\Sigma\phi,\nabla_\Sigma\psi\big\rangle_{\nu_\Sigma} = \big\langle\phi,\mathcal{L}\psi\big\rangle_{\nu_\Sigma}. \tag{4.27}$$

Thanks to this expression, we can prove that the measure ν_Σ is indeed invariant for Eq. (4.7). Let us introduce the forward Kolmogorov equation: given a test function $\phi\in\mathcal{C}_c^\infty(\Sigma)$

$$\partial_t u(t,q) = \mathcal{L}u(t,q)\ t\geq 0,\ q\in\Sigma, \qquad u(0,q)=\phi(q).$$

The solution to this equation is verified to be $u(t,q)=\mathbb{E}(\phi(q_t)\mid q_0=q)$ (see the Feynmann–Kac formula) and is usually denoted as $u(t,q)=e^{t\mathcal{L}}\phi(q)$. The measure ν_Σ is invariant if for any $t\geq 0$, $\int_\Sigma u(t,q)\,\mathrm{d}\nu_\Sigma(q) = \int_\Sigma u(0,q)\,\mathrm{d}\nu_\Sigma(q) = \langle\phi\rangle_{\nu_\Sigma}$. This is easily verified thanks to Eq. (4.27)

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}t}\int_\Sigma u(t,q)\,\mathrm{d}\nu_\Sigma(q) = \frac{\mathrm{d}}{\mathrm{d}t}\int_\Sigma e^{t\mathcal{L}}\phi(q)\,\mathrm{d}\nu_\Sigma(q)\\ &= \int_\Sigma \mathcal{L}e^{t\mathcal{L}}\phi(q)\,\mathrm{d}\nu_\Sigma(q) = \big\langle\mathcal{L}e^{t\mathcal{L}}\phi,\mathbf{1}\big\rangle_{\nu_\Sigma} = 0.\end{aligned}$$

Proof. Let us write λ_t as the Itô process

$$\mathrm{d}\lambda_t = \mu(q_t)\,\mathrm{d}t + \sigma(q_t)\,\mathrm{d}\mathcal{W}_t, \tag{4.28}$$

where $\mu:\mathbb{R}^d\to\mathbb{R}^m$, $\sigma:\mathbb{R}^d\to\mathbb{R}^{m\times d}$ and $\mathcal{W}_t$ is the same Wiener process as in Eq. (4.7). Using this expression in Eq. (4.7) brings

$$\mathrm{d}q_t = \big(-\nabla U(q_t) - G(q_t)^T\mu(q_t)\big)\,\mathrm{d}t + \big(\sqrt{2\beta^{-1}}I - G(q_t)^T\sigma(q_t)\big)\,\mathrm{d}\mathcal{W}_t,$$

where we recall the notation for the Jacobian $G=\nabla_q^T g$. Using Itô's formula we find

$$\begin{aligned}0 = \mathrm{d}g(q_t) &= G(q_t)\,\mathrm{d}q + b_t\,\mathrm{d}t\\ &= G(q_t)\big(-\nabla U(q_t)\,\mathrm{d}t + \sqrt{2\beta^{-1}}\,\mathrm{d}\mathcal{W}_t - G(q_t)^T\mathrm{d}\lambda_t\big) + b_t\,\mathrm{d}t,\end{aligned} \tag{4.29}$$

where b_t is the d-dimensional process defined as (omitting the dependence on q_t)

$$\begin{aligned}(b_t)_i &= \frac{1}{2}\big(\sqrt{2\beta^{-1}}I - G^T\sigma\big)\big(\sqrt{2\beta^{-1}}I - G^T\sigma\big)^T : \nabla^2 g_i\\ &= \beta^{-1}\Delta g_i - \frac{\sqrt{2\beta^{-1}}}{2}\big(G^T\sigma + \sigma^T G\big) : \nabla^2 g_i + \frac{1}{2}G^T\sigma\sigma^T G : \nabla^2 g_i.\end{aligned} \tag{4.30}$$

From Eq. (4.29) we obtain

$$\begin{aligned}\mathrm{d}\lambda_t = \big(G(q_t)G(q_t)^T\big)^{-1}G(q_t)\Big(&-\nabla U(q_t)\,\mathrm{d}t\\ &+\sqrt{2\beta^{-1}}\,\mathrm{d}\mathcal{W}_t\Big) + \big(G(q_t)G(q_t)^T\big)^{-1}b_t\,\mathrm{d}t.\end{aligned} \tag{4.31}$$

Identifying with Eq. (4.28) we find $\sigma(q) = \sqrt{2\beta^{-1}}(G^+(q))^T$, which used in Eq. (4.30) yields

$$(b_t)_i = \beta^{-1}\big(\Delta g_i - \big(G^T(G^+)^T + G^+G\big) : \nabla^2 g_i + G^T(G^+)^T G^+G : \nabla^2 g_i\big).$$

As G^+G is symmetric and $GG^+ = I_m$, we obtain

$$(b_t)_i = \beta^{-1}\big(\Delta g_i - G^+G : \nabla^2 g_i\big) = \beta^{-1}\Pi : \nabla^2 g_i. \tag{4.32}$$

Inserting Eq. (4.31) in Eq. (4.7) brings

$$\mathrm{d}q_t = -\Pi(q_t)\nabla U(q_t)\mathrm{d}t + \sqrt{2\beta^{-1}}\,\Pi(q_t)\mathrm{d}\mathcal{W}_t - G^+(q_t)b_t\,\mathrm{d}t. \tag{4.33}$$

To conclude the proof we require the following technical relations on the mean curvature vector (Eq. (4.34a) follows from a direct computation; the proof of Eq. (4.34b) is direct but involved and can be found in [169][Lemma 3.15]).

Lemma 4.7.4. The projection Π and the vector $\mathcal{H}$ defined in (4.23) and (4.24), respectively, satisfy the following equalities

$$\mathcal{H} = (I - \Pi)\nabla \cdot \Pi, \tag{4.34a}$$

$$\Pi : \nabla^2 g_i = -(G\mathcal{H})_i, \qquad 1 \le i \le d, \tag{4.34b}$$

Equality Eq. (4.34a) ensures that $\Pi\mathcal{H} = 0$. Combining Eq. (4.32) and Eq. (4.34b) we can write $b_t = -\beta^{-1}G\mathcal{H}$. Thanks to these relations and the definition of Π, we obtain

$$-G^+ b_t = \beta^{-1}G^+G\mathcal{H} = \beta^{-1}(I - \Pi)\mathcal{H} = \beta^{-1}\mathcal{H}.$$

This equality combined with Eq. (4.33) concludes the proof of Lemma 4.7.3.

□

4.7.1.2 Poincaré Inequality on a Manifold

Poincaré inequalities, also called spectral gap inequalities, form an important family of functional inequalities in the theory of Markov diffusion processes. They are the simplest inequalities that provide results on the convergence to equilibrium. Stronger results can be obtained with the family of log-Sobolev inequalities, which are at the center of the Bakry–Émery theory [13]. We follow here closely the book [14] on this subject (more specifically §1.16.2 and Section 4.2, 4.8, and C.6). For the necessary terminology of Riemannian manifolds we recommend the introductory textbook [154] (the literature on this topic is vast and contains many other works of high quality).

As presented in [14][Chapter 4], a Poincaré inequality can be obtained as a consequence of a curvature-dimension condition. For the sake of presentation, we introduce this result in the setting of a weighted Riemannian manifold. Let $(\mathcal{M}, \mathfrak{g})$ be an n-dimensional Riemannian manifold, where $\mathfrak{g}$ is the Riemannian metric. We consider the diffusion operator

$$\mathcal{L} = \Delta_{\mathfrak{g}} - \langle \nabla_{\mathfrak{g}} W, \nabla_{\mathfrak{g}} \cdot \rangle_{\mathfrak{g}},$$

where $\Delta_{\mathfrak{g}}$ denotes the Laplace–Beltrami operator on the manifold $\mathcal{M}$, $\nabla_{\mathfrak{g}}$ denotes the Levi–Civita connection (covariant derivative) and $\langle \cdot, \cdot \rangle_{\mathfrak{g}}$ denotes the Riemannian metric ($\langle X, Y \rangle_{\mathfrak{g}} = \mathfrak{g}(X, Y)$ for all vector fields X, Y). We verify that the associated invariant measure is $\mathrm{d}\mu = Z^{-1}e^{-W}\mathrm{d}\mu_{\mathfrak{g}}$, where $\mathrm{d}\mu_{\mathfrak{g}}$ is the Riemannian measure [14][§1.11.3]. For $N \in [n, \infty]$, we define the 2-tensor

$$\mathrm{Ric}_N(\mathcal{L}) = \mathrm{Ric}_{\mathfrak{g}} + \nabla^2_{\mathfrak{g}} W - \frac{1}{N-n}\mathrm{d}W \otimes \mathrm{d}W.$$

where $\mathrm{Ric}_{\mathfrak{g}}$ is the Ricci curvature 2-tensor and $\nabla^2_{\mathfrak{g}}$ denotes the Hessian operator on $\mathcal{M}$ (the case $N = n$ is considered only if W is constant). In this context, a curvature-dimension condition $CD(\rho, N)$ for $\rho \in \mathbb{R}$ and $N \ge n$ holds if and only if (see [14][C.6])

$$CD(\rho, N): \qquad \mathrm{Ric}_N(\mathcal{L}) \ge \rho\mathfrak{g}, \tag{4.35}$$

in the sense of symmetric $(0, 2)$-tensors (covariant 2-tensors). In the flat space $\mathcal{M} = \mathbb{R}^n$, the condition $CD(\rho, \infty)$ reads $\nabla^2 W \ge \rho I$, which is nothing but the convexity of the potential W. Under $CD(\rho, N)$, the measure μ is proved to satisfy a Poincaré inequality (in [14], combine Theorem 4.8.4 with the discussion in Section C.6).

Theorem 4.7.5. [14][Theorem 4.8.4] Under the curvature-dimension condition $CD(\rho, N)$ with $\rho > 0$ and $N \geq n$, $N > 1$, the measure μ satisfies the Poincaré inequality

$$\mathrm{Var}_\mu(\phi) = \|\phi - \langle\phi\rangle_\mu\|^2_{L^2(\mu)} \leq C_P \|\nabla_{\mathfrak{g}}\phi\|^2_{L^2(\mu)}, \tag{4.36}$$

with constant $C_P = \frac{N-1}{\rho N}$, $\forall \phi \in L^2(\mu) \cap H^1(\mu)$.

As the tensor $\mathrm{d}W \otimes \mathrm{d}W$ is positive semi-definite, we verify the monotonicity $\mathrm{Ric}_{N+M}(\mathcal{L}) \geq \mathrm{Ric}_N(\mathcal{L})$ for any $M \geq 0$. This implies in particular that $CD(\rho, N) \Rightarrow CD(\rho, \infty)$ for any $N \in [n, \infty]$. Hence, among all choices of $N \geq n$, $CD(\rho, \infty)$ is the weaker condition.

Let us now consider this result in the context of the constraint manifold Σ in Eq. (4.1). We consider the space $\mathbb{R}^d$ with its Riemannian manifold structure given by the Euclidean metric $\bar{\mathfrak{g}}(v, w) = v \cdot w$ for all $v, w \in \mathbb{R}^d$ (for all $q \in \mathbb{R}^d$, $p \in T_q\mathbb{R}^d$ is identified with $\mathbb{R}^d$ through a canonical isomorphism). Assuming that g is smooth and that $\nabla_q^T g$ has everywhere full row-rank, Σ is a smooth embedded submanifold of $\mathbb{R}^d$ of dimension $n = d - m$ (see e.g. [154]). Furthermore, Σ is equipped with the metric induced by $\bar{\mathfrak{g}}$: for a local parameterization of $\psi : V \subset \Sigma \to \mathbb{R}^d$, $\bar{\mathfrak{g}}$ is given locally on V by

$$\bar{\mathfrak{g}} = \sum_{i=1}^{d} \sum_{j,k=1}^{n} \frac{\partial \psi^i}{\partial x^j} \frac{\partial \psi^i}{\partial x^k} \mathrm{d}x^j \mathrm{d}x^k = \left(\nabla_x \psi \nabla_x^T \psi\right)_{jk} \mathrm{d}x^j \mathrm{d}x^k. \tag{4.37}$$

We now define the potential $W = \beta U|_\Sigma$, where $U|_\Sigma$ denotes the restriction of U to Σ. Assumption 4.18 corresponds then to condition $CD(\rho, \infty)$ above. Applying Theorem 4.7.5 we obtain Poincaré's inequality on the constraint manifold Σ. We note that for a function ϕ defined on $\mathbb{R}^d$, the covariant derivative in $\mathbb{R}^d$ of $\phi|_\Sigma$ on the manifold is the orthogonal projection of the directional derivative of ϕ (in the ambient manifold $\mathbb{R}^d$) onto the cotangent space: $\nabla_{\mathfrak{g}}(\phi|_\Sigma)(q) = \Pi(q)\nabla_q \phi(q)$. Furthermore, we note that the surface measure σ_Σ equals the Riemannian measure on the manifold (compare [169][Rem. 3.4] with [154][Prop. 2.41] and Eq. (4.37)). We thus obtain the result of Theorem 4.7.1 with constant $C_P = \frac{1}{\rho} = \frac{1}{2L}$.

4.7.1.3 Exponential Convergence to Equilibrium and Central Limit Theorem

Let us define the norm of a linear operator $\mathcal{A} : L^2_0(\nu_\Sigma) \to L^2_0(\nu_\Sigma)$ as

$$\|\mathcal{A}\|_{\mathcal{B}(L^2_0(\nu_\Sigma))} = \sup_{\phi \in L^2_0(\nu_\Sigma)} \frac{\|\mathcal{A}\phi\|_{L^2_0(\nu_\Sigma)}}{\|\phi\|_{L^2_0(\nu_\Sigma)}}.$$

Denote $\bar{\phi} = \phi - \langle\phi\rangle_{\nu_\Sigma} \in L^2_0(\nu_\Sigma)$. The Poincaré inequality Eq. (4.19), rewritten on the subspace $L^2_0(\nu_\Sigma)$, is as follows:

$$\|\bar{\phi}\|^2_{L^2_0(\nu_\Sigma)} \leq \frac{1}{2L} \|\nabla_\Sigma \bar{\phi}\|^2_{L^2_0(\nu_\Sigma)} \quad \forall \bar{\phi} \in L^2_0(\nu_\Sigma) \cap H^1(\nu_\Sigma). \tag{4.38}$$

Using the reversibility of the measure Eq. (4.27), we can prove the following result (the proof follows the same lines as [168][Prop. 2.3], see also [14][Theorem 4.2.5]).

Lemma 4.7.6. The measure ν_Σ satisfies the Poincaré inequality Eq. (4.38) if and only if

$$\|e^{t\mathcal{L}}\|_{\mathcal{B}(L^2_0(\nu_\Sigma))} \leq e^{-2\frac{L}{\beta}t}. \tag{4.39}$$

Exponential convergence to equilibrium is then directly obtained from Lemma 4.7.6

$$\begin{aligned} \|e^{t\mathcal{L}}\bar{\phi}\|_{L^2_0(\nu_\Sigma)} &\leq \|e^{t\mathcal{L}}\|_{\mathcal{B}(L^2_0(\nu_\Sigma))} \|\bar{\phi}\|_{L^2_0(\nu_\Sigma)} \\ &\leq e^{-2\frac{L}{\beta}t} \|\bar{\phi}\|_{L^2_0(\nu_\Sigma)}. \end{aligned} \tag{4.40}$$

This inequality implies Eq. (4.20) (note that $e^{t\mathcal{L}}\langle\phi\rangle_{\nu_\Sigma} = \langle\phi\rangle_{\nu_\Sigma}$) and thus proves the first assertion of Corollary 4.7.2.

A consequence of the exponential convergence to equilibrium Eq. (4.40) is the following central limit theorem for time averages $\langle\phi\rangle_T = \frac{1}{T}\int_0^T \phi(q_t)\,\mathrm{d}t$ (see also [140]).

Theorem 4.7.7. [25] If Eq. (4.40) holds, then the following convergence in law is satisfied

$$\sqrt{T}\big(\langle\phi\rangle_T - \langle\phi\rangle_{\nu_\Sigma}\big) \to \mathcal{N}(0,\sigma_\phi^2) \quad \text{as } T\to\infty,$$

where the asymptotic variance σ_ϕ^2 is given by the formula $\sigma_\phi^2 = 2\langle\bar\phi, -\mathcal{L}^{-1}\bar\phi\rangle$ with $\bar\phi = \phi - \langle\phi\rangle_{\nu_\Sigma}$.

To quantify the asymptotic variance, we use the following classical result.

Lemma 4.7.8. (e.g., [168][Prop. 2.1]) If Eq. (4.39) holds, then the generator $\mathcal{L}$ is invertible and the resolvent can be expressed as $-\mathcal{L}^{-1} = \int_0^\infty e^{t\mathcal{L}}\,\mathrm{d}t$ and satisfies the bound $\|\mathcal{L}^{-1}\|_{\mathcal{B}(L^2_0(\nu_\Sigma))} \le \frac{\beta}{2L}$.

Using Lemma 4.7.8 and Cauchy–Schwartz inequality, the asymptotic variance in Theorem 4.7.7 can thus be bounded as

$$\begin{aligned}\sigma_\phi^2 = 2\int_\Sigma \bar\phi(-\mathcal{L}^{-1}\bar\phi)\,\mathrm{d}\nu_\Sigma &\le 2\|\mathcal{L}^{-1}\|_{\mathcal{B}(L^2_0(\nu_\Sigma))}\|\bar\phi\|^2_{L^2_0(\nu_\Sigma)}\\ &\le \frac{\beta}{L}\|\bar\phi\|^2_{L^2_0(\nu_\Sigma)}.\end{aligned}$$

This estimate completes the proof of the second assertion of Corollary 4.7.2.

4.7.2 Discretization of Constrained Langevin Dynamics

We present here the details of the constrained training methods considered in this work. Both the overdamped Eq. (4.7) and underdamped Eq. (4.12) Langevin dynamics are discretized for the constraints presented in Section 4.3. We emphasize that the initialization of each given method must be done with care: the constrained parameters, the potential slack variable, as well as their momenta in the underdamped case, have to satisfy the constraint initially.

Recall the notation introduced in Section 4.3: $\theta\in\mathbb{R}^n$ is the vector of all the parameters of the model, we consider the variable $q=(\theta,\xi)\in\mathbb{R}^d$, $d = n+n^\xi$, where $\xi\in\mathbb{R}^s$ is a slack variable to enforce the potential inequality constraints. The loss is extended $q=(\theta,\xi)$ as $U(q) = L_X(\theta)$ (in particular $\nabla_\xi U = 0$) and constraints are given by a map $g:\mathbb{R}^d\to\mathbb{R}^m$. The parameters are partitioned as $\theta = (\theta^u,\theta^c)$, where $\theta^u\in\mathbb{R}^{n^u}$ are not involved in any constraint while $\theta^c\in\mathbb{R}^{n^c}$ are.

4.7.2.1 Discretization of Constrained Overdamped Langevin (General Constraint)

Following [169][Chapter 3] a simple discretization of the constrained overdamped Langevin dynamics Eq. (4.7) is given by the iteration $q_n\in\Sigma\mapsto q_{n+1}$ defined as

$$\begin{aligned}&\bar q_{n+1} = q_n - \nabla_q U(q_n)h + \sqrt{2\beta^{-1}h}\,R_n,\\ &q_{n+1} = \bar q_{n+1} - \nabla_q g(q_n)\lambda_n,\\ &\text{where } \lambda_n\in\mathbb{R}^m \text{ is such that } g(q_{n+1}) = 0,\end{aligned}\tag{4.41}$$

where $R_n\sim N(0,I)$ is a vector of iid standard normal random variable. The first step of Eq. (4.41), $\bar q_{n+1}$, is an Euler–Maruyama step for standard overdamped Langevin. As $\bar q_{n+1}$ in $\mathbb{R}^d$ is generally not on the constrained manifold Σ, the last term is present to project $\bar q_{n+1}$ back onto Σ, ensuring $g(q_{n+1}) = 0$. In particular, for the unconstrained parameter we have $\nabla^T_{\theta^u} g = 0_{m\times n^u}$ which implies that $\theta^u_{n+1} = \bar\theta^u_{n+1}$ is a standard EM step.

In general, projecting back onto the manifold Σ, i.e., finding λ_n, can be done using root-finding algorithms. Nevertheless, for certain constraints g the roots can be found explicitly. This is the case for the circle constraint Eq. (4.2) (see Section 4.7.2.3). A potential weakness of method Eq. (4.41) is that the projection process can be guaranteed only for small enough step size h (i.e. $\bar q_n$ must be close to Σ). Indeed, even for the circle constraint if h is too large it might not be possible to project $\bar q_{n+1}$ back onto the circle following the direction $\nabla_q g(q_n)$.

See [167] for some discussion of methods to allow computation to be performed in the large timestep regime.

An alternative method is given by the iteration $q_n \in \Sigma \mapsto q_{n+1} \in \Sigma$ defined as in [169][Chapter 3]

$$\begin{aligned} &\bar{q}_{n+1} = q_n - \nabla_q U(q_n)\mathrm{d}t + \sqrt{2\beta^{-1}h}\, R_n, \\ &q_{n+1} = \bar{q}_{n+1} - \nabla_q g(q_{n+1})\lambda_n, \\ &\text{where } \lambda_n \in \mathbb{R}^m \text{ is such that } g(q_{n+1}) = 0, \end{aligned} \tag{4.42}$$

where $R_n \sim N(0, I)$ is a vector of iid standard normal random variable. The projection used in method Eq. (4.42) is in general more robust. The circle constraint is a good illustration of this: while in Eq. (4.41) we project following an oblique direction, in Eq. (4.42) the projection is orthogonal and always exists (see Section 4.7.2.3).

4.7.2.2 Discretization of Constrained Underdamped Langevin (General Constraint)

We next consider the discretization of the constrained underdamped Langevin dynamics Eq. (4.12) where we denote by $p = (p^u, p^c, p^\xi) \in \mathbb{R}^{n^u+n^c+n^\xi}$ the momenta associated with the configuration $q = (\theta^u, \theta^c, \xi)$. Following [158], the system is split into A,B,O components Eq. (4.14), where B represents a projected impulse defined by the loss gradient (restricted to the cotangent space), O represents a projected stochastic impulse, and A represents evolution along geodesics (i.e., for circle constraints, these are rotations on the circles).

As in the overdamped case, the equality $\nabla_{\theta^u}^T g = 0_{m\times n^u}$ ensures that the unconstrained parameters and their momenta (θ^u, p^u) evolve following the A,B,O steps for unconstrained underdamped Langevin (see [162]). As the B and O components only involve a variation in the momentum p_t and because the constraint only involves q_t, they can be solved exactly for any constraint. The A component involves a variation of the configuration q_t and thus cannot be solved exactly (in law) for any constraint. However, as this part does not include any force evaluation (which would require back-propagation to compute the gradient), it can be approximated cheaply using a few steps of standard well-known schemes such as SHAKE or RATTLE (see Section 4.7.2.6 for orthogonal constraints). Furthermore, for simple constraints such as the circle constraint Eq. (4.2) the A component can be solved explicitly (see Section 4.7.2.4).

Let us present the details of the B and O steps. For convenience, let us introduce the following notation for the variables involved in the constraint $w = (\theta^c, \xi) \in \mathbb{R}^{n^c+n^\xi}$ and associated momentum $p^w = (p^c, p^\xi) \in \mathbb{R}^{n^c+n^\xi}$. The projection onto the cotangent space Eq. (4.23) is then as

$$\begin{aligned} &\Pi(q) = I_d - \begin{pmatrix} 0 & 0 \\ 0 & \Pi_w(q) \end{pmatrix}, \\ &\text{with } \Pi_w = \begin{pmatrix} g_{\theta^c}^T H^{-1} g_{\theta^c} & g_\xi^T H^{-1} g_{\theta^c} \\ g_{\theta^c}^T H^{-1} g_\xi & g_\xi^T H^{-1} g_\xi \end{pmatrix}, \end{aligned} \tag{4.43}$$

where we have denoted the partial Jacobians by $g_{\theta^c} = \nabla_{\theta^c}^T g \in \mathbb{R}^{m\times n^c}$, $g_\xi = \nabla_\xi^T g \in \mathbb{R}^{m\times n^\xi}$ and the matrix $H = g_{\theta^c} g_{\theta^c}^T + g_\xi g_\xi^T \in \mathbb{R}^{m\times m}$.

B component. Given $q_0, p_0 \in T^*\Sigma$ and a time $t > 0$

$$q_t = q_0, \qquad p_t = p_0 - t\nabla_q U(q_0) - \nabla_q g(q_0)(\mu_t - \mu_0),$$

where μ_t is such that $p_t \in T_{q_t}^*\Sigma$ (i.e., it satisfies the constraint $0 = \nabla_q g(q_t)p_t$). Note that as q_0, p_0 satisfy the constraints we have $\mu_0 = 0$. Projecting onto the cotangent space $T_{q_t}^*\Sigma = T_{q_0}^*\Sigma$ and using $\Pi(q_0)\nabla_q g(q_0) = 0$ and $p_0 = \Pi(q_0)p_0$, we obtain

$$\begin{aligned} p_t = \Pi(q_t)p_t &= \Pi(q_0)\big(p_0 - t\nabla_q U(q_0) - \nabla_q g(q_0)\mu_t\big) \\ &= p_0 - t\Pi(q_0)\nabla_q U(q_0). \end{aligned}$$

The B step is thus obtained for a chosen stepsize $h > 0$ as: given $q_n = (\theta_n^u, \theta_n^c, \xi_n) \in \Sigma$ and

$p_n = (p_n^u, p_n^c, p_n^\xi) \in T^*_{q_n}\Sigma$

$$
\text{(B, gen.)} \quad
\begin{aligned}
&\theta^u_{n+1} = \theta^u_n, \qquad \theta^c_{n+1} = \theta^c_n, \qquad \xi_{n+1} = \xi_n,\\
&p^u_{n+1} = p^u_n - h\nabla_{\theta^u} L_X(\theta_n),\\
&\bar{p}^c_{n+1} = p^c_n - h\nabla_{\theta^c} L_X(\theta_n), \qquad \bar{p}^\xi_{n+1} = p^\xi_n,\\
&\begin{pmatrix} p^c_{n+1} \\ p^\xi_{n+1} \end{pmatrix} = \Pi_w(w_n) \begin{pmatrix} \bar{p}^c_{n+1} \\ \bar{p}^\xi_{n+1} \end{pmatrix},\\
&\text{where } w_n = \begin{pmatrix} \theta^c_n \\ \xi_n \end{pmatrix}
\end{aligned}
\tag{4.44}
$$

O component. Similarly as for the B part, the O part can be solved exactly in law for any constraint. Given $q_0, p_0 \in T^*\Sigma$ and a time $t > 0$, we have

$$
\begin{aligned}
q_t &= q_0,\\
p_t &= p_0 - \gamma \int_0^t p_t \,\mathrm{d}t + \sqrt{2\gamma\tau} \int_0^t \mathrm{d}\mathcal{W}_t - \nabla_q g(q_0)\nu_t,
\end{aligned}
$$

where ν_t ensures that $p_t \in T^*_{q_t}\Sigma$. Projecting to the cotangent space $T^*_{q_t}\Sigma = T^*_{q_0}\Sigma$ as before, we obtain

$$
\begin{aligned}
p_t &= \Pi(q_t)p_t\\
&= p_0 - \gamma \int_0^t \Pi(q_0)p_t \,\mathrm{d}t + \sqrt{2\gamma\tau}\Pi(q_0) \int_0^t \mathrm{d}\mathcal{W}_t.
\end{aligned}
$$

We thus recognize that p_t is an Ornstein–Uhlenbeck process:

$$
p_t \overset{\text{law}}{=} \Pi(q_0)\big(e^{-\gamma t}p_0 + \sqrt{\tau(1 - e^{-2\gamma t})}R\big)
$$

with $R \sim N(0, I_d)$, where the equality holds in law.

The O step is thus obtained for a chosen stepsize $h > 0$ as: given $q_n = (\theta^u_n, \theta^c_n, \xi_n) \in \Sigma$ and $p_n = (p^u_n, p^c_n, p^\xi_n) \in T^*_{q_n}\Sigma$

$$
\text{(O, gen.)} \quad
\begin{aligned}
&\theta^u_{n+1} = \theta^u_n, \qquad \theta^c_{n+1} = \theta^c_n, \qquad \xi_{n+1} = \xi_n,\\
&p^u_{n+1} = e^{-\gamma h}p^u_n + \sqrt{\tau(1 - e^{-2\gamma h})}R^u,\\
&\bar{p}^c_{n+1} = e^{-\gamma h}p^c_n + \sqrt{\tau(1 - e^{-2\gamma h})}R^c,\\
&\bar{p}^\xi_{n+1} = e^{-\gamma h}p^c_n + \sqrt{\tau(1 - e^{-2\gamma h})}R^\xi,\\
&\begin{pmatrix} p^c_{n+1} \\ p^\xi_{n+1} \end{pmatrix} = \Pi_w(w_n) \begin{pmatrix} \bar{p}^c_{n+1} \\ \bar{p}^\xi_{n+1} \end{pmatrix}\\
&\text{where } w_n = \begin{pmatrix} \theta^c_n \\ \xi_n \end{pmatrix},
\end{aligned}
\tag{4.45}
$$

and R^u, R^c, and R^ξ are independent standard normal random variables.

4.7.2.3 Circle Constraint, Overdamped Langevin (c-CoLod)

We consider here the circle constraint Eq. (4.2), for which the partial Jacobians are computed as

$$
\begin{aligned}
\nabla_q^T g &= \big(\nabla^T_{\theta^u} g, \nabla^T_{\theta^c} g, \nabla^T_\xi g\big) \in \mathbb{R}^{m\times(n^u+n^c+m)},\\
\partial_{\theta^u_j} g_i &= 0, \quad \partial_{\theta^c_j} g_i = 2\theta^c_i \delta_{ij}, \quad \partial_{\xi_j} g_i = 2\xi_i\delta_{ij},
\end{aligned}
\tag{4.46}
$$

where δ_{ij} is the Kronecker delta.

For this constraint, the projection step in Eq. (4.41) can be computed explicitly. Indeed λ_n can be found by solving the m quadratic equations $0 = g_i(\bar{q}_{n+1} - \nabla_q g(q_n)\lambda_n)$ $1 \leq i \leq m$. The (potential) two roots of each equation corresponds to the (potential) two projections of $\bar{q}_{n+1}$ onto the circle following the direction $\nabla g_i(q_n) = 2(\theta^c_{n,i}, \xi_{n,i})$. When two roots are found, we may select the one closest to the point of origin $(\theta^c_{n,i}, \xi_{n,i})$. However, if the point to project $(\bar{\theta}^c_{n+1,i}, \bar{\xi}_{n+1,i})$ is too far away from the circle, this oblique projection may not be possible (i.e., the quadratic equation has no real root).

For the circle constraint, method Eq. (4.42) thus leads to a more robust projection process. Indeed, as $\nabla g_i(q_{n+1}) = 2(\theta^c_{n+1,i}, \xi_{n+1,i})$, the direction of the projection is now orthogonal to the circle. To find an expression for the orthogonal projection P of a point $(\bar{\theta}_1, \bar{\xi}_1)$ on the circle, it is easier to use a geometrical approach than to find the Lagrange multipliers:

$$(\theta_1, \xi_1) = P(\bar{\theta}_1, \bar{\xi}_1) = \big(r_i \cos(\alpha), r_i \sin(\alpha)\big),$$

where $\alpha = \arctan\left(\frac{\bar{\xi}_1}{\bar{\theta}_1}\right)$. We obtain the following discretization of the overdamped Langevin with circle constraints. We initialize the parameters of the neural network using standard PyTorch initialization [219, 96], i.e., $\mathcal{U}(-1/\sqrt{N_{in}}, 1/\sqrt{N_{in}})$, where N_{in} is the number of inputs to a layer. The auxiliary variables ξ_i corresponding to the constrained parameters θ^c_i are initialized to obey the constraint $(\theta^c_i)^2 + \xi_i^2 = r_i^2$. For a chosen stepsize $h > 0$ and given a configuration $q_n = (\theta^u_n, \theta^c_n, \xi_n) \in \Sigma$, one step of the method is defined by $q_{n+1} = (\theta^u_{n+1}, \theta^c_{n+1}, \xi_{n+1}) \in \Sigma$ as

$$\begin{aligned}
\theta^u_{n+1,i} &= \theta^u_{n,i} - h\partial_{\theta^u_i} L_X(\theta_n) + \sqrt{2\beta^{-1}h} R^u_i, \\
\bar{\theta}^c_{n+1,i} &= \theta^c_{n,i} - h\partial_{\theta^c_i} L_X(\theta_n) + \sqrt{2\beta^{-1}h} R^c_i, \\
\bar{\xi}_{n+1,i} &= \xi_{n,i} + \sqrt{2\beta^{-1}h} R^\xi_i, \\
\alpha_{n,i} &= \arctan\left(\frac{\bar{\xi}_{n+1,i}}{\bar{\theta}^c_{n+1,i}}\right), \\
\theta^c_{n+1,i} &= r_i \cos(\alpha_{n,i}), \\
\xi_{n+1,i} &= r_i \sin(\alpha_{n,i}),
\end{aligned} \tag{4.47}$$

where R^u_i, R^c_i, R^ξ_i are independent standard normal random variables.

4.7.2.4 Circle Constraint, Underdamped Langevin (c-CoLud)

We provide here the full discretization of the underdamped Langevin dynamics in the case of the circle constraint Eq. (4.2).

A component. For the circle constraint we can solve the A step explicitly. First recall that as $\nabla^T_{\theta^u} g = 0$, the unconstrained parameters θ^u are obtained with a standard A step of the unconstrained underdamped Langevin. Let us then focus on solving the constrained components: we denote $w = (\theta^c, \xi), p^w = (p^c, p^\xi)$. Then for $1 \leq i \leq m$ the A step in Eq. (4.14) corresponds to the constrained ODEs

$$\begin{aligned}
\dot{w}_i &= p^w_i \\
\dot{p}^w_i &= -2\lambda_i w_i \\
|\theta^c_i|^2 + |\xi_i|^2 &= r_i^2, \qquad \theta^c_i p^c_i + \xi_i p^\xi_i = 0.
\end{aligned} \tag{4.48}$$

As these constrained ODEs are uncoupled, let us drop the specification of the index i. By assumption, we are given initial conditions that satisfy the constraint $(w_0, p^w_0) \in T^*\Sigma$. Solving the second order ODE $\ddot{w} = -2\lambda w$, we find that any solution has the form $w_t = R^{2\lambda}_t w_0$, where

R_t^ω is a rotation matrix with angular speed ω given with its time derivative as

$$R_t^\omega = \begin{pmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{pmatrix},$$
$$\dot{R}_t^\omega = \omega \begin{pmatrix} -\sin(\omega t) & \cos(\omega t) \\ -\cos(\omega t) & -\sin(\omega t) \end{pmatrix}.$$

Computing the momentum $p_t^w = \dot{w}_t = \dot{R}_t^\omega w_0$, and using the properties of R_t^ω we verify that w_t, p_t^w satisfy the constraints in Eq. (4.48) ($\|.\|$ denotes the Euclidean norm in $\mathbb{R}^2$ and $\cdot$ the dot product):

$$\|w_t\|^2 = \|R_t^\omega w_0\|^2 = \|w_0\|^2 = r^2,$$
$$w_t \cdot p_t^w = w_0^T (R_t^\omega)^T \dot{R}_t^\omega w_0 = 0.$$

We still have to find the angular speed $\omega = \sqrt{2\lambda}$ such that the momentum p_t^w is consistent with its initial value p_0^w (we denote $w_0 = (\theta_0^c, \xi_0)$ and $p_0^w = (p_0^c, p_0^\xi)$):

$$p_0^w = \dot{R}_0^\omega w_0 \quad \Leftrightarrow \quad p_0^c = \omega \xi_0 \;\text{ and }\; p_0^\xi = -\omega \theta_0^c.$$

We thus find that

$$\xi_0 p_0^c - \theta_0^c p_0^\xi = \omega\big(|\xi_0|^2 + |\theta_0^c|^2\big) = \omega r^2$$
$$\Leftrightarrow \quad \omega = \frac{1}{r^2}\big(\xi_0 p_0^c - \theta_0^c p_0^\xi\big).$$

We have thus found an explicit expression for the solution of the A component for circle constraints Eq. (4.48).

To complete the B and O steps given in Eq. (4.44) and Eq. (4.45), we need an explicit expression for the projection Π_w in Eq. (4.43) (using Eq. (4.46), recall that $m = n^c = n^\xi$):

$$\Pi_w(w) = \begin{pmatrix} I_m - D^{11} & -D^{12} \\ -D^{12} & I_m - D_{22} \end{pmatrix},$$

where $D^{kl} \in \mathbb{R}^{m \times m}$ are the diagonal matrices defined as

$$D_{ii}^{11} = \frac{|\theta_i^c|^2}{|\theta_i^c|^2 + |\xi_i|^2}, \quad D_{ii}^{12} = \frac{\theta_i^c \xi_i}{|\theta_i^c|^2 + |\xi_i|^2},$$
$$D_{ii}^{22} = \frac{|\xi_i|^2}{|\theta_i^c|^2 + |\xi_i|^2}.$$

Assuming that $w = (\theta^c, \xi)$ satisfies the constraint, the projection of $(\bar{p}^c, \bar{p}^\xi)$ is thus computed as

$$\begin{pmatrix} p^c \\ p^\xi \end{pmatrix} = \Pi_w(w) \begin{pmatrix} \bar{p}^c \\ \bar{p}^\xi \end{pmatrix},$$
$$\text{where} \quad \begin{aligned} p_i^c &= \bar{p}_i^c - \frac{\theta_i^c}{r_i^2}\big(\theta_i^c \bar{p}_i^c + \xi_i \bar{p}_i^\xi\big) \quad 1 \le i \le m, \\ p_i^\xi &= \bar{p}_i^\xi - \frac{\xi_i}{r_i^2}\big(\theta_i^c \bar{p}_i^c + \xi_i \bar{p}_i^\xi\big) \quad 1 \le i \le m. \end{aligned}$$

Note that in the B step Eq. (4.44), the above expressions can be simplified by combining the simple definition of $(\bar{p}_n^c, \bar{p}_n^\xi)$ with the constraint

$$0 = \big(\nabla^T g(q) p\big)_i = 2\big(\theta_i^c p_i^c + \xi_i p_i^\xi\big).$$

We provide below the explicit updates for the A, B and O components for circle constraints. We initialize the parameters of the net using standard PyTorch initialization [219, 96]. The

auxiliary variables ξ corresponding to the constrained parameters θ^c are initialized to obey the constraint $(\theta^c)^2 + \xi^2 = r^2$, so that $q_0 = (\theta_0^u, \theta_0^c, \xi_0) \in \Sigma$. The momenta, p^u, p^c, and p^ξ, are generated in the same manner as for standard SGD with momentum in PyTorch, i.e., as equal to the initial gradients. Subsequently, the momenta belonging to the constrained variables p^c and to the auxiliary variables p^ξ are projected using Π_w, so that $p_0 = (p_0^u, p_0^c, p_0^\xi) \in T_{q_0}^*\Sigma$. For a stepsize $h > 0$ we obtain

$$\text{(A)}\quad \begin{cases} \theta^u_{n+1,i} = \theta^u_{n,i} + h p^u_{n,i}, \\ \omega_i = \dfrac{1}{r_i^2}\big(\xi_{n,i} p^c_{n,i} - \theta^c_{n,i} p^\xi_{n,i}\big), \\ \theta^c_{n+1,i} = \cos(\omega_i h)\theta^c_{n,i} + \sin(\omega_i h)\xi_{n,i}, \\ \xi_{n+1,i} = -\sin(\omega_i h)\theta^c_{n,i} + \cos(\omega_i h)\xi_{n,i}, \\ p^u_{n+1,i} = p^u_{n,i}, \\ p^c_{n+1,i} = \omega_i\big(-\sin(\omega_i h)\theta^c_{n,i} + \cos(\omega_i h)\xi_{n,i}\big), \\ p^\xi_{n+1,i} = -\omega_i\big(\cos(\omega_i h)\theta^c_{n,i} + \sin(\omega_i h)\xi_{n,i}\big), \end{cases}$$

$$\text{(B)}\quad \begin{cases} \theta^u_{n+1} = \theta^u_n, \qquad \theta^c_{n+1} = \theta^c_n, \qquad \xi_{n+1} = \xi_n, \\ p^u_{n+1} = p^u_n - h\nabla_{\theta^u} L_X(\theta_n), \\ \bar{p}^c_{n+1,i} = p^c_{n,i} - h\Big(1 - \dfrac{1}{r_i^2}|\theta^c_{n,i}|^2\Big)\partial_{\theta^c_i} L_X(\theta_n), \\ \bar{p}^\xi_{n+1,i} = p^\xi_{n,i} + h\dfrac{1}{r_i^2}\theta^c_{n,i}\xi_{n,i}\partial_{\theta^c_i} L_X(\theta_n), \end{cases}$$

$$\text{(O)}\quad \begin{cases} \theta^u_{n+1} = \theta^u_n, \qquad \theta^c_{n+1} = \theta^c_n, \qquad \xi_{n+1} = \xi_n, \\ p^u_{n+1} = e^{-\gamma h} p^u_n + \sqrt{\beta^{-1}(1 - e^{-2\gamma h})}R^u, \\ \bar{p}^c_{n+1} = e^{-\gamma h} p^c_n + \sqrt{\beta^{-1}(1 - e^{-2\gamma h})}R^c, \\ \bar{p}^\xi_{n+1} = e^{-\gamma h} p^c_n + \sqrt{\beta^{-1}(1 - e^{-2\gamma h})}R^\xi, \\ p^c_{n+1,i} = \Big(1 - \dfrac{|\theta^c_{n,i}|^2}{r_i^2}\Big)\bar{p}^c_{n+1,i} - \dfrac{1}{r_i^2}\theta^c_{n,i}\xi_{n,i}\bar{p}^\xi_{n+1,i}, \\ p^\xi_{n+1,i} = -\dfrac{\theta^c_{n,i}\xi_{n,i}\bar{p}^c_{n+1,i}}{r_i^2} + \Big(1 - \dfrac{|\xi_{n,i}|^2}{r_i^2}\Big)\bar{p}^\xi_{n+1,i}, \end{cases}$$

where R^u, R^c, and R^ξ are vectors of independent standard normal random variables.

4.7.2.5 Orthogonality Constraint, Overdamped Langevin (o-CoLod)

We present here a particular discretization of the constrained overdamped Langevin dynamics Eq. (4.7) for the orthogonality constraint Eq. (4.4).

For notational convenience, we present the updates for the weight matrix W^ℓ of a given layer ℓ. The updates for the biases are standard Euler–Maruyama steps such as given for θ^u in Eq. (4.47).

Referring to Eq. (4.4), we denote

$$\begin{array}{llll} Q = W^\ell, & r = n^\ell, & s = n^{\ell-1} & \text{if } n^{\ell-1} \le n^\ell, \\ Q = (W^\ell)^T, & r = n^{\ell-1}, & s = n^\ell & \text{otherwise.} \end{array} \tag{4.49}$$

so that $Q \in \mathbb{R}^{r\times s}$. With this notation, the constraint Eq. (4.4) is $g(Q) = 0$ where

$$g : \mathbb{R}^{r\times s} \to \mathbb{R}^{s\times s}, \qquad g(Q) = Q^T Q - I_s. \tag{4.50}$$

Recall that due to symmetry, the matrix equality $g(Q) = 0_s$ corresponds to $s(s+1)/2$ constraints. We compute the partial derivative

$$\begin{aligned} &\partial_{Q_{kl}} g_{ij}(Q) = \delta_{li} Q_{kj} + \delta_{lj} Q_{ki} \\ &1 \le i, j, k \le s,\ 1 \le l \le r. \end{aligned} \tag{4.51}$$

In particular, if Λ is an $s \times s$ symmetric matrix, we verify that

$$\sum_{i,j=1}^{s} \partial_{Q_{kl}} g_{ij}(Q) \Lambda_{ij} = 2\big(Q\Lambda\big)_{kl}.$$

We thus obtain the natural matrix form of the constrained dynamics Eq. (4.7): $Q_t : (0,\infty) \to \mathbb{R}^{r\times s}$ solves

$$\begin{aligned} &\mathrm{d}Q_t = -\nabla_Q U(Q_t)\,\mathrm{d}t + \sqrt{2\beta^{-1}}\,\mathrm{d}\mathcal{W}_t - Q_t\,\mathrm{d}\Lambda_t, \\ &g(Q_t) = 0, \end{aligned} \tag{4.52}$$

where $\big(\nabla_Q U\big)_{ij} = \partial_{Q_{ij}} U = \partial_{W^\ell_{ij}} L_X$ (or $\partial_{W^\ell_{ji}} L_X$) and $\mathcal{W}_t$ is a Wiener process in $\mathbb{R}^{r\times s}$. Furthermore the process Λ_t has values in the $s \times s$ symmetric matrices and is the Lagrange multiplier corresponding to the $s(s+1)/2$ constraints.

Applying discretization scheme Eq. (4.41) to Eq. (4.52), we obtain the iteration step $Q_n \in \Sigma \mapsto Q_{n+1} \in \Sigma$ given by

$$\begin{aligned} &\bar{Q}_{n+1} = Q_n - h\nabla_Q U(Q) + \sqrt{2\beta^{-1}h} R_n, \\ &Q_{n+1} = \bar{Q}_{n+1} - Q_n \Lambda_n, \end{aligned} \tag{4.53}$$

where Λ_n is a symmetric $s\times s$ matrix s.t. $g(Q_{n+1}) = 0$ and $R_n \in \mathbb{R}^{r\times s}$ is a matrix of independent standard normal random variables.

Note that the projection step in Eq. (4.53) requires to solve a non-linear system. Following a similar technique as described in [160][Chapter 8], we derive a quasi-Newton scheme for that task. Using the fact that Q_n satisfies the constraint we verify that

$$\bar{Q}_{n+1}^T Q_n = I_s - h\nabla_Q U(Q_n)^T Q_n + \sqrt{2\beta^{-1}h} R_n^T Q_n.$$

The constraint $g(Q_{n+1}) = 0$ thus reads

$$\begin{aligned} 0 &= \big(\bar{Q}_{n+1} - Q_n\Lambda_n\big)^T \big(\bar{Q}_{n+1} - Q_n\Lambda_n\big) - I_s \\ &= \big(\bar{Q}_{n+1}^T \bar{Q}_{n+1} - I_s\big) - 2\Lambda_n + \mathcal{O}(\sqrt{h}), \end{aligned} \tag{4.54}$$

where $\mathcal{O}(\sqrt{h})$ denotes a matrix whose 2-norm has order $\sqrt{h}$. Solving for Λ_n, we find

$$\Lambda_n = \frac{1}{2}\big(\bar{Q}_{n+1}^T \bar{Q}_{n+1} - I_s\big) + \mathcal{O}(\sqrt{h}).$$

Neglecting the terms of order $\sqrt{h}$ and higher, we obtain the following quasi-Newton scheme: setting $Q^{(0)} = \bar{Q}_{n+1}$, repeat the iteration

$$\begin{aligned} &Q^{(k+1)} = Q^{(k)} - Q_n \Lambda^{(k)}, \\ &\text{where } \Lambda^{(k)} = \frac{1}{2}\big((Q^{(k)})^T Q^{(k)} - I_s\big), \end{aligned} \tag{4.55}$$

until the process reaches convergence and set $Q_{n+1} = Q^{(k+1)}$. To assess whether convergence has been reached, a tolerance on the 2-norm of $\Lambda^{(k)}$ can be assigned: $\|\Lambda^{(k)}\| \le \mathrm{TOL}$. However in practice, to ensure that the process ends and to avoid undesirable overhead we typically prefer to either combine this stopping criterion with a limit for the number K of iterations, or use a fixed number of iterations K. Note that estimate Eq. (4.54) ensures that a small number of iterations K is sufficient for the constraint to be satisfied up to a small error.

The initialization for the constrained weights is performed following [248], which is an built-

in option in PyTorch. Other parameters are initialized using the standard PyTorch initialization [219, 96] unless otherwise indicated. Constraints are applied layer-wise, where for convolutional layers with weight tensors of the size $n_l \times n_{l-1} \times n_h \times n_w$ (where n_h and n_w are the height and width of the kernel) the weight matrices are reshaped as $n_l \times n_{l-1} n_h n_w$. For CNNs these reshaped matrices are typically rectangular. If they are thin, but long (i.e., $n_l > n_{l-1} n_h n_w$) we apply the constraint $W^T W = I$, but if they have more columns than rows we apply the constraint $W W^T = I$.

4.7.2.6 Orthogonality Constraint, Underdamped Langevin (o-CoLud)

To discretize the underdamped Langevin constrained dynamics, we need the orthogonal projection Π onto the cotangent space $T_Q^*\Sigma$. As the constraint Eq. (4.50) is given in a matrix form, using the formula Eq. (4.23) is not very convenient so we will rather derive Π from its projection property.

Using Eq. (4.51), we find that for $1 \leq i, j \leq s$

$$0 = \sum_{k=1}^{s} \sum_{l=1}^{r} \partial_{Q_{kl}} g_{ij}(Q) P_{kl} = (P^T Q + Q^T P)_{ij},$$

which leads to the following convenient expression for the cotangent space

$$T_Q^*\Sigma = \left\{ P \in \mathbb{R}^{r \times s} \mid P^T Q + Q^T P = 0_s \right\}.$$

Now, given $\bar{P} \in \mathbb{R}^{r \times s}$ we want to find a symmetric $s \times s$ matrix Λ such that $P = \bar{P} - Q\Lambda$ belongs to $T_Q^*\Sigma$, i.e.,

$$0_s = P^T Q - Q^T P = \bar{P}^T Q + Q^T \bar{P} - \Lambda Q^T Q - Q^T Q \Lambda.$$

This equation is easily solved for $Q \in \Sigma$ and we find $\Lambda = \frac{1}{2}(\bar{P}^T Q + Q^T \bar{P})$. We obtain the following expression for the projection onto the cotangent space:

$$\begin{aligned} &\Pi_Q : \mathbb{R}^{r \times s} \to \mathbb{R}^{r \times s}, \\ &\bar{P} \mapsto \Pi_Q \bar{P} = \bar{P} - \frac{1}{2} Q(\bar{P}^T Q + Q^T \bar{P}). \end{aligned}$$

We then verify that Π_Q is indeed a projection onto the cotangent space $T_Q^*\Sigma$ (i.e., $\Pi_Q \bar{P} \in T_Q^*\Sigma$ $\forall \bar{P} \in \mathbb{R}^{r \times s}$ and $\Pi_Q^2 = \Pi_Q$) and that this projection is orthogonal with respect to the Frobenius inner product on $\mathbb{R}^{r \times s}$ (i.e., $\langle \bar{P} - \Pi_Q \bar{P}, P \rangle = 0$, where $\langle A, B \rangle = \mathrm{tr}(A^T B)$).

A component. For the orthogonal constraint, the A component in Eq. (4.14) can only be solved approximately. A simple yet efficient discretization of A is the RATTLE scheme (see e.g. [160][Chapter 8]):

$$\begin{aligned} &Q_{n+1} = Q_n + h P_{n+1/2}, \\ &P_{n+1/2} = P_n - Q_n \Lambda_{n+1/2} \\ &\quad \text{where } \Lambda_{n+1/2} \text{ is s.t. } Q_{n+1}^T Q_{n+1} = I_s, \\ &P_{n+1} = P_{n+1/2} - Q_{n+1} \Lambda_{n+1} \\ &\quad \text{where } \Lambda_{n+1} \text{ is s.t. } Q_{n+1}^T P_{n+1} + P_{n+1}^T Q_{n+1} = 0_s. \end{aligned} \tag{4.56}$$

Denoting $\bar{\Lambda}_{n+1/2} = h\Lambda_{n+1/2}$, $\bar{P}_{n+1} = P_{n+1/2}$ and using the projection operator Π_Q, Eq. (4.56)

can be rewritten as

$$\begin{aligned}
&\bar{Q}_{n+1} = Q_n + hP_n,\\
&Q_{n+1} = \bar{Q}_{n+1} - Q_n\bar{\Lambda}_{n+1/2}\\
&\quad \text{where } \bar{\Lambda}_{n+1/2} \text{ is s.t. } Q_{n+1}^T Q_{n+1} = I_s \quad (\text{use Eq. (4.55)}),\\
&\bar{P}_{n+1} = P_n - \frac{1}{h}Q_n\bar{\Lambda}_{n+1/2}, \qquad P_{n+1} = \Pi_{Q_{n+1}}\bar{P}_{n+1}.
\end{aligned} \tag{4.57}$$

As in the overdamped case, we may now use the quasi-Newton scheme Eq. (4.55) for the projection step (to approximate $\bar{\Lambda}_{n+1/2}$). Using K iterations of the quasi-Newton scheme Eq. (4.55) (i.e., $Q_{n+1} = Q^{(K)}$), we verify that $-Q_n\bar{\Lambda}_{n+1/2}$ satisfies

$$\begin{aligned}
-Q_n\bar{\Lambda}_{n+1/2} &= \sum_{k=0}^{K-1} Q_n\Lambda^{(k)} = \sum_{k=0}^{K-1} Q^{(k+1)} - Q^{(k)}\\
&= Q^{(K)} - Q^{(0)} = Q_{n+1} - \bar{Q}_{n+1},
\end{aligned}$$

so that $\bar{P}_{n+1} = P_n + \frac{1}{h}(Q_{n+1} - \bar{Q}_{n+1})$.

We obtain the following full discretization of the underdamped Langevin dynamics with orthogonality constraint. The initialization for the constrained weights is performed following [248]. Corresponding momenta are initialized as the initial gradients (equivalently to standard PyTorch initialization) and subsequently projected using $P_0 = \bar{P}_0 - \frac{1}{2}Q_0(\bar{P}_0^T Q_0 + Q_0^T \bar{P}_0)$. The A,B,O steps are then given as:

$$\text{(A, OG)} \begin{cases}
\bar{Q}_{n+1} = Q_n + hP_n, \quad Q^{(0)} = \bar{Q}_{n+1},\\
\text{for } k = 0 \text{ to } K-1\text{:}\\
\qquad Q^{(k+1)} = Q^{(k)} - Q_n\Lambda^{(k)},\\
\quad \text{where } \Lambda^{(k)} = \frac{1}{2}\Big(\big(Q^{(k)}\big)^T Q^{(k)} - I_s\Big),\\
Q_{n+1} = Q^{(K)},\\
\bar{P}_{n+1} = P_n + \frac{1}{h}\big(Q_{n+1} - \bar{Q}_{n+1}\big),\\
P_{n+1} = \Pi_{Q_{n+1}}\bar{P}_{n+1} = \bar{P}_{n+1}\\
\quad - \frac{1}{2}Q_{n+1}\Big(\bar{P}_{n+1}^T Q_{n+1} + \big(Q_{n+1}\big)^T \bar{P}_{n+1}\Big).
\end{cases}$$

$$\text{(B, OG)} \begin{cases}
Q_{n+1} = Q_n,\\
\bar{P}_{n+1} = P_n - h\nabla_Q U(Q_n),\\
P_{n+1} = \Pi_{Q_n} P_{n+1}\\
= \bar{P}_{n+1} - \frac{1}{2}Q_n\Big(\bar{P}_{n+1}^T Q_n + \big(Q_n\big)^T \bar{P}_{n+1}\Big),
\end{cases}$$

$$\text{(O, OG)} \begin{cases}
Q_{n+1} = Q_n,\\
\bar{P}_{n+1} = e^{-\gamma h}P_n + \sqrt{\beta^{-1}(1 - e^{-2\gamma h})}R_n,\\
P_{n+1} = \Pi_{Q_n}\bar{P}_{n+1}\\
= \bar{P}_{n+1} - \frac{1}{2}Q_n\Big(\bar{P}_{n+1}^T Q_n + \big(Q_n\big)^T \bar{P}_{n+1}\Big),
\end{cases}$$

where R_n is a matrix of independent standard normal random variables.

4.7.3 Additional Numerical Details and Results

We perform all experiments using PyTorch [219] on NVIDIA DGX-1 GPUs. We compare our constrained methods with PyTorch's SGD with momentum optimiser. Unless otherwise indicated, we use for SGD $h = 0.1$ and $mom = 0$ (to compare with our constrained overdamped

Langevin method) or $mom = 0.9$ (to compare with our constrained underdamped Langevin method). We use standard PyTorch initialization for all unconstrained parameters [96, 219]. Below we provide implementation details for all our experiments.

4.7.3.1 Orthogonality Constraints

To create the first class of the the planar spiral dataset binary classification problem (as used to produce Figure 4.4 and Figure 4.5) is generated using

$$\begin{aligned} x &= 2\sqrt{t}\cos(8\sqrt{t}\pi) + 0.02\mathcal{N}(0,1), \\ y &= 2\sqrt{t}\sin(8\sqrt{t}\pi) + 0.02\mathcal{N}(0,1), \end{aligned} \tag{4.58}$$

where t is drawn repeatedly from the uniform distribution $\mathcal{U}(0,1)$ to generate data points. The other class of this dataset is obtained by shifting the argument of the trigonometric functions by π. For our experiments we used 500 training data, 1000 test data points and 5% subsampling.

To generate the results presented in Figure 4.4 and 4.5, which show the effect of orthogonality constraints on this spiral dataset, we use multi-layer perceptrons with ReLU activation and binary cross entropy (BCE) loss. In our experiments we vary the number of 100-node hidden layers of the multi-layer perceptrons. To compare the performance of our o-CoLod constrained method with standard SGD we set the temperature $\tau = 0$ and $h = 0.1$ for all methods to generate Figure 4.4. For Figure 4.5 we do a grid-search to find the optimal value of the penalty strength for the orthogonal regularization approach with respect to the stepsize. In Figure 4.8 we show the effect of using a small temperature perturbation $\tau =$ 1e-6. The size of the temperature parameter was chosen to approximately match observed fluctuations in the loss function. A more precise parameterization is left for a subsequent work.

We also applied our orthogonality-constrained methods to the ResNet-34 architecture on CIFAR-10 image classification data [146], see Figure 4.6. The input data is pre-processed using random crop (pad=4), random horizontal flip, and normalization. In this setting, running SGD with orthogonal initialization worsened the generalization performance of the resulting net and hence the standard PyTorch initialization was used for SGD. We train for 150 epochs and use a batchsize of 128. In Figure 4.9 we compare the overdamped variant o-CoLod (with $\tau = 0$) to its unconstrained counterpart. We observe that the use of an orthogonality constraint gives lower test loss throughout training.

4.7.3.2 Circle Constraints

For the results shown in Figure 4.1, Figure 4.2, Figure 4.3, and Table 4.1 the first class of the dataset is generated using

$$\begin{aligned} x &= \sqrt{t}\cos(4\sqrt{t}\pi) + 0.05\mathcal{N}(0,1), \\ y &= \sqrt{t}\sin(4\sqrt{t}\pi) + 0.05\mathcal{N}(0,1), \end{aligned} \tag{4.59}$$

where t is repeatedly drawn from $\mathcal{U}(0,1)$. The other class is obtained by shifting the argument of the trigonometric functions by π. For our experiments we used 100 training data points, 2000 test data points and 2% subsampling. We use a 500-node single hidden layer perceptron, with ReLU activation and BCE loss. We choose the optimal weight decay value for SGD through line search. The results in Figure 4.3 were obtained by computing the gradient of the predictions of a trained classifier (after 10,000 epochs) on a 1000x1000 grid using second order accurate central differences.

For our Fashion-MNIST [292] example we reduce the number of training data samples to 10,000 and we increase the number of test data samples to 60,000. We use a 1000-node SHLP with ReLU activation, cross entropy loss and batchsize 128. Our main result with our circle constrained approach is presented in Figure 4.7, the accompanying mean test accuracies with standard deviations are: 87.63±0.04% (c-CoLud), 87.39 ±0.06% (SGD), 87.47 ±0.38% (SGD with WD = 1e-4), 87.29 ±0.58% (SGD with WD = 5e-5), 87.45 ±0.06% (SGD with WD = 1e-5). Hyperparameters SGD: $h = 0.1, mom = 0.8$. Hyperparameters c-CoLud: $h = 0.3, \gamma = 1, r_0 = 0.05, r_1 = 0.1, \tau = 0$.

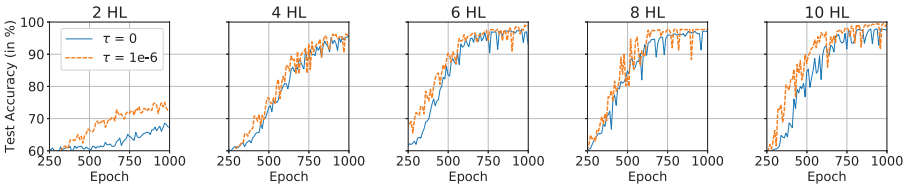

Figure 4.8: The effect of temperature on the performance of the o-CoLA-od optimizer for the 4-turn spiral dataset (same set-up as for Figure 4.4). MLPs with varying numbers of hidden layers (HL) were trained using o-CoLA-od with $h = 0.1$ and either $\tau = 0$ (blue line) or $\tau =$ 1e-6 (orange line). Results are averaged over 5 runs. The use of temperature is shown to speed up training and often slightly increases the obtained test accuracies.

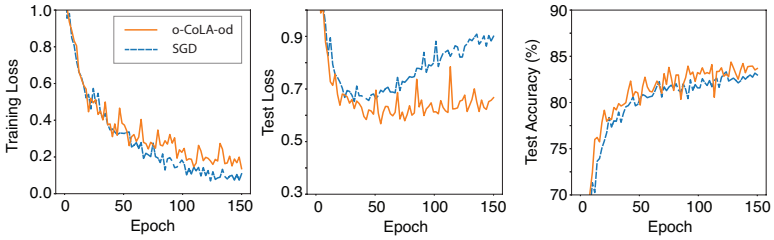

Figure 4.9: Training loss (left), test loss (middle) and test accuracy (right) of a ResNet-34 trained using SGD vs. o-CoLA-od on CIFAR-10 data, $h = 0.1$ (averaged over 5 runs). The orthogonality constraint provides modestly higher test accuracy and inhibits overfitting.

In Table 4.3 we present extensive hyperparameter tests for the test accuracy and test loss obtained after 400 epochs (averaged over 5 runs) using SGD-m with and without weight decay (WD). In Figure 4.10 we show that both the test loss and the maximum magnitude of the weights of the network remains small and stable throughout training for our circle constrained approach, while SGD shows signs of overfitting.

Table 4.3: These results are obtained for the Fashion-MNIST dataset using SGD with momentum to train a 1000-node SHLP. The results presented in the two right-hand columns are all obtained with weight decay set to 1e-4. We found this value to give the best results for SGD-m during a hyperparameter search. In comparison to the results for SGD-m shown in this table our circle constrained net reaches test accuracy **87.63%**, with test loss **0.386** without using weight decay (see Figure 4.7). Hence it outperforms standard SGD with momentum both with and without weight decay.

SGD with mom		no WD		with WD	
		Test Accuracy	Test Loss	Test Accuracy	Test Loss
h = 0.2	mom = 0.8	87.18%	1.06	84.05%	0.696
	mom = 0.7	87.38%	0.890	87.0%	0.547
h = 0.1	mom = 0.9	86.97%	1.133	85.35%	0.634
	mom = 0.8	87.39%	0.824	87.47%	0.531
	mom = 0.7	87.39%	0.750	87.25%	0.517
h = 0.05	mom = 0.95	86.67%	1.226	85.63%	0.623
	mom = 0.9	87.33%	0.837	86.24%	0.569
	mom = 0.8	87.27%	0.719	87.33%	0.511

We also evaluate the performance of a small transformer model [273] on the Penn Treebank [193] and Wikitext-2 [199] data. The transformer has 2 encoder layers. Each encoder layer consists of self-attention with 2 heads and a feedforward network with 200 nodes followed by layer norms. We use batchsize 1024 for the Penn Treebank data and batchsize 128 for the Wikitext-2 dataset. We present the lowest validation loss obtained in 200 epochs by SGD-m

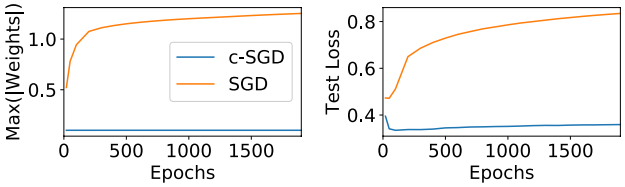

Figure 4.10: Result is obtained for the Fashion-MNIST dataset with the same hyperparameter settings as in Figure 4.7. We observe that the maximum absolute size of weights in the output layer of the network (left) and test loss (right) remain small and stable throughout training for the circle constrained method (c-SGD or c-CoLA-ud). In contrast SGD shows clear signs of overfitting.

and our circle constrained method c-CoLud in Table 4.2. In Table 4.4 we provide a comparison with weight decay.

Table 4.4: Minimum validation loss on Penn Treebank and Wikitext-2 using a transformer trained using SGD-m. We found weight decay set to WD = 1e-4 to give the best results for SGD-m. In comparison, the transformer trained using c-CoLA-ud obtains a minimum validation loss of 4.81 (Penn Treebank) and 5.09 (Wikitext-2). Using c-CoLA-ud therefore outperforms standard SGD-m in the case without WD, but does less well than SGD-m with weight decay, if the magnitude of the weight decay has been carefully tuned. In contrast, for Fashion-MNIST image data using a MLP we find that c-CoLA-ud outperforms both SGD-m with weight decay and SGD-m without weight decay (see Table 4.3).

Optimizer	Penn Treebank	Wikitext-2
Without WD		
$mom = 0.7$	4.87	5.13
$mom = 0.8$	4.83	5.13
$mom = 0.9$	4.84	5.13
With WD		
$mom = 0.7$	4.84	5.01
$mom = 0.8$	4.77	5.02
$mom = 0.9$	4.77	5.02

4.7.3.3 Curvature

It is difficult to establish a commonly agreed definition of curvature for a boundary that is potentially non-differentiable at a finite number of points. We computed our curvature estimates using the method described below which we suggest is indicative of the curvature of the locally smoothed classification boundary and allows us to compare the relative curvature estimates of classifiers trained using different optimizers.

We evaluate the smoothness properties of our trained classifiers after a fixed number of 10,000 epochs. The curvature of a level curve $\phi(x, y) = 0$ is defined as [223]:

$$\kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} = \frac{\phi_{xx}\phi_y^2 - 2\phi_y\phi_x\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}}$$

However, since we do not have access to the exact form of ϕ, we fit a contour to the model's predictions on a 1000x1000 grid using matplotlib.pyplot.contour, which returns an array containing the coordinates of points along the contour. We view these as discrete samples from the parametric curve $(x(t), y(t))$. The gradients of these are computed using second order

accurate central differences. This can then be used to compute the approximate curvature [91]:

$$\kappa = \frac{|x''y' - x'y''|}{(x'^2 + y'^2)^{3/2}} \tag{4.60}$$

Although this results in a rough estimate, by averaging our results over 100 runs, we suggest this gives us some insight on relative curvature estimates of classifiers trained using different optimizers.

In Table 4.5 and Figure 4.11 we study the effect of varying the temperature hyperparameter τ, which controls the additive noise level (see Eq. (4.7)), on the generalization performance and curvature of the resulting classifiers on the spiral dataset defined by Eq. (4.59). We show that there appears to be an ideal choice of temperature (in this case τ = 5e-5), for which the best generalization performance is obtained using our circle constrained approach. We also show that the trained classifier which has the lowest curvature estimate also obtains the best generalization performance.

Table 4.5: Same set-up as for Figure 4.1, 4.2, and Figure 4.11. We present the mean curvature, standard deviation (std), and maximum (max) curvature of classifier boundaries obtained using our constrained approach with different values of the temperature τ. The lowest curvature is obtained using τ = 5e-5, which also corresponds to the classifier which obtains the best generalization performance (see Figure 4.11). These results are averaged over 100 runs.

	Curvature Approximation		
τ for C-SGLD	Mean	Std	Max
$\tau = 0$	9.38	317	$5.58 \cdot 10^5$
τ = 1e-6	9.01	273	$1.63 \cdot 10^6$
τ = 5e-6	7.75	166	$5.86 \cdot 10^5$
τ = 1e-5	7.06	108	$4.06 \cdot 10^5$
τ = 5e-5	6.08	40.8	$1.43 \cdot 10^5$
τ = 1e-4	7.62	178	$9.47 \cdot 10^5$
τ = 5e-4	15.9	850	$5.07 \cdot 10^6$

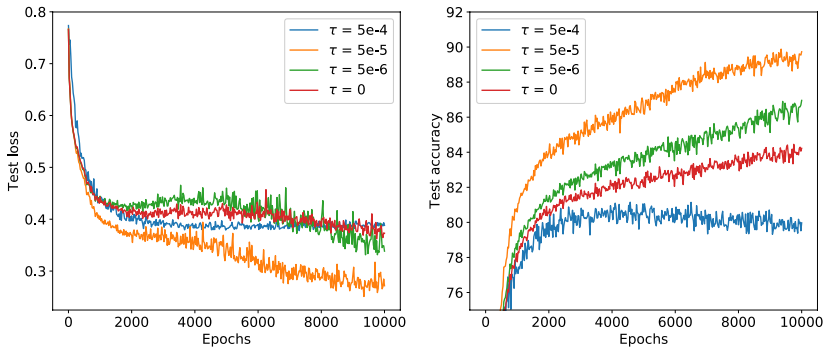

Figure 4.11: Test loss and test accuracy averaged over 100 runs for constrained approaches with varying levels of additive noise, i.e., with different values of the temperature hyperparameter τ. The set-up is the same as for Figure 4.1 and Figure 4.2, i.e., we train a 500-node single hidden layer perceptron for a spiral binary classification problem (Eq. (4.59)). Hyperparameter settings: $h = 0.05$, 2% subsampling, $r_0 = 1, r_1 = 5$ (see Eq. (4.2)). The best performance is obtained using temperature τ = 5e-5. This is also the temperature that results in the classifier with the lowest curvature estimate (see Table 4.5).

Chapter 5

Multirate Training of Neural Networks

The results presented in this chapter are currently under review and are joint work with Benedict Leimkuhler [278].

Abstract

We propose multirate training of neural networks: partitioning neural network parameters into "fast" and "slow" parts which are trained on different time scales. By choosing appropriate partitionings we can obtain substantial computational speed-up for transfer learning tasks. We show for applications in vision and natural language processing that we can fine-tune deep neural networks in almost half the time, without reducing the generalization performance of the resulting models. We analyze the convergence properties of our multirate scheme and draw a comparison with vanilla SGD. We also discuss splitting choices for the neural network parameters which could enhance generalization performance when neural networks are trained from scratch. A multirate approach can be used to learn different features present in the data and as a form of regularization. Our work unlocks the potential of using multirate techniques for neural network training and provides several starting points for future work in this area.

5.1 Introduction

Multirate techniques have been widely used for efficient simulation of multiscale ordinary differential equations (ODEs) and partial differential equations (PDEs) [233, 77, 78, 92, 66, 46]. Motivations for using multirate techniques are the presence of fast and slow time scales in the system dynamics and to simulate systems which are computationally infeasible to evolve with a single stepsize.

In Figure 5.1 we demonstrate the effect of latent multiple time scales in deep learning applications. Here we have trained a WideResNet-16 architecture on a patch-augmented CIFAR-10 dataset [178] using SGD with momentum and weight decay and different learning rates. Some images contain both the patch and CIFAR-10 data, while other images only contain the patch or are patch-free. When training using a large learning rate, the network is unable to memorize the patch, but achieves high accuracy on patch-free data. Meanwhile, when training using a small learning rate the network can memorize the patch quickly, but the accuracy on clean data is lower. We demonstrate that a multirate approach trained on two time scales can both memorize the patch and obtain a high accuracy on the patch-free data. Multirate methods thus show potential for simultaneously gathering information on different features of the data, for settings where fixed learning rate approaches fail.

The idea of using fast and slow weights in a machine learning context has been around for a long time [68, 100, 10], originally inspired by neuroscience as synapses in the brain have dynamics at different time scales. However, the use of multirate methods has so far been largely

overlooked for this area. In this work we seek to change this. We illustrate the benefit of using multirate techniques for a variety of neural network training applications. We describe connections with the current machine learning literature in Section 5.6. As main application we use a multirate approach to obtain computational speed-up for transfer learning tasks by evaluating the gradients associated with the computationally expensive (slow) part of the system less frequently (Section 5.4). The multirate algorithm we propose is straightforward to implement and can be used in combination with existing PyTorch optimizers.

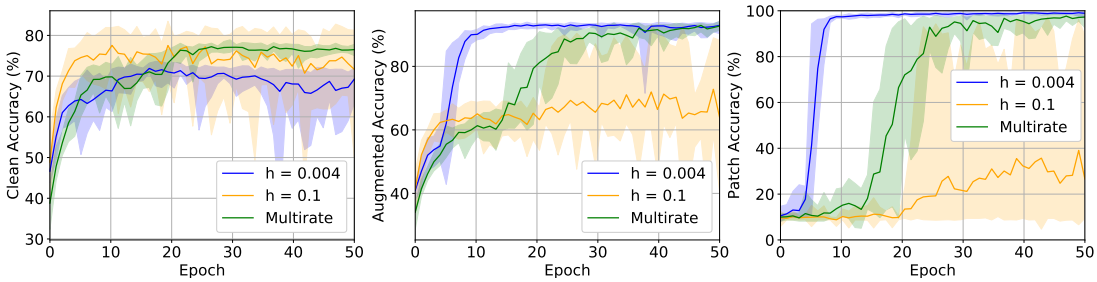

Figure 5.1: WideResNet-16 trained on patch-augmented CIFAR-10 [178]. Of the training data: 20% is patch-free, 16% has only the patch, and the rest has both data and patch. More details are provided in Section 5.9.3. Left: clean validation set. Middle: augmented data with patches. Right: patch-only data. A net trained using a small learning rate (blue) learns the patch quickly, whereas a large learning rate (orange) gives higher accuracy on clean data. A multirate scheme (green) trained on both time scales ($h_F = 0.004$, $h_S = 0.1$, see Section 5.3 and Section 5.9.3) is able to memorize the patches and to simultaneously obtain high accuracy on the clean data.

Our contributions are as follows:

- We propose multirate training of neural networks, which requires partitioning neural network parameters into fast and slow parts. We illustrate the versatility of this approach by demonstrating the benefits of different partitioning choices for different training applications.
- We describe a novel multirate scheme that uses linear drift of the slow parameters during the fast parameter update and show that the use of linear drift enhances performance. We compare its convergence properties to vanilla SGD.
- We use our multirate method to train deep neural networks for transfer learning applications in vision and natural language processing in half the time, without reducing the generalization performance of the resulting model.
- We propose a regularization technique for the training of neural networks, which randomly selects new subsets of the neural network to form the slow parameters using an iterative process.

We conclude that multirate methods can enhance neural network training and provide a promising direction for future theoretical and experimental work.

5.2 Background

Multirate methods use different stepsizes for different parts of the system. Faster parts are integrated with smaller stepsizes, while slow components are integrated using larger stepsizes, which are integer multiples of the fast stepsize. Multirate methods have been used for more than 60 years [233] in a wide variety of areas [66, 92]. Gear [77] analyzed the accuracy and stability of Euler-based multirate methods applied to a system of ODEs with slow and fast components.

The system of ODEs that forms the starting point for most neural network training schemes is $\mathrm{d}\theta = G(\theta)\mathrm{d}t$, where $\theta \in \mathbb{R}^n$ are the neural network parameters and G represents the gradient of the loss of the entire dataset. As a starting point for our multirate approach we partition the parameters as $\theta = (\theta_F, \theta_S)$, with $\theta_F \in \mathbb{R}^{n_F}, \theta_S \in \mathbb{R}^{n_S}$, $n = n_F + n_S$, and obtain system of ODEs

$$\mathrm{d}\theta_F = G_F(\theta)\mathrm{d}t, \quad \mathrm{d}\theta_S = G_S(\theta)\mathrm{d}t, \tag{5.1}$$

where G_F and G_S are the gradients with respect to θ_F and θ_S, respectively. For neural network training the loss gradient is typically evaluated on a randomly selected subset of the training data and the pure gradient in Eq. (5.1) is subsequently replaced by a noisy gradient which we denote $\tilde{G}(\theta)$. In the stochastic gradient Langevin dynamics method of Welling and Teh [285], the system is further driven by constant variance additive noise. Further, most training procedures incorporate momentum [226, 263]. As a somewhat general model, one may consider a partitioned underdamped Langevin dynamics system of the form

$$\begin{aligned} \mathrm{d}\theta_\alpha &= p_\alpha \ \mathrm{d}t, \quad \text{where } \alpha = F, S \\ \mathrm{d}p_\alpha &= \tilde{G}_\alpha(\theta)\mathrm{d}t - \gamma_\alpha p_\alpha \ \mathrm{d}t + \sqrt{2\gamma_\alpha \tau_\alpha} \ \mathrm{d}W_\alpha, \end{aligned} \tag{5.2}$$

with momentum $p = (p_F, p_S) \in \mathbb{R}^n$ and hyperparameters $\gamma_\alpha, \tau_\alpha > 0$. When evaluating the gradient on the full dataset, Langevin dynamics is provably ergodic [195], under mild assumptions, and samples from a known distribution. The temperature hyperparameter τ_α controls the driving noise and thus the transition between pure optimization and pure sampling; when small it can benefit neural network optimization [163, 287]. In this work we will focus on the case $\tau_\alpha = 0$, which corresponds to standard stochastic gradient descent (SGD) with momentum under re-scaling of the hyperparameters, however, our multirate approach can easily be extended to the more general case. We have also opted to use the same momentum hyperparameter (γ_α in Eq. (5.2)) for both subsystems to provide a fair comparison with standard SGD with momentum. Using different optimizer hyperparameters, as well as exploration of methods which combine different optimizers for different components, is left for future study (see Section 5.7 and Section 5.9.1). Algorithms can easily be designed based on partitioning into multiple independent components (not just two) evolving at different rates, as we illustrate in Section 5.3.1.

5.3 Multirate Training of Neural Networks

In their most general formulation multirate methods involve separating the model parameters (and if applicable, their accompanying momentum variables) Θ into multiple components $\Theta_1, ..., \Theta_N$ corresponding to different time scales. Slow parameters are updated less frequently than their fast counterparts but with larger stepsizes. Synchronization of the parts occurs every slow time step. This is illustrated for two time scales (and accompanying fast Θ_F and slow Θ_S parameters) in Figure 5.2. In this section we propose a multirate technique that can be directly applied to the training of neural networks and discuss application-specific appropriate choices for the fast and slow parameters.

5.3.1 A Partition-based Multirate Approach

The type of multirate algorithms we consider in this work take the following approach for two time scales:

1. Separate model parameters into a fast and slow part.

2. At every step, compute the gradients with respect to the fast variables. Update the fast variables using the optimizer of your choice with fast stepsize h_F.

3. Every $k \in \mathbb{Z}_+$ steps: Compute gradients with respect to the slow variables. Update slow variables using the optimizer of your choice with slow stepsize $h_S = kh_F$.

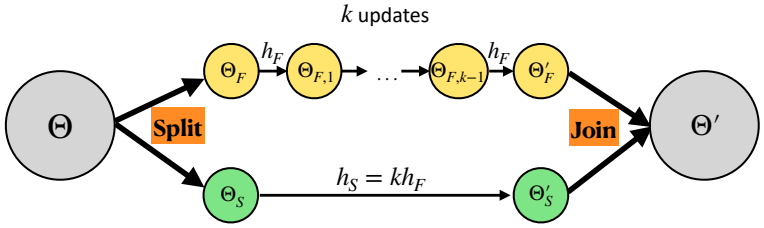

Figure 5.2: The basic principle of the multirate techniques considered in this work is illustrated for two time scales in this figure. We first split our parameters into fast and slow components, Θ_F and Θ_S, respectively. The fast components are then updated every step with stepsize h_F, whereas the slow components are updated every k steps with stepsize $h_S = k \cdot h_F$.

This multirate approach can be combined with different optimization schemes, such as of the form in Eq. (5.2). In this work, for our analysis and numerical experiments we shall focus on using as base algorithm stochastic gradient descent (SGD), where the gradients are computed for every mini-batch of m training examples. We will compare our multirate approach with PyTorch's standard SGD with momentum implementation [219] and hence for consistency we present our method in the same notation and manner as used in the PyTorch code. Our multirate scheme is described by Algorithm 5.1. We refer to the model parameters and momenta associated with the slow system as θ_S and p_S, respectively, and for the fast system as θ_F and p_F. We denote by $\mathcal{L}(\theta_S, \theta_F)$ the neural network loss as evaluated on a minibatch of training examples. We use the cross-entropy loss for classification tasks. We use μ to denote the momentum hyperparameter, which we typically set to $\mu = 0.9$.

We discuss variations of Algorithm 5.1 such as combining this multirate approach with other optimizers, the use of weight decay, or using different initializations for the fast and slow systems in Section 5.9.1.

Linear Drift. In Algorithm 5.1 we continuously push the slow parameters along a linear path defined by their corresponding momenta. This means that although the gradients for the slow parameters are only computed every k steps, the slow neural network parameters do get updated every step in the direction of the previous gradient. This is a novel technique for multirate training, where approaches similar to that in Algorithm 5.2 are more prevalent. We compare these approaches in ablation studies in Section 5.4.3 and show that the use of linear drift enhances performance.

Algorithm 5.1 Multirate SGD with linear drift

$p_S := \mu p_S + \nabla_{\theta_S} \mathcal{L}(\theta_S, \theta_F)$
for $i = 1, 2, ..., k$ **do**
  $p_F := \mu p_F + \nabla_{\theta_F} \mathcal{L}(\theta_S, \theta_F)$
  $\theta_F := \theta_F - \frac{h}{k} p_F$
  $\theta_S := \theta_S - \frac{h}{k} p_S$
end for

Algorithm 5.2 Multirate SGD no linear drift

$p_S := \mu p_S + \nabla_{\theta_S} \mathcal{L}(\theta_S, \theta_F)$
$\theta_S := \theta_S - h p_S$
for $i = 1, 2, ..., k$ **do**
  $p_F := \mu p_F + \nabla_{\theta_F} \mathcal{L}(\theta_S, \theta_F)$
  $\theta_F := \theta_F - \frac{h}{k} p_F$
end for

Choice of Partitioning. Examples of possible separations of the model parameters into fast and slow components are layer-wise, weights vs. biases, or by selecting (random) subgroups. The appropriate separation is application-specific and will be discussed in more detail

in upcoming sections. In Section 5.4 we explore obtaining computational speed-up using Algorithm 5.1 through layer-wise partitioning, where our fast parameters are chosen such that the gradients corresponding to the fast system are quick to compute, while gradients of the full net are only computed every k steps. In Section 5.9.4 we study the effect of putting the biases of a neural network on the slow time scale. Finally, in Section 5.5.1 we use partitioning using random subgroups to develop a regularization technique for neural network training.

Extension to More Scales. Although we have presented the algorithm for two time scales, the scheme can easily be extended to more scales. To extend our framework to multiple components operating at r scales, one can use stepsizes $h_i = h_{i-1}/K_{i-1}$, $i = 1, 2, \ldots, r$, $K_i \in \mathbb{Z}_+$, recursively dividing the step sequences in Algorithm 5.1 and 5.2 into finer ones at each successive level of the parameter hierarchy.

5.3.2 Convergence Analysis

To study the convergence properties of multirate SGD in the non-convex setting we make the following (standard) assumptions:

Assumption 5.3.1. We assume the function $f : \mathbb{R}^n \to \mathbb{R}$ to be L-smooth, i.e., f is continuously differentiable and its gradient is Lipschitz continuous with Lipschitz constant $L > 0$

$$\|\nabla f(\varphi) - \nabla f(\theta)\|_2 \leq L\|\varphi - \theta\|_2, \ \forall \theta, \varphi \in \mathbb{R}^n. \tag{5.3}$$

Assumption 5.3.2. We assume that the second moment of the stochastic gradient is bounded above, i.e., there exists a constant M for any sample x_i such that

$$\|\nabla f_{x_i}(\theta)\|_2^2 \leq M, \quad \forall \theta \in \mathbb{R}^n. \tag{5.4}$$

Assumption 5.3.2 guarantees that the variance of the stochastic gradient is bounded. Under Assumption 5.3.1 and 5.3.2 we show in Section 5.9.2 that Theorem 5.3.3 holds for our layer-wise partitioned multirate SGD approach.

Theorem 5.3.3. We assume that 5.3.1 and 5.3.2 hold. Then

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] \leq \frac{2(f(\theta^0) - f(\theta^*))}{hT} + hLM\ell\left(\frac{1}{3}hLk^2 + 1\right), \tag{5.5}$$

where T is the number of iterations, L and M are as defined in Assumptions 5.3.1 and 5.3.2, ℓ is the number of layers, k is the additional hyperparameter associated with our multirate method, and θ^* is the optimal solution to $f(\theta)$.

From Theorem 5.3.3 one sees that as $T \to \infty$, the $hLM\ell\left(\frac{1}{3}hLk^2 + 1\right)$ term controls the upper bound. The expression in Theorem 5.3.3 is very similar to that obtained for vanilla SGD where the rightmost term is replaced by $hLM/2$ (see Section 5.9.2). Therefore, by decreasing the stepsize h, SGD can get closer to the neighborhood of a critical point. For our algorithm the choice of k (the additional hyperparameter introduced by our multirate method) also plays a role, where smaller values of k will lower the upper bound, but also increase the computational cost (in particular for our transfer learning application described in Section 5.4).

5.4 A Multirate Approach to Transfer Learning

We now discuss the application of our multirate scheme in the context of transfer learning, proposing a specific layer-wise division of the model parameters into fast and slow components

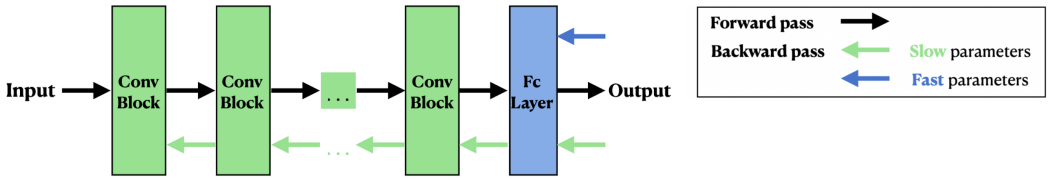

Figure 5.3: We indicate in blue the fast parameters and in green the slow parameters of a convolutional architecture, which consists of several convolutional blocks (conv block) and fully connected (fc) layer(s). When setting the fast parameters to be the final fc layer(s) (and optionally the conv block directly preceding it), the gradient computation for the backpropagation algorithm is very fast.

within Algorithm 5.1. We will see that this can significantly reduce the computational cost of fine-tuning.

The use of pre-trained deep neural networks has become a popular choice of initialization [53, 298]. These pre-trained networks are readily available through popular machine learning libraries such as PyTorch [219], and are usually trained on large datasets, such as ImageNet for vision applications [116] or large text corpora for natural language processing [110]. Using a pre-trained network as initialization has been shown to significantly accelerate training and typically improves the generalization performance of the resulting model [298, 98, 229]. The procedure is typically as follows: start with a pre-trained model, remove task-specific layers, and then re-train (part of) the network on the new target task. Later layers of neural networks tend to capture more task-specific knowledge, while early layers encode more general features, which can be shared across tasks [298, 94, 213, 231]. Hence to speed up training (and, in low target-data scenarios, to prevent overfitting), one sometimes does not re-train the full neural network, but only the later layers, in particular the final fully connected layer. This process is called fine-tuning [110, 47]. There exists a delicate balance between computational cost and generalization performance of fine-tuned deep neural network architectures. "Fine-tuning the whole network usually results in better performance" [174], but also increases the computational cost.

We propose to split a neural network into two parts, the fully connected layer parameters – the fast part – and the other parameters of the deep neural network – the slow part. The fast part is updated with a stepsize h/k, while the other part (the slow part) is only updated every k steps with a stepsize h. The slow part is very large compared to the fast part. For example, for a ResNet-34 architecture [97], the fully connected layer parameters (the fast part) only constitute 0.024% of the total parameters. Because of the way the backpropagation algorithm works, for our fast parameter updates we do not need to compute gradients for the full network, because the fast part is the very last layer of the neural network. This is illustrated schematically in Figure 5.3. Assuming that computing the gradients constitutes the largest cost of neural network training, we obtain significant speed-up by only needing to compute the full network gradients every k steps. We show that by choosing an appropriate k we can obtain a good generalization performance for nearly half the computational cost.

5.4.1 Numerical Results

We study the computational speed-up and generalization performance of Algorithm 5.1 compared to standard fine-tuning approaches. We consider a ResNet-34 architecture [97], which has been pre-trained on ImageNet [219], to classify CIFAR-10 data [146]. The standard procedure is to first replace the final fully connected layer of the architecture, to be able to match the number of classes of the target dataset, and then to retrain either the full architecture on the target set or only some of the bottom layers (with layer we refer to convolutional blocks in this setting). In contrast our multirate approach only updates the final fully connected (fc) layer every step and updates the rest of the parameters every 5 steps (we have set $k = 5$ in Algorithm 5.1). Both approaches use as base algorithm SGD with momentum without learning rate decay or weight decay. We use pre-trained ResNet architectures from PyTorch [219].

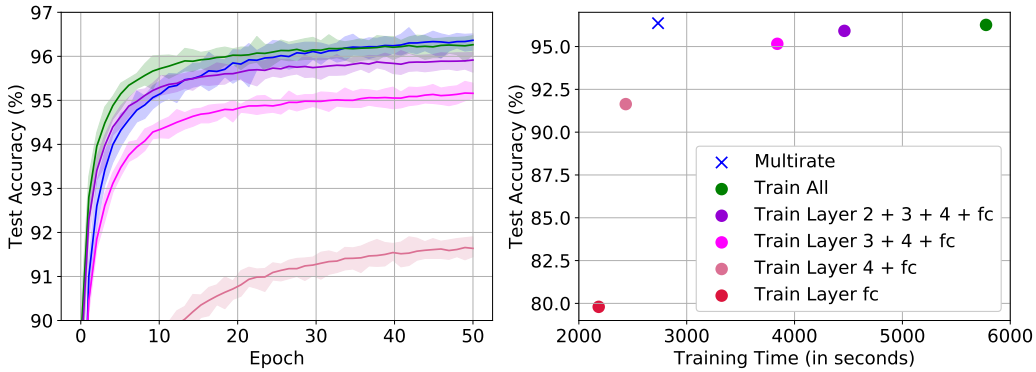

Figure 5.4: A pre-trained ResNet-34 being trained on rescaled CIFAR-10 using different fine-tuning approaches and our multirate approach (blue). Results are averaged over 20 runs and approaches are trained using SGD with momentum. We set $h/k = 0.001, k = 5$, and $\mu = 0.9$ in Algorithm 5.1. The highest test accuracy is reached using our multirate approach (blue), which can be used to train the net in almost half the time. Typical fine-tuning approaches only train the bottom layers of the network, e.g., layer 4 + fc, which results in a comparable speed-up, but much lower test accuracy.

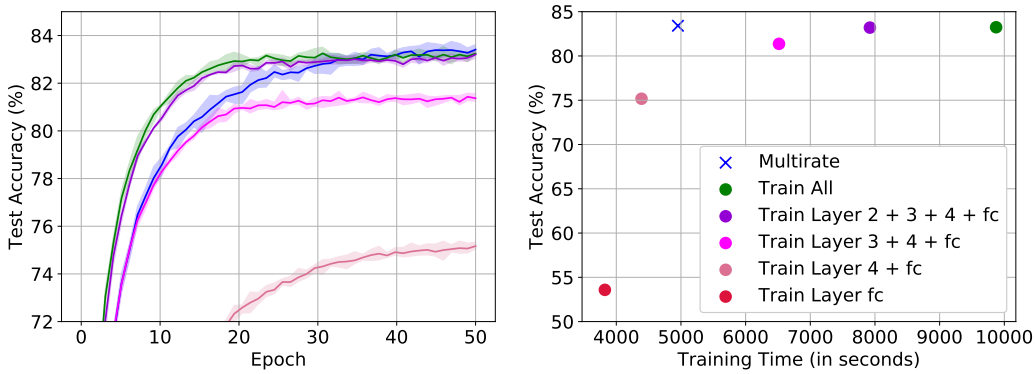

Figure 5.5: Same set-up as in Figure 5.4, but here we consider a pre-trained ResNet-50 architecture for rescaled CIFAR-100 data. The highest test accuracy is reached using our multirate approach (blue), which can be used to train the net in almost half the time.

We compare our approach to different fine-tuning approaches in Figure 5.4. Our multirate approach can be used to train the network in almost half the time, without affecting the test accuracy of the resulting net. We show in Figure 5.5 that the same observations hold for a ResNet-50 architecture pre-trained on ImageNet, which is fine-tuned on CIFAR-100 data. We also test our multirate approach on natural language data and consider a pre-trained DistilBERT architecture (obtained from the HuggingFace, transformers library). We fine-tune DistilBERT on SST-2 data [260] and show the computational speed-up and maintained generalization performance obtained using our multirate approach in Figure 5.6. Just as for standard fine-tuning approaches, there exists a trade-off between generalization performance and training time. We find that also including the attention block directly preceding the final fully connected layer into the fast parameters further enhances the generalization performance, without significantly increasing the training time.

5.4.2 Complexity Analysis

The number of floating point operations (FLOPs) for a forward pass through a neural network forms the lower bound of the execution time [129]. The number of FLOPs required will depend on the architecture and amount of data, whereas the timing of the FLOPs depends on the

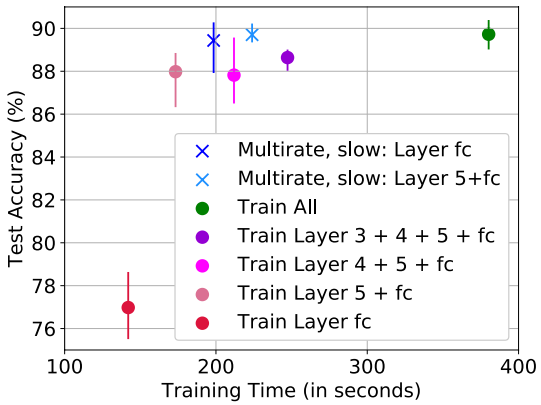

Figure 5.6: A pre-trained DistilBERT being trained on SST-2 data using different fine-tuning approaches, including our multirate approach (blue). Including the final attention block (together with the fc layer) in the fast parameters gives enhanced generalization performance for limited additional cost (light blue) and lowers the variance across multiple runs. We set $h/k =$ 1e-4, $k = 5$, $\mu = 0.9$ in Algorithm 5.1, batchsize $= 16$, and average results over 10 runs.

hardware used [227]. In our case, the number of FLOPs for the forward pass is the same for our multirate algorithm and standard approaches. The speed-up we obtain on ResNet and DistilBERT architectures arises from only needing to compute the gradients for the full net every k steps, while computing the gradients for the final fc layer(s) (and optionally the final convolutional/attention block) of the network every backward pass. The backward pass for the multirate method is hence a subset of the backward pass through the full net. A relevant measure of the speed-up obtained is therefore a ratio of the standard backward pass cost compared to the backward pass cost for our multirate approach over k steps

$$\frac{\text{backward pass full net}}{\text{backward pass multirate}} = \frac{kL}{L + (k-1)\ell}, \ \ell \ll L \tag{5.6}$$

for a neural network with L layers, where the slow parameters are the final ℓ layers of the network and $\ell \ll L$. When only fine-tuning the last ℓ layers, the L in the numerator is replaced by ℓ, but depending on the choice of ℓ this typically results in a lowered generalization performance. If the number of layers L is large, one can obtain a large speed-up using our multirate approach by only having to backpropagate through the full network every k steps. The exact speed-up obtained depends on the size of the layers and the hardware used. We performed our experiments in PyTorch on NVIDIA DGX-1 GPUs.

5.4.3 Ablation Studies

We consider a pre-trained DistilBERT being fine-tuned on SST-2 using our multirate approach (same set-up as in Figure 5.6) and perform ablation studies. In Table 5.1 we show that pushing the slow parameters along a linear path (as in Algorithm 5.1) improves the test accuracy compared to Algorithm 5.2. We also find that using the same stepsize $h_S = h_F$ for both the fast and slow parameters, but still only updating the slow parameters every k steps, does not lead to the same performance as using larger stepsize $h_S = k \cdot h_F$ for the slow parameters (Table 5.2). Finally, we provide a study on the role of k (Table 5.3 in Section 5.9.5). Every epoch the slow parameters only see $1/k$-th of the minibatches and are updated less frequently with a timestep k times larger than for the fast parameter update. We find optimal performance with $k = 5$, although training time can be decreased by choosing larger values of k. This trade-off needs to be taken into account when choosing k.

Table 5.1: **Effect of Continuously Pushing Slow Parameters along a Linear Path.** Same setting as in Figure 5.6, where fast parameters θ_F are set to be the fully connected (fc) layer + optionally the final attention block (denoted as layer 5) of a DistilBERT architecture. Results are presented over 10 runs. We compare Algorithm 5.1 (uses linear drift) to Algorithm 5.2. We find that pushing the slow parameters along a linear path during the fast parameter update improves the mean test accuracy.

θ_F are	*Linear*	Test acc		
Layer	*path?*	Mean	Min	Max
fc	Yes	**89.43%**	87.92%	90.28%
	No	88.69%	87.53%	89.68%
5 + fc	Yes	**89.70%**	89.35%	90.23%
	No	89.54%	88.91%	90.44%

Table 5.2: **Same learning rate for fast and slow parameters.** Same setting as in Figure 5.6 for a DistilBERT architecture. We study the effect of using the same learning rate for both the fast θ_F and slow θ_S parameters, i.e., $h_S = h_F =$ 1e-4 vs. using $h_S = k \cdot h_F =$ 5e-4. Results are presented over 10 runs. We observe that using a larger learning rate for the slow parameters aids performance.

θ_F are	*Higher* h	Test acc		
Layer	*for* θ_S *?*	Mean	Min	Max
fc	Yes	**89.43%**	87.92%	90.28%
	No	88.78%	87.64%	89.90%
5 + fc	Yes	**89.70%**	89.35%	90.23%
	No	89.29%	88.08%	89.95%

5.5 Multirate Training From Scratch

Whereas the previous section focused on using a multirate approach to obtain computational speed-up in transfer learning settings, we will now discuss how multirate training can be used to enhance the generalization performance of neural networks trained from scratch. We already illustrated this for the patch-augmented CIFAR-10 dataset in Figure 5.1, where a two-scale multirate approach can both memorize the patch and simultaneously obtain good performance on clean data, whereas fixed learning rate approaches fail to do both. In this section we will show the potential of using a multirate approach to regularize neural networks.

5.5.1 A Multirate Approach for Neural Network Regularization

Instead of using layer-wise partitioning, in this section we use randomly selected subsets of the neural network weight matrices (and optionally the biases) to form the slow parameters in Algorithm 5.1. Every k optimization steps a different subset of the network parameters is randomly selected. For the technique presented here we slightly modify Algorithm 5.1, by setting all the slow parameters θ_S to be zero during the k fast weight updates. We then re-activate the slow parameters and update them together with the fast weights in a single step with a larger time-step $h \cdot k$ for the slow weights. The base algorithm we use is again SGD and we incorporate our technique directly into a ready-to-use torch.optimizer. We discuss parallels between our method and dropout in the related work section. We show that this technique can be used to obtain enhanced performance on a single hidden layer perceptron applied to MNIST data in Figure 5.7. We compare our technique with dropout in Figure 5.8 for a small transformer trained on the Penn Treebank dataset [193] and obtain enhanced validation loss.

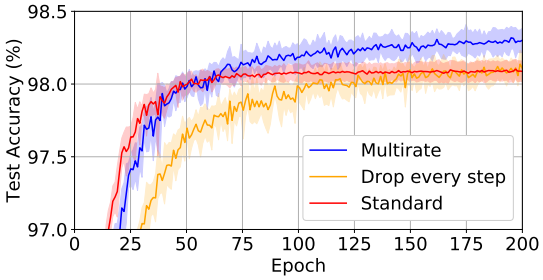

Figure 5.7: Single hidden layer perceptron trained on MNIST data using SGD with $h = 0.1$. Our multirate technique (blue) de-activates weights in the input and hidden layer with a probability of 0.8 and 0.5, respectively, and obtains a higher test accuracy than standard SGD (red). We also test removing the multirate component from our approach, which results in an algorithm which sets a different part of the weights to zero every step (orange), and does not perform as well as the multirate technique (blue).

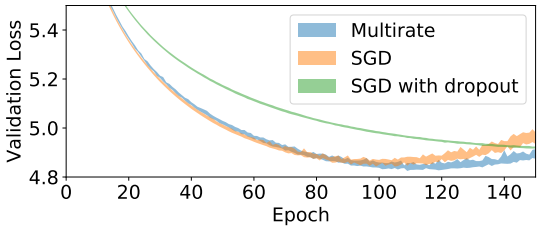

Figure 5.8: A transformer trained on Penn Treebank data [193] using SGD with $h = 0.1$ and batchsize 128. The transformer has 2 encoder layers, where each encoder layer consists of self-attention with 2 heads and a feedforward network with 200 nodes followed by layer norms. We compare our multirate technique (blue) with SGD (orange) and SGD with appropriately tuned dropout (green) for the encoder layers. Our multirate approach obtains lower validation loss.

5.6 Related Work

Intuition for using fast and slow weights in a machine learning context can be found in neuroscience, as synapses operate at different time scales. One of the earliest mentions of fast and slow weights in the machine learning literature was by Hinton and Plaut [100], who set each connection to have both a rapidly changing weight (which was supposed to act as a temporary memory) and a slowly changing weight which stores long-term knowledge. More recently, Ba et al. [10] used fast weights as a temporary memory to improve recurrent neural networks.

Our multirate approach to transfer learning (Section 5.4) has similarities to multiple time-stepping techniques used in molecular dynamics, such as r-RESPA [270, 271], where the fast dynamics is typically cheap to compute in comparison with the slow dynamics. This is similar to our transfer learning application, where we set the final layer of the net to be the fast part to obtain computational speed-up. Although the use of more refined transfer learning schemes may lead to further test accuracy enhancement, the focus of our approach is to obtain significant computational speed-up, while maintaining the same test accuracy. Further, we introduce linear drift of the slow parameters during the fast parameter update. The use of linear drift draws inspiration from the reversible averaging approach to multiple time-stepping by Leimkuhler and Reich [159], but forms a novel technique for multirate methods.

Further inspiration arises from the use of partitioned integrators for neural network training. It is well-known that different layers play different roles [307] and that later layers capture more task-specific knowledge, while early layers capture more general features, which can be shared across tasks [298, 94, 213, 231]. It is hence natural to train different layers of the neural network using layer-wise adaptive learning rates [299], layer-wise large-batch optimization techniques [301], different optimizers [163], or by only being Bayesian for certain layers [145, 202].

The multirate regularization technique (Section 5.5.1) has similarities to Dropout [262] and DropConnect [282]. Dropout can enhance the robustness and generalization performance of neural networks, and is used widely, although its performance in combination with batch normalization [121] is an ongoing area of research [187, 177, 42]. In contrast to dropout and its variants we do not modify the network architecture but incorporate our technique inside the optimizer, which randomly selects a subset of the weights as the slow part and keeps these de-activated for multiple steps. These weights are then re-activated and updated with a larger time-step, before de-activating a different subset. Instead of making strong claims, in this work we merely aim to illustrate the potential of using multirate techniques as a manner of regularization. We see an exploration of multirate variants of dropout as an exciting avenue for future work.

5.7 Discussion and Future Work

We outline possible directions for future work using multirate methods for neural network training along two axes: 1) different splitting choices of the neural network parameters into fast and slow parts and 2) using different optimizers or optimizer hyperparameter settings to train the different partitions. Our methods can be further generalized by combining them with well-known machine learning techniques, such as dropout or by exploring their behaviour under learning rate scheduling.

Splitting Choices. Multirate Algorithm 5.1 was based on a separation of the neural network parameters into fast and slow parts. We illustrated the potential of this approach for different parameters splittings. In Section 5.5.1 we used random subgroups, where we randomly selected a different subset of the network parameters to be the slow parameters every k optimization steps. In Section 5.4 we used a layer-wise partitioning, where we set the final layer(s) to be the fast parameters and the remaining parameters to be the slow parameters, for transfer learning applications.

An interesting direction for future work is to further explore layer-wise splitting when training networks from scratch, e.g., one could separate the early from later layers and train these with different time scales. It is important to note that the computational speed-up we obtained for the transfer learning setting by only computing the gradients for the final layer(s) at every step (Section 5.4), does not easily transfer to training from scratch, where the same approach significantly reduces generalization performance (earlier layers need to be updated more frequently to train well). Although for different choices of the layer-wise splitting the computational speed-up is lost, the use of layer-wise partitioned multirate algorithms may still enhance generalization performance compared to vanilla optimizers. You et al. [299] found that layer-wise adaptive learning rates can aid training. Further, network layers were shown to have different sensitivities to re-initialization [307] and to optimizer hyperparameter settings such as the learning rate [277]. This motivates training different layers with different initializations or learning rates. For transformers one could e.g. consider the self-attention layers as fast parameters. Another splitting option is to set the biases of a multi-layer perceptron architecture to be the slow parameters, while keeping the weights on the fast time scale. In Section 5.9.4 we show that using this approach we can obtain higher test accuracies on spiral data and provide ablation studies. This illustrates the potential of other parameter splittings.

Hybrid Optimization Schemes. For our multirate approach we partition the network into multiple parts which we train on different time scales. A natural extension is to also use different optimizers or optimizer hyperparameters to train the different partitions, e.g. to use SGD for the slow part, but SGD with momentum for the fast part(s), or using sampling techniques such as SGLD or discretized Langevin dynamics for certain parts. The latter was considered for layer-wise partitionings in [163, 202]. In this work we used the same base algorithm for all partitions to keep the focus on the role of different time scales. However, we expect that further performance enhancement may be achieved by using hybrid optimization schemes.

5.8 Conclusion

Our results illustrate the potential of multirate methods for various neural network training applications. In particular, we show that a multirate approach can be used to significantly reduce the computational cost for fine-tuning neural networks, without losing test accuracy or requiring extensive hyperparameter tuning. By introducing the use of multirate techniques for neural network training, showing their use in different training settings, and outlining various directions for future work, we hope to have built a strong foundation for further research in this area.

5.9 Supplementary Discussion

Section 5.9.1 discusses variations of our multirate training schemes. In Section 5.9.2 we prove Theorem 5.3.3 for our layer-wise partitioned multirate SGD approach and derive the corresponding (existing) result for vanilla SGD. Further experimental details are provided in Section 5.9.3 and the use of "slow biases" is discussed in Section 5.9.4. Finally, we provide further ablation studies on the role of k in Section 5.9.5.

5.9.1 Variants of our Multirate Training Algorithms

Our multirate training scheme partitions the network parameters into multiple components (Algorithm 5.1 and 5.2). This setting lends itself naturally to training the different components (or copies) using different optimization strategies.

To discuss this more concretely, recall Langevin dynamics from Eq. (5.2)

$$\begin{aligned} \mathrm{d}\theta_\alpha &= p_\alpha \ \mathrm{d}t, \\ \mathrm{d}p_\alpha &= \tilde{G}_\alpha(\theta)\mathrm{d}t - \gamma_\alpha p_\alpha \ \mathrm{d}t + \sqrt{2\gamma_\alpha \tau_\alpha} \ \mathrm{d}W_\alpha, \text{ where } \alpha = F, S, \end{aligned}$$

with neural network parameters $\theta = (\theta_F, \theta_S) \in \mathbb{R}^n$, momentum $p = (p_F, p_S) \in \mathbb{R}^n$, noisy (due to subsampling) gradient $\tilde{G}_\alpha(\theta)$ of the loss with respect to θ_α, Wiener process W, and hyperparameters $\gamma_\alpha, \tau_\alpha > 0$. Using discretized Langevin dynamics to train neural networks allows for incorporation of both momentum and additive noise, the size of which is controlled by the γ_α and τ_α hyperparameters, respectively. A straightforward variant is thus to use different values for γ_α and/or τ_α for the fast and slow components. Hyperparameter τ controls the transition between a pure optimization and sampling approach and when small can benefit neural network optimization [163, 287]. Using small values of τ for parts of the dynamics that require further exploration may thus benefit training. On the other hand, to train a component with stochastic gradient descent with momentum one can set $\tau = 0$. An example of a possible combination of Langevin dynamics with additive noise for the fast dynamics and without additive noise (corresponding to SGD) for the slow dynamics is then

$$\begin{aligned} \mathrm{d}\theta_F &= p_F \ \mathrm{d}t, \ \mathrm{d}p_F = \tilde{G}_F(\theta)\mathrm{d}t - \gamma_F p_F \ \mathrm{d}t + \sqrt{2\gamma_F \tau} \ \mathrm{d}W_F \\ \mathrm{d}\theta_S &= p_S \ \mathrm{d}t, \ \mathrm{d}p_S = \tilde{G}_S(\theta)\mathrm{d}t - \gamma_S p_S \ \mathrm{d}t. \end{aligned}$$

Equivalently, one could change the value of the momentum hyperparameter γ_α for the different partitionings. Or use different optimizers for the different components, such as Adam and SGD. Finally, it would be interesting to study the effect of using different size initializations for the different components, which essentially starts off the different components on different scales. Of course, any of these suggestions require extra tuning of the algorithm, which is why we focused on SGD with the same momenta values and initializations for all components in this work. We expect however that using hybrid optimization schemes may lead to even further performance enhancement, which we aim to explore in future work.

As a simple extension of Algorithm 5.1 we provide the case with weight decay in Algorithm 5.3, where we have used ω to denote the amount of weight decay. Our implementation of weight decay is the same as used in PyTorch for SGD [219].

Algorithm 5.3 Multirate SGD with linear drift and weight decay

$p_S := \mu p_S + \nabla_{\theta_S} \mathcal{L}(\theta_S, \theta_F) + \omega_S \theta_S$
for $i = 1, 2, ..., k$ **do**
  $p_F := \mu p_F + \nabla_{\theta_F} \mathcal{L}(\theta_S, \theta_F) + \omega_F \theta_F$
  $\theta_F := \theta_F - \frac{h}{k} p_F$
  $\theta_S := \theta_S - \frac{h}{k} p_S$
end for

5.9.2 Convergence Analysis

Recall our main assumptions.

Assumption 5.9.1. We assume function $f : \mathbb{R}^n \to \mathbb{R}$ to be L-smooth, i.e., f is continuously differentiable and its gradient is Lipschitz continuous with Lipschitz constant $L > 0$

$$\|\nabla f(\varphi) - \nabla f(\theta)\|_2 \leq L\|\varphi - \theta\|_2, \ \forall \theta, \varphi \in \mathbb{R}^n. \tag{5.7}$$

Assumption 5.9.2. We assume that the second moment of the stochastic gradient is bounded above, i.e., there exists a constant M for any sample x_i such that

$$\|\nabla f_{x_i}(\theta)\|_2^2 \leq M, \quad \forall \theta \in \mathbb{R}^n. \tag{5.8}$$

Assumption 5.9.2 guarantees the variance of the stochastic gradient to be less than M, because

$$\begin{aligned} \mathrm{Var}(\nabla f_{x_i}(\theta)) &= \mathbb{E}\|\nabla f_{x_i}(\theta) - \mathbb{E}[\nabla f_{x_i}(\theta)]\|_2^2 \\ &= \mathbb{E}\|\nabla f_{x_i}(\theta) - \nabla f(\theta)\|_2^2 \\ &= \mathbb{E}\|\nabla f_{x_i}(\theta)\|_2^2 - \|\nabla f(\theta)\|_2^2 \end{aligned} \tag{5.9}$$

where we used $\mathbb{E}[\nabla f_{x_i}(\theta)] = \nabla f(\theta)$ (unbiased gradient) for the second equality and $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

If we assume Assumption 5.9.1 holds, we obtain Lemma 5.9.3, which we will need for the proof of the main theorem.

Lemma 5.9.3. If $f : \mathbb{R}^n \to \mathbb{R}$ is L-smooth then $\forall \theta, \varphi \in \mathbb{R}^n$

$$|f(\varphi) - (f(\theta) + \nabla f(\theta)^T(\varphi - \theta))| \leq \frac{L}{2}\|\varphi - \theta\|_2^2. \tag{5.10}$$

Proof of Lemma 5.9.3. From the fundamental theorem of calculus

$$\int_0^1 \nabla f[\theta + t(\varphi - \theta)]^T(\varphi - \theta) \ \mathrm{d}t = f(\varphi) - f(\theta)$$

So using Cauchy-Schwartz and the assumption that f is L-smooth we obtain

$$\begin{aligned} |f(\varphi) - f(\theta) - \nabla f(\theta)^T(\varphi - \theta)| &= \left| \int_0^1 \left(\nabla f[\theta + t(\varphi - \theta)] - \nabla f(\theta)\right)^T (\varphi - \theta) \ \mathrm{d}t \right| \\ &\leq \int_0^1 \|\nabla f[\theta + t(\varphi - \theta)] - \nabla f(\theta)\|_2 \|\varphi - \theta\|_2 \mathrm{d}t \\ &\leq L\|\varphi - \theta\|_2^2 \int_0^1 t \ \mathrm{d}t = \frac{L}{2}\|\varphi - \theta\|_2^2. \end{aligned}$$

□

The multirate method update for base algorithm SGD is

$$\theta_\ell^{t+1} = \theta_\ell^t - h\nabla f_{\ell,x_i}(\theta^t), \tag{5.11}$$

where θ_ℓ^t are the parameters in layer ℓ at iteration t, h is the stepsize, and $\nabla f_{\ell,x_i}$ denotes the gradient of the loss of the ith training example for parameters in layer $l \in \{F, S\}$, where $\nabla f_{F,x_i}(\theta^t) = \nabla f_{F,x_i}(\theta^t)$ and with linear drift: for any $t \in [\tau, \tau + k - 1]$, where τ is divisible by k, $\nabla f_{S,x_i}(\theta^t) = \nabla f_{S,x_i}(\theta^\tau)$.

Now we want to prove Theorem 5.3.3.

Theorem 5.9.4. Assume that Assumptions 5.9.1 and 5.9.2 hold. Then

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] \leq \frac{2(f(\theta^0) - f(\theta^*))}{hT} + hLM\ell\left(\frac{1}{3}hLk^2 + 1\right), \tag{5.12}$$

where θ^* is the optimal solution to $f(\theta)$.

Proof of Theorem 5.9.4. Because f is L-smooth, from Lemma 5.9.3 it follows that

$$\begin{aligned} f(\theta^{t+1}) &\leq f(\theta^t) + \nabla f(\theta^t)^{Tr}(\theta^{t+1} - \theta^t) + \frac{L}{2}\|\theta^{t+1} - \theta^t\|_2^2 \\ &\leq f(\theta^t) - h\nabla f(\theta^t)^{Tr}\left(\sum_\ell \nabla f_{\ell,x_i}(\theta^t)\right) + \frac{h^2 L}{2}\left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^t)\right\|_2^2 \end{aligned} \tag{5.13}$$

Taking the expectation on both sides gives (because of unbiased gradient $\mathbb{E}[\nabla f_{x_i}(\theta)] = \nabla f(\theta)$ and Assumption 5.9.2)

$$\mathbb{E}[f(\theta^{t+1}) - f(\theta^t)] \leq -h\nabla f(\theta^t)^{Tr}\left(\sum_\ell \nabla f_\ell(\theta^t)\right) + \frac{h^2 LM\ell}{2} \tag{5.14}$$

for number of layers ℓ. So in T iterations we have θ^T such that (using a telescoping sum)

$$\begin{aligned} f(\theta^*) - f(\theta^0) &\leq \mathbb{E}[f(\theta^T)] - f(\theta^0) \\ &\leq -h\underbrace{\sum_{t=0}^{T-1} \nabla f(\theta^t)^{Tr}\left(\sum_\ell \nabla f_\ell(\theta^t)\right)}_{\mathcal{A}} + \frac{h^2 LM\ell}{2}T. \end{aligned} \tag{5.15}$$

For term $\mathcal{A}$ we get

$$\mathcal{A} = \sum_{t=0}^{T-1} a_t = \sum_{t=0}^{k-1} a_t + \sum_{t=k}^{2k-1} a_t + \cdots + \sum_{t=\tau}^{\tau+k-1} a_t + \cdots + \sum_{t=T-k}^{T-1} a_t, \tag{5.16}$$

where $\sum_{t=\tau}^{\tau+k-1} a_t$ is given by

$$\begin{aligned} \sum_{t=\tau}^{\tau+k-1} \nabla f(\theta^t)^{Tr}\left(\sum_\ell \nabla f_\ell(\theta^t)\right) &= \sum_{t=\tau}^{\tau+k-1} \nabla f(\theta^t)^{Tr}(\nabla f_F(\theta^t) + \nabla f_S(\theta^\tau)) \\ &= \sum_{t=\tau}^{\tau+k-1} \nabla f(\theta^t)^{Tr}(\nabla f_F(\theta^t) + \nabla f_S(\theta^\tau) - \nabla f_S(\theta^t) + \nabla f_S(\theta^t)) \\ &= \sum_{t=\tau}^{\tau+k-1} \|\nabla f(\theta^t)\|_2^2 + \underbrace{\sum_{t=\tau}^{\tau+k-1} \nabla f(\theta^t)^{Tr}(\nabla f_S(\theta^\tau) - \nabla f_S(\theta^t))}_{\mathcal{B}}. \end{aligned}$$

Because $xy \leq \frac{1}{2}\|x\|_2^2 + \frac{1}{2}\|y\|_2^2$ (combination of Cauchy-Schwarz and Young's inequality) (gives

1st inequality) and Assumption 5.9.1 (gives 2nd inequality) we get for term $\mathcal{B}$

$$\mathcal{B} \leq \frac{1}{2}\sum_{t=\tau}^{\tau+k-1} \|\nabla f(\theta^t)\|_2^2 + \frac{1}{2}\sum_{t=\tau}^{\tau+k-1} \|\nabla f_S(\theta^\tau) - \nabla f_S(\theta^t)\|_2^2 \tag{5.17}$$

$$\leq \frac{1}{2}\sum_{t=\tau}^{\tau+k-1} \|\nabla f(\theta^t)\|_2^2 + \frac{L^2}{2}\underbrace{\sum_{t=\tau+1}^{\tau+k-1} \|\theta^\tau - \theta^t\|_2^2}_{\mathcal{C}}. \tag{5.18}$$

We get for term $\mathcal{C}$ from Eq. (5.11) (gives 2nd equality), $\|a_1+\cdots+a_m\|_2^2 \leq m(\|a_1\|_2^2+\cdots+\|a_m\|_2^2)$ (gives 1st inequality), Assumption 5.9.2 (gives 2nd inequality), and $k > 1$ (final inequality)

$$\mathcal{C} = \|\theta^\tau - \theta^{\tau+1}\|_2^2 + \|\theta^\tau - \theta^{\tau+2}\|_2^2 + \cdots + \|\theta^\tau - \theta^{\tau+k-1}\|_2^2 \tag{5.19}$$

$$= h^2\left(\left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^\tau)\right\|_2^2 + \left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^\tau) + \sum_\ell \nabla f_{\ell,x_i}(\theta^{\tau+1})\right\|_2^2 \right.$$
$$\left. + \cdots + \left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^\tau) + \cdots + \sum_\ell \nabla f_{\ell,x_i}(\theta^{\tau+k-2})\right\|_2^2\right) \tag{5.20}$$

$$\leq h^2\left(\sum_{m=1}^{k-1} m\left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^\tau)\right\|_2^2 + \sum_{m=2}^{k-1} m\left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^{\tau+1})\right\|_2^2 + \right.$$
$$\left. + \cdots + (k-1)\left\|\sum_\ell \nabla f_{\ell,x_i}(\theta^{\tau+k-2})\right\|_2^2\right) \tag{5.21}$$

$$\leq h^2 M\ell\left((k-1)^2 + (k-2)^2 + \cdots + 1\right) = h^2 M\ell\sum_{m=1}^{k-1} m^2 = h^2 M\ell\left(k/6 - k^2/2 + k^3/3\right)$$

$$\leq h^2 M\ell k^3/3. \tag{5.22}$$

So overall for term $-h\mathcal{A}$ we get

$$-h\sum_{t=0}^{T-1} \nabla f(\theta^t)^{Tr}\left(\sum_\ell \nabla f_\ell(\theta^t)\right) \leq -h\sum_{t=0}^{T-1} \|\nabla f(\theta^t)\|_2^2 + h\left|\sum_\tau \mathcal{B}\right|$$
$$\leq -\frac{h}{2}\sum_{t=0}^{T-1} \|\nabla f(\theta^t)\|_2^2 + \frac{1}{6}h^3L^2M\ell k^2T. \tag{5.23}$$

Substituting this into Eq. (5.15) and again taking the expectation gives

$$f(\theta^*) - f(\theta^0) \leq \mathbb{E}[f(\theta^T)] - f(\theta^0)$$
$$\leq -\frac{h}{2}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] + \frac{1}{6}h^3L^2M\ell k^2T + \frac{h^2LM\ell}{2}T$$
$$= -\frac{h}{2}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] + \frac{1}{2}h^2LM\ell T\left(\frac{1}{3}hLk^2 + 1\right). \tag{5.24}$$

This gives Theorem 5.9.4

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] \leq \frac{2(f(\theta^0) - f(\theta^*))}{hT} + hLM\ell\left(\frac{1}{3}hLk^2 + 1\right).$$

□

For comparison, the convergence analysis for vanilla SGD with fixed stepsize h update

$$\theta^{t+1} = \theta^t - h\nabla f_{x_i}(\theta^t), \tag{5.25}$$

where ∇f_{x_i} denotes the gradient of the loss of the ith training example, gives Theorem 5.9.5.

Theorem 5.9.5. Assume that Assumptions 5.9.1 and 5.9.2 hold. Then

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] \leq \frac{2(f(\theta^0) - f(\theta^*))}{hT} + \frac{hLM}{2} \tag{5.26}$$

where θ^* is the optimal solution to $f(\theta)$.

Proof of Theorem 5.9.5. Because f is L-smooth, from Lemma 5.9.3 it follows that

$$\begin{aligned} f(\theta^{t+1}) - f(\theta^t) &\leq \nabla f(\theta^t)^{Tr}(\theta^{t+1} - \theta^t) + \frac{L}{2}\|\theta^{t+1} - \theta^t\|_2^2 \\ &\leq -h\nabla f(\theta^t)^{Tr}\nabla f_{x_i}(\theta^t) + \frac{h^2 L}{2}\|\nabla f_{x_i}(\theta^t)\|_2^2 \end{aligned}$$

Taking the expectation on both sides gives (because of Assumption 5.9.2 and unbiased gradient $\mathbb{E}[\nabla f_{x_i}(\theta)] = \nabla f(\theta)$)

$$\mathbb{E}[f(\theta^{t+1}) - f(\theta^t)] \leq -h\mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] + \frac{h^2 L}{2}M \tag{5.27}$$

So in T gradient steps we have θ^T such that

$$f(\theta^*) - f(\theta^0) \leq \mathbb{E}[f(\theta^T)] - f(\theta^0) \leq -h\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] + \frac{h^2 LMT}{2} \tag{5.28}$$

This gives

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(\theta^t)\|_2^2\right] \leq \frac{2(f(\theta^0) - f(\theta^*))}{hT} + \frac{hLM}{2} \tag{5.29}$$

□

5.9.3 Further Experimental Details

We run our experiments (unless indicated otherwise) with SGD with momentum set to 0.9. The learning rate varied per experiment and is detailed in the captions of the figures. For the transfer learning experiments (Section 5.4) it was set to $h = 0.001$ for the ResNet architectures and to $h =$ 1e-4 for the DistilBERT architecture and we did not use weight decay. In Algorithm 5.1 we set $k = 5$ and varied our partitionings of the network parameters into fast and slow parts. All our experiments were run in PyTorch using NVIDIA GPUs. We will discuss specific experiments that require further details below.

Patch-augmented CIFAR-10. The patch-augmented CIFAR-10 dataset that we used for Figure 5.1 was adapted from the paper by Li et al. [178]. We first split the 50000 CIFAR-10 training images, into 10000 patch-free images and 40000 images which contain only a patch with probability 0.2 and contain a patch mixed with CIFAR-10 data with probability 0.8. The 7×7 pixel patch is located in the center of the images. Following Li et al. [178] to generate the patch, we first sample $z \sim \mathcal{N}(0, 1.5625), a$ a random float in [0,1), and $\zeta_i \sim [-0.1, 0.1]$ for classes $i = 1, \ldots, 10$. Then for patch-only images belonging to class i we set everything to 0 and add $z \pm 1.75a\zeta_i$. To generate images containing both a patch and CIFAR-10 data we add $z \pm \zeta_i$. For the multirate training approach we partitioned a composite network system into two parts, where each subnetwork was trained on a different timescale. The weights sampled from

both parts were averaged and merged every k steps. The exact same learning rates were used as in the original paper by Li et al. [178], so $h_F = 0.004$, $h_S = 0.1$, and thus $k = h_S/h_F = 25$.

5.9.4 Slow Biases

We study the effect of putting all the biases of a neural network on the slow time scale, while keeping the weights on the fast time scale and only updating the slow parameters every k steps using Algorithm 5.1. Surprisingly, this gives big performance improvements on 4-turn spiral data (adapted from [163]) as shown in Figure 5.9. Figure 5.10 confirms that this enhanced performance is caused by our multirate technique, i.e., freezing the biases for k steps and then boosting them with a larger time-step. Simply putting the biases on a different time scale or freezing the biases for k steps and using the same time-step does not lead to the same performance improvement. In Figure 5.11 we show that this effect is caused by the input layer biases, in particular. This seems to suggest a possible connection with data normalization (and the lack thereof for the spiral dataset). In Figure 5.12 we show that for a ResNet-34 architecture on CIFAR-10 data one also obtains a small performance improvement by using slow biases for the fully connected layer, especially when no data augmentation is used.

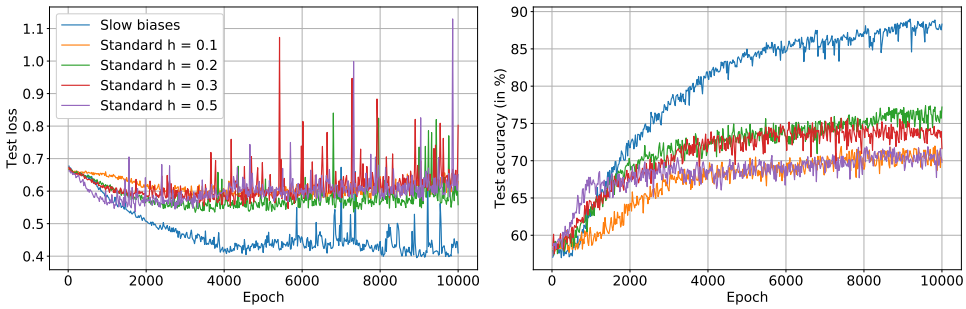

Figure 5.9: We use both standard SGD and our multirate approach as defined in Algorithm 5.1 to train a SHLP on a 4-turn spiral problem with 5% subsampling. We set the biases of the neural network to be θ_S and the weights to be θ_F and set $k = 5$, $h = 1$ in Algorithm 5.1. Results are averaged over 5 runs.

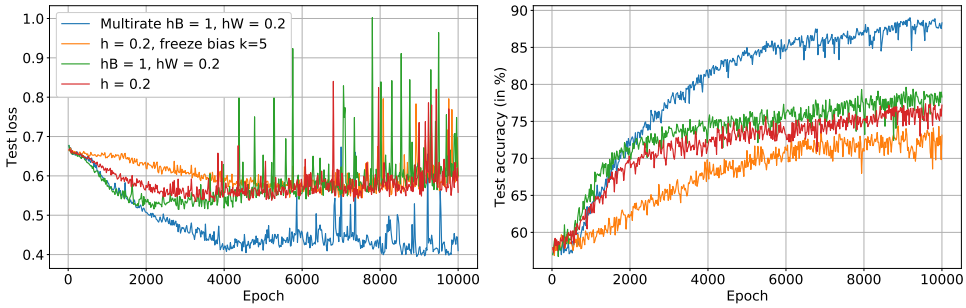

Figure 5.10: Same setting as in Figure 5.9, with a SHLP being trained on spiral data using SGD with $h = 0.2$ (red). We show that putting the biases on a different time scale (green) or freezing the biases for $k = 5$ steps and then updating them with the same stepsize as for the fast (weight) parameters (orange) both do not lead to the same performance improvement as our multirate technique (blue).

5.9.5 Further Ablation Studies

The effect of k is studied in Table 5.3 for fine-tuning a pre-trained DistilBERT on SST-2 data using Algorithm 5.1. We find that although smaller values of k can improve the generalization

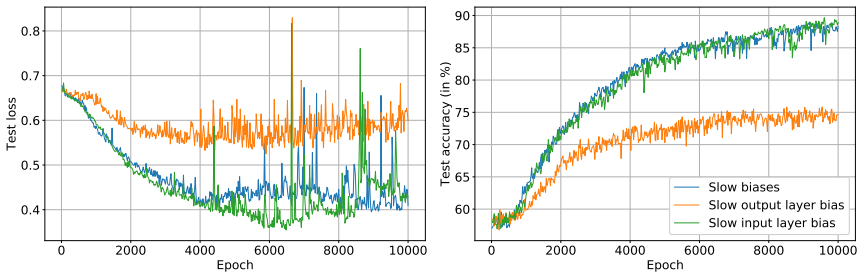

Figure 5.11: Same setting as in Figure 5.9, but here we study the effect of only putting the input biases (green) or only the output biases (orange) on the slow time scale. Clearly, using slow input biases (green) appears to be key to the enhanced generalization performance of the multirate approach (blue) in this setting.

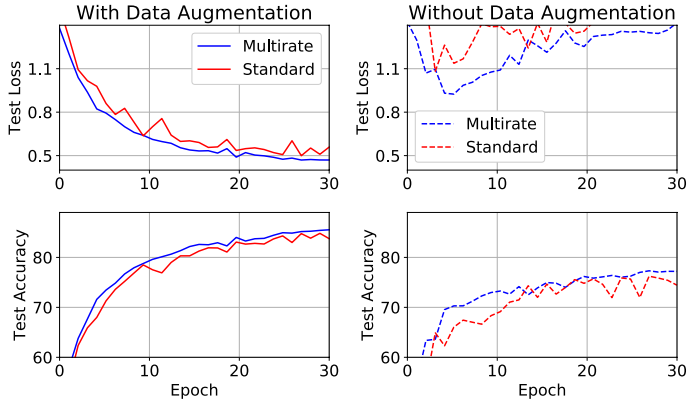

Figure 5.12: We use both standard SGD (with $h = 0.1$) and our multirate approach as defined in Algorithm 5.1 to train a ResNet-34 architecture on CIFAR-10 data. We set the biases of the fully connected layer of the neural network to be θ_S and the weights to be θ_F and set $k = 10$, $h = 1$ in Algorithm 5.1 and use batchsize = 128. Results are averaged over 10 runs.

performance, the training time gets increased. This trade-off needs to be taken into account when choosing k. Apart from this, recall that lower values of k also lower the stepsize used for the slow parameters, which can affect performance. As a rule of thumb, we found that setting $k = 5$ gives enough speed-up, without affecting the accuracy.

Table 5.3: **Effect of k.** A pre-trained DistilBERT being trained on SST-2 data using our multirate approach for different values of k (same setting as in Figure 5.6), where the fast parameters are set to be the fully connected (fc) layer + optionally the final attention block (layer 5). We set h/k = 1e-4 and $\mu = 0.9$ in Algorithm 5.1 and use a batchsize of 16. Results are presented over 10 runs.

Fast params	k	Mean test acc	Min test acc	Max test acc	Avg Time (s)
Layer fc	$k = 3$	89.26%	88.69%	90.01%	245
	$k = 5$	**89.43%**	87.92%	90.28%	198
	$k = 10$	88.53%	84.79%	89.79%	180
Layer 5 + fc	$k = 3$	88.91%	87.42%	90.06%	264
	$k = 5$	**89.70%**	89.35%	90.23%	224
	$k = 10$	88.65%	87.97%	89.73%	207

Chapter 6

Re-evaluating Metrics for Neural Network Training

In this chapter we will revisit existing deep learning metrics. In Section 6.1 we will ask the question what linear interpolation of neural network loss landscapes can tell us. Specifically, we shall study whether the shape along the linear path between initial and final parameter state of a neural network model can provide useful insights into optimization. In Section 6.2 we shall discuss potential limitations of using test accuracy as the sole metric for model evaluation for classification problems.

6.1 Revisiting Linear Interpolation as a Measure of Neural Network Loss Landscapes

The results presented in this section are currently under review and are joint work with Jonathan Frankle [277].

Abstract

Studying neural network loss landscapes provides insights into the nature of the underlying optimization problems. Unfortunately, loss landscapes are notoriously difficult to visualize in a human-comprehensible fashion. One common way to address this problem is to plot linear slices of the landscape, for example from the initial state of the network to the final state after optimization. On the basis of this analysis, prior work has drawn broader conclusions about the difficulty of the optimization problem. In this work, we put inferences of this kind to the test, systematically evaluating how linear interpolation and final performance vary when altering the data, choice of initialization, and other optimizer and architecture design choices. Further, we use linear interpolation to study the role played by individual layers and substructures of the network. We find that certain layers are more sensitive to the choice of initialization and optimizer hyperparameter settings, and we exploit these observations to design custom optimization schemes. However, our results cast doubt on the broader intuition that the presence or absence of barriers when interpolating necessarily relates to the success of optimization.

6.1.1 Introduction

Neural network loss landscapes are difficult to visualize due to their high-dimensionality and the complicated nature of the actual optimization path. This motivated the use of the loss along the linear path between the initial and final parameters of a neural network as a crude yet simple measure of the loss landscape [87]. In this work we revisit this 1D linear interpolation technique and address whether the shape of the path reflects the test accuracy of the model and can inform training.

Table 6.1: The effect of various interventions in training in (1) the shape of loss along the linear path between the initial and final state of the network and (2) the final test accuracy of the network. Different shapes of loss along the linear path are: no barrier (*NB*), barrier (*B*), and plateau (P). If the height of layer-wise barriers changes, we denote this as *H-LB* (higher) and *L-LB* (lower layer-wise barriers).

Category	Intervention	Shape of the linear path	Test accuracy
Initialization *(Sec. 6.1.4)*	Pre-train full model (Fig. 6.2A)	NB	Better
	Pre-train on random labels (Fig. 6.2B)	H-LB	Worse (if no weight decay)
	Height of the barrier initialization (Fig. 6.3)	NB	Better
	Partial (pre-)training (Fig. 6.4)	B/NB/P	Often worse
	Partial random label pre-training (Fig. 6.5)	B/NB	Often Worse
Data *(Sec. 6.1.5)*	Less data (Fig. 6.6)	NB / L-LB	Worse
	No data augmentation (Fig. 6.6)	L-LB	Worse
Optimizer *(Sec. 6.1.6)*	Less weight decay (Fig. 6.7)	NB / B	Worse
	More weight decay (Fig. 6.7)	P	Worse
	Fixed learning rate $h = 0.01$ (Fig. 6.8)	P	Better
	Fixed learning rate $h \neq 0.01$ (Fig. 6.8)	NB	Worse
	Smaller initial h for conv blocks with barriers (Fig. 6.9)	NB	Worse
Model *(Sec. 6.1.7)*	Depth (Fig. 6.10)	H-LB	Better
	No batch normalization (Section 6.1.10.6)	NB	Worse
	MLPs: overparameterize (Section 6.1.10.6)	NB	Often better

Linear Interpolation from Beginning to End. Goodfellow et al. [87] observed that for fully-connected and convolutional networks with maxout [85] trained on MNIST data the loss decays monotonically along the linear path between their initial and final state. Although the loss landscape for overparameterized neural networks is not glassy [12, 80], this observed monotonic decay behavior is still surprising. The absence of obstacles along the linear path led Goodfellow et al. [87] to conclude that "these tasks are relatively easy to optimize." This result has been cited widely as an indication of the ease of training (e.g., Li et al. [171], McCandlish et al. [196], Fort et al. [70]) and the linear interpolation technique itself was used in many papers [112, 136, 122, 125, 94]. In this work we address empirically how meaningful the use of this linear path actually is. The exact definition of training tasks being "easy to optimize" is an open question and optimization choices directly influence which linear path we observe. It is also arguable whether we actually want tasks to be easier to optimize; lowering the amount of training data or reducing regularization simplifies training but lowers test accuracy.

The work by Goodfellow et al. [87] was recently revisited and extended by Frankle et al. [72] and Lucas et al. [185] for a range of modern neural network models, such as ResNet and VGG architectures, on image datasets. Frankle et al. [72] observed for default parameter settings that the loss often remained at the level of random chance until close to the optimum for these models, different than the behavior observed by Goodfellow et al. [87]. In addition to concurrently confirming this result, Lucas et al. [185] found that the monotonic decay property observed by Goodfellow et al. [87] was often maintained when BatchNorm was removed and non-adaptive optimizers were used. On this basis, they hypothesized that "large distances moved in weight space encourage non-monotonic interpolation."

In our work, we interrogate conjectures stated in prior work that the shape of loss along the linear path relates to the "success" of optimization (which we measure in terms of test accuracy) or other aspects of optimization (e.g., distance travelled). We systematically study the influence of various optimizer and architecture design choices on the shape of the linear path and the test accuracy of the final model, and examine interpolation for individual layers in addition to the entire network. An overview of our results is shown in Table 6.1. Our main finding is that there are situations that both support and violate the aforementioned intuitions on the shape of the linear path. As such, we recommend caution when using this analysis to infer other information about the nature of the optimization problem.

Our findings:

- Pre-training on ImageNet consistently removes the presence of barriers for ResNet architectures trained on CIFAR-10 data, whereas adversarial initialization on random labels increases barriers. The former typically increases the final test accuracy, whereas (in the

absence of weight decay) the adversarial initialization lowers it.

- Layers have different levels of sensitivity to the choice of initialization. We introduce the concept of partial pre-training, where we set some layers to a trained (on CIFAR-10) or pre-trained (on ImageNet) state, while using random initialization for others. We use this setting as initialization and train on CIFAR-10. We find that partial pre-training generally leads to worsened test accuracy and (uncorrelated) affects the shape of the linear path.
- The shape of the linear path of different layers can actively inform optimization procedures by using different optimizer hyperparameter choices for layers that do not exhibit barriers.
- The amount of weight decay used during training directly influences both the shape of the linear interpolation path and the final test accuracy, but there is no correlation between them.
- The distance between the initial and final parameter state is not a reliable indicator of non-monotonic behaviour along the linear path.
- The shape of the linear path from initial to final parameter state is not a reliable indicator of test accuracy.

6.1.2 Related Work

Interpolation on the Loss Landscape. Interpolation between networks in various forms has been a valuable tool for gaining insight into the structure of the optimization landscape. Frankle et al. [72] found that although the loss did not increase from initial to final state, barriers did appear along the linear path from later iterations to the final state. Other work focused on interpolations between different optima: Draxler et al. [58] and Garipov et al. [76] showed that there exist non-linear paths with (nearly) constant low-loss that connect a pair of minima trained from different random initializations. Fort and Jastrzębski [69] generalized this to show that there exist m low-loss connectors between $(m+1)$-tuples of optima, again using non-linear paths. Mannelli et al. [192] proposed to use the string method [64, 65] to probe the loss landscape and construct minimum energy paths connecting (for their set-up) a teacher and student model. Neyshabur et al. [213] used linear interpolation to study transfer learning. They did not observe barriers along the linear path between the final states of two ResNet-50 models both trained from pre-trained, while barriers did occur between two models trained from scratch (even when trained with the same random initialization). They interpreted this as "pre-trained weights guide the optimization to a flat basin of the loss landscape." We comment on the role of pre-training in Section 6.1.4.

Loss Landscape Visualization. To improve upon linear interpolation, 2D or 3D visualizations of the loss landscape were made by Goodfellow et al. [87], Im et al. [120], Li et al. [173], Hao et al. [94]. The reduction of the high-dimensional loss landscape into 2D or 3D slices requires a choice of directions. The chosen directions strongly affect the resulting observed behaviour. Although the linear interpolation method also sacrifices information by taking a 1D slice, it does so with perfect fidelity by considering the path between initial and final state. Linear interpolation is seen as "a simple and lightweight method to probe neural network loss landscapes" [185] and therefore remains frequently used, e.g. by Keskar et al. [136], Jastrzębski et al. [125], Lucas et al. [184], Neyshabur et al. [213].

Role of Layers. Zhang et al. [307] studied the role of different layers by training a neural network and then re-setting specific layers to their initial value or a random value, while keeping the other layers fixed at their final state. They observed that certain critical layers are much more sensitive to this perturbation. Chatterji et al. [39] extended this analysis by studying linear interpolation for specific modules. They relate their concept of "module criticality" with high robustness to noise and valley width. Neyshabur et al. [213] studied both the direct and optimization path from initial to final state for modules of pre-trained models and found that later layers have tighter valleys.

Our Novel Contributions. Previous work studied the shape of the linear path [72, 185], but did not interrogate the connection with the success of optimization. Dating back to Goodfellow et al. [87] intimating that an absence of barriers along the linear path means that "tasks are relatively easy to optimize", numerous works have implicitly relied on the presence of such a connection to make other claims [196, 173, 70, 94, 184, 213]. In our work, we study exhaustively if such a connection exists by systematically altering initialization, data, and other optimizer and architecture design choices. We also study the hypothesis by Lucas et al. [185] that "large distances moved in weight space encourage non-monotonic interpolation". Further, we introduce several novel modes of analysis, such as initializing from the height of the barrier and using linear interpolation to study the role played by individual layers and substructures of the network. In particular, we illustrate the sensitivity of different layers to the choice of initialization and demonstrate the adversarial effect of partial pre-training (Section 6.1.4).

6.1.3 Linear Interpolation from Start to Finish

We use the 1D linear interpolation technique [87] to study the linear path between the initial and final state of the model. We introduce this technique and how we study the linear path layer-wise in Section 6.1.3.1. In Section 6.1.3.2 we discuss which different shapes of this linear path we observe and compare our results with the literature [87, 72, 185].

6.1.3.1 Methodology

Training. We focus on a ResNet-18 [97] architecture with batch normalization trained for 100 epochs on CIFAR-10 data [146] using SGD with momentum (0.9) and weight decay (5e-4) using PyTorch [219]. We use initial learning rate $h = 0.1$ that drops by 10x at epochs 33 and 66. For pre-trained settings, we use initial learning rate $h = 0.001$ that drops by 10x after 30 epochs. Results are averaged over 10 runs unless indicated otherwise. By modifying different aspects of this training problem, we will study the role of the initialization, data, optimizer, and model throughout this work. We also consider other architectures, including other ResNets, VGG architectures, and multi-layer perceptrons.

Linear Interpolation Measure for the Full Model. Consider a L-layer neural network with parameters $\theta = (\theta^{(0)}, \ldots, \theta^{(L)})$. We use θ_i to refer to the initial state of these parameters and θ_f to refer to the parameters after training using algorithm $\mathcal{T}_t(\theta_i, \mathcal{D})$ on dataset $\mathcal{D}$ for t steps. Following Goodfellow et al. [87], a linear interpolation path between θ_i and θ_f is created as follows

$$\theta_\alpha = (1-\alpha)\theta_i + \alpha\theta_f \quad \text{for} \quad \alpha \in [0,1]. \tag{6.1}$$

To examine the loss landscape along this path, we plot the loss for a discrete set of values of α from 0 (initial state) to 1 (final state). One can extend this technique to evaluate the linear path between the state of the model at different steps in training, where θ_i and θ_f are replaced by the model states at the considered steps. One can also study the path between θ_f and a different random initialization θ_i', which is separately sampled from the initialization distribution.

Layer-wise Linear Interpolation. We also study linear interpolation in a layer-wise fashion: we vary a single layer (or convolutional block) from initial to final state while keeping all other parameters fixed at their final state. Concretely, for an L-layer network where we vary layer ℓ

$$\theta_\alpha^{(\ell)} = (1-\alpha)\theta_0^{(\ell)} + \alpha\theta_f^{(\ell)}, \quad \theta_\alpha^{(k)} = \theta_f^{(k)},\ k \neq \ell. \tag{6.2}$$

This technique was first proposed by Chatterji et al. [39], who found that certain ResNet layers were more robust to parameter perturbations. In this work, we use the layer-wise linear path to study the role played by substructures of the network. We will also use the shape of the layer-wise linear path to actively inform custom optimization schemes that use different optimizer hyperparameters for different layers. We will vary convolutional blocks as a whole.

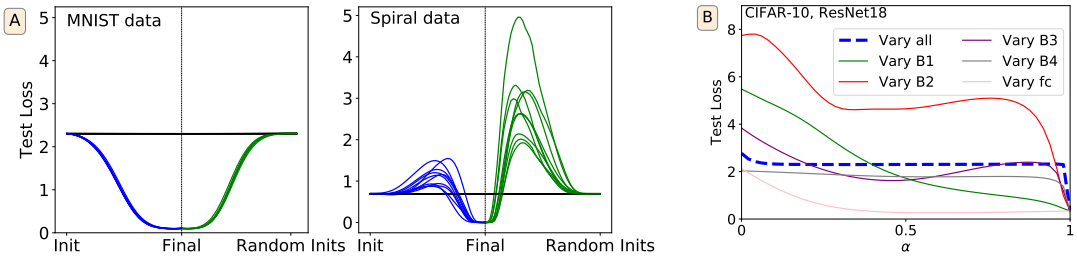

Figure 6.1: *(A)* Left: MNIST data. Right: spiral data. Test loss over 10 runs for a two hidden layer MLP along the linear path between: θ_i and θ_f (blue), θ_f and θ_i' (green), θ_i and θ_i' (black). *(B)* Test loss when interpolating for convolutional blocks (denoted as B), fully connected layer (fc), or the entire network (blue dashed line) for a ResNet-18 architecture on CIFAR-10 data, averaged over 10 runs.

6.1.3.2 Appearance of Barriers in the Loss Landscape

Defining Barriers. We consider the linear path to contain a *barrier* if it exhibits a monotonic increase in loss. Goodfellow et al. [87] observed that, for fully-connected networks on MNIST data, loss decays monotonically along the linear path between initial and trained model, i.e. there are no barriers. Figure 6.1A (left) reproduces this behaviour for an MLP with two hidden layers trained on MNIST (blue) (full details in Section 6.1.10.1); moreover, the same monotonic decrease occurs when interpolating between these final weights and *any* random initialization (green). However, this barrier-free linear interpolation is not a universal phenomenon. As a counterexample, when using the exact same architecture and optimizer but a different dataset (the spiral dataset), barriers do appear along the linear path (Figure 6.1A, right), in fact rising above the level of loss at initialization; barriers rise even higher when interpolating to other initializations.

Barriers in Modern Neural Network Architectures. Frankle et al. [72] and Lucas et al. [185] updated the results of Goodfellow et al. [87] by studying linear interpolation in modern vision settings. Both observed that, in many cases, "loss plateaus and error remains at the level of random chance ... until near the optimum" [72]; that is, loss remains flat, neither monotonically decreasing or encountering barriers. Our results for a ResNet-18 architecture on CIFAR-10 data agree with these findings when interpolating for the entire network as the dashed blue line in Figure 6.1B illustrates.

Block-wise Interpolation. Individual convolutional blocks behave differently from the full network. As the solid lines in Figure 6.1B show, different blocks take on a variety of behaviors including barriers and monotonic decay. This suggests that it may be valuable to study the connection between linear interpolation and other properties of the network at the finer granularity of individual structural components rather than at the coarse granularity of the entire network.

6.1.4 The Role of Initialization

The choice of initialization affects the path that the model follows and the optimum it finds. It is therefore natural to believe that it also affects the nature of the loss when linearly interpolating from start to finish, a relationship we study in this section. We observe an intuitive relationship between initialization and barriers: actions that make the task easier (e.g., pre-training the model or initializing at the barrier) remove barriers, while those that make the task harder (e.g., initializing adversarially) increase the size of barriers. Pre-training only certain layers can both create barriers and worsen test accuracy, although not in a correlated fashion.

Pre-training. Training a model from a pre-trained state (on ImageNet) causes the loss along the linear path to monotonically decay and improves test accuracy (Figure 6.2A). This is

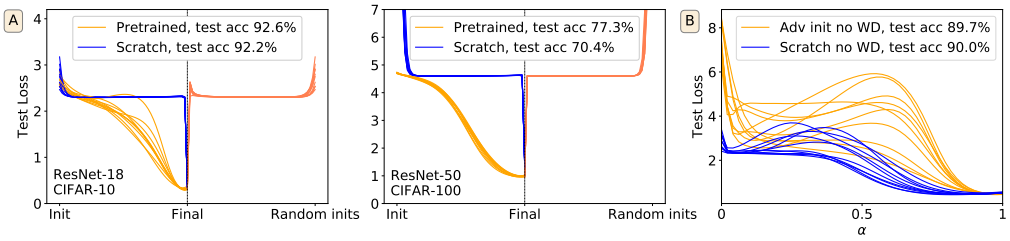

Figure 6.2: *(A)* Test loss between θ_i and θ_f of a ResNet-18 for CIFAR-10 (left) and ResNet-50 for CIFAR-100 (middle), when trained from scratch (blue) or pre-trained on ImageNet (orange) over 10 runs. Orange-red lines represent the linear path between θ_f (trained from pre-trained) to θ_i'. *(B)* Test loss between θ_i and θ_f of a ResNet-18 trained with SGD without weight decay (WD) on CIFAR-10 either from scratch (blue) or from a network pre-trained on 100% random labels (orange).

distinct from the behavior when interpolating to a new random initialization, which exhibits a plateau. This result aligns with the intuition that pre-training simplifies optimization [94, 213].

Adversarial Initialization. We perform adversarial initialization by training on 100% random labels until 100% training accuracy is reached and use this state as initialization for training. In the absence of weight decay, pre-training a ResNet-18 on random labels lowers the final test accuracy [180] and increases the barrier height between initial and final state (Figure 6.2B). This directly opposes the effect of pre-training (Figure 6.2A), suggesting adversarial initialization complicates optimization.

Height of the Barrier Initialization. The linear path of a ResNet-18 trained without weight decay on CIFAR-10 exhibits barriers for some random seeds. For the runs that exhibit barriers, we save the model state at the height of the barrier and use this as initialization for training to study if there exists a barrier between this state and the new optimum. Although it was initialized at a higher loss than occurs at random initialization, the resulting network obtains a higher test accuracy than the network trained from scratch and its linear path is barrier free (Figure 6.3).[1] This mirrors the pre-training results (Figure 6.2A), suggesting initialization at the height of the barrier aids optimization.

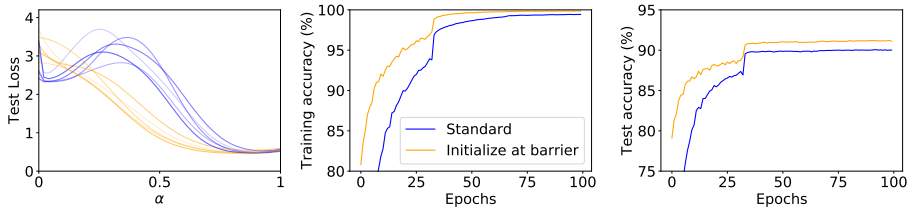

Figure 6.3: ResNet-18 on CIFAR-10 without weight decay. Left: test loss between θ_i and θ_f, when trained either from scratch (blue) or from a initialization corresponding to the height of the barrier along each blue line (orange). Initialization at the height of the barrier removes the presence of a barrier (left), speeds up training (middle), and improves the test accuracy (right), averaged over 5 runs.

Partial Pre-training. Zhang et al. [307] measured the change in test accuracy when re-setting specific layers of a trained neural network to their initial state and found large differences across layers. To extend this work, one can study the loss from the initial state of a specific layer to its final state, while keeping all other layers fixed at their final state [39]. We found that the shape of this path greatly varies per convolutional block of a ResNet-18 (Figure 6.1B). But what these studies do not address is the effect of different layer-wise (or block-wise) initialization

[1] Using other states along the linear path, which are far enough from the original initialization, as initialization often delivers similar improvements on the final test accuracy (see Section 6.1.10.3).

Method	Test acc (%)
T-All but RI-1	91.79 ±0.23
T-All but RI-2	91.83 ±0.21
T-All but RI-3	92.35 ±0.20
T-All but RI-4	90.97 ±0.31
P-All but RI-1	89.97 ±0.13
P-All but RI-2	89.91 ±0.21
P-All but RI-3	91.78 ±0.22
P-All but RI-4	92.78 ±0.22

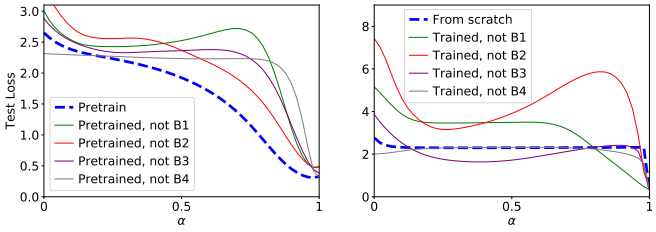

Figure 6.4: We set a ResNet-18 to a pre-trained *(P)* (middle) or trained *(T)* (right) state, but re-set a specific convolutional block *(B)* to a random initialization *(RI)*. Then we train as usual on CIFAR-10. This often affects the test accuracy of the final model (left). Sometimes barriers appear along the linear path for the full net between θ_i and θ_f (middle/right), averaged over 10 runs.

on training itself. We thus introduce the concept of *"partial (pre-)training"*: we first train on CIFAR-10 (or pre-train on ImageNet) and then re-set a specific convolutional block to its initial (random) state, while keeping the other parameters at their (pre-)trained state. We then use this state of the model as initialization and train as usual on CIFAR-10 data.

We find that while pre-training a ResNet-18 leads to higher test accuracy (Figure 6.2A), partial pre-training often leads to a lower accuracy of the final trained network (Figure 6.4, left) compared to training the net from scratch. Further, while pre-training the full net leads to monotonic decay along the linear path between initial and final state (Figure 6.2A), partial pre-training often generates barriers (Figure 6.4). The sensitivity of the network to partial pre-training varies per convolutional block and also between the trained and pre-trained setting (e.g., when re-setting convolutional block 4 (RI-4), using a trained state for the rest of the net (T-All) strongly affects test accuracy, but using a pre-trained state (P-All) does not). We conclude that the choice of initialization for different convolutional blocks strongly affects test accuracy after training and changes the nature of optimization. It is remarkable that using a random initialization for the full net leads to higher test accuracy than using a partially pre-trained or partially trained net as initialization.

Block-wise Adversarial Initialization. To further explore the effect of using different initializations for individual convolutional blocks on the linear path and test accuracy, we set one convolutional block to a random label adversarial initialization while using a standard random initialization for the rest of the net. We then re-train using this initialization. We find that while adversarial initialization for the first convolutional blocks lowers the test accuracy, using adversarial initialization for convolutional block 4 slightly increases the final test accuracy compared to training from scratch (Figure 6.5). This is also reflected by the shape of the linear path: when using an adversarial initialization for convolutional block 4 the linear path exhibits monotonic decay while others settings exhibit barriers.[2]

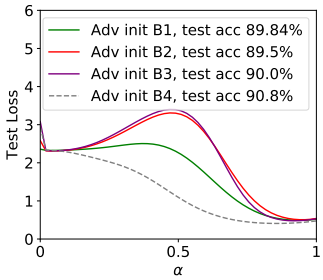

Figure 6.5: ResNet-18 on CIFAR-10 without weight decay. Test loss between θ_i and θ_f when one conv block (B) was adversarially initialized using random labels, averaged over 10 runs.

[2]Similar results are obtained for different levels of weight decay (see Section 6.1.10.3).

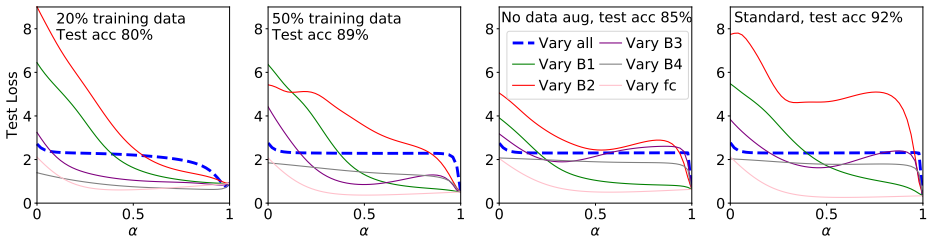

Figure 6.6: Test loss between θ_i and θ_f for ResNet-18 on CIFAR-10 when varying the amount of training data and whether data augmentation (horizontal flips, random crops) is used. Averaged over 10 runs.

Summary. Whereas the use of pre-training removes the presence of barriers and increases test accuracy, the use of adversarial initialization increases barriers and lowers test accuracy. Initializing neural networks at the height of the barrier along their initial to converged state leads to monotonic decay along the linear path, speeds up training, and improves test accuracy. Further, we find that certain convolutional blocks are more sensitive to partial pre-training than others, but the change in test accuracy is not correlated with the shape of the linear path. Whereas pre-training of the full net increases test accuracy and leads to monotonic decay, partial pre-training typically worsens the test accuracy of the resulting net and can generate barriers.

6.1.5 The Role of the Dataset

As illustrated in Figure 6.1A, where we used the same architecture, initialization, and optimizer to train on two different datasets, the data has a direct influence on the behaviour when linearly interpolating. In this section, we further investigate the effect of data on interpolation.

Number of Examples. When only a subset of the training dataset is used throughout training, we expect optimization to be easier and faster, yet the test accuracy of the final model is typically lowered. We find that using a smaller amount of CIFAR-10 training data induces monotonic decay when interpolating and lowers test accuracy (Figure 6.6, left).

Role of Data Augmentation. Throughout this work we use a horizontal flip and random crop as data augmentation for CIFAR-10. Removing data augmentation lowers the layer-wise barriers along the linear path (Figure 6.6, center right). In Section 6.1.10.4, we show that increasing data augmentation increases the height of layer-wise barriers, while removing data augmentation when using only a subset of the training data further reduces the presence of layer-wise barriers.

Summary. The shape of the layer-wise linear path is affected by changes to the data, but the shape of the full model linear path is less representative (blue dashed lines, Figure 6.6). Reducing the complexity of the task (e.g., reducing the amount of data or removing augmentation) lowers or removes layer-wise barriers. The setting with the highest layer-wise barriers reaches the highest test accuracy, and the setting without barriers reaches the lowest test accuracy.

6.1.6 The Role of the Optimizer

Optimizer hyperparameter settings, such as the learning rate and weight decay values, strongly affect which optimum is found and the test accuracy. Further, Lucas et al. [185] found that training modern vision networks with Adam (as opposed to the more typical SGD with momentum) more frequently leads to non-monotonic decay of the loss when linearly interpolating. Adam also increases the distance that the network travels from initialization, leading Lucas et al. [185] to posit "large distances moved in weight space encourage non-monotonic interpolation". Inspired by this, we study how using different levels of weight decay or different fixed learning

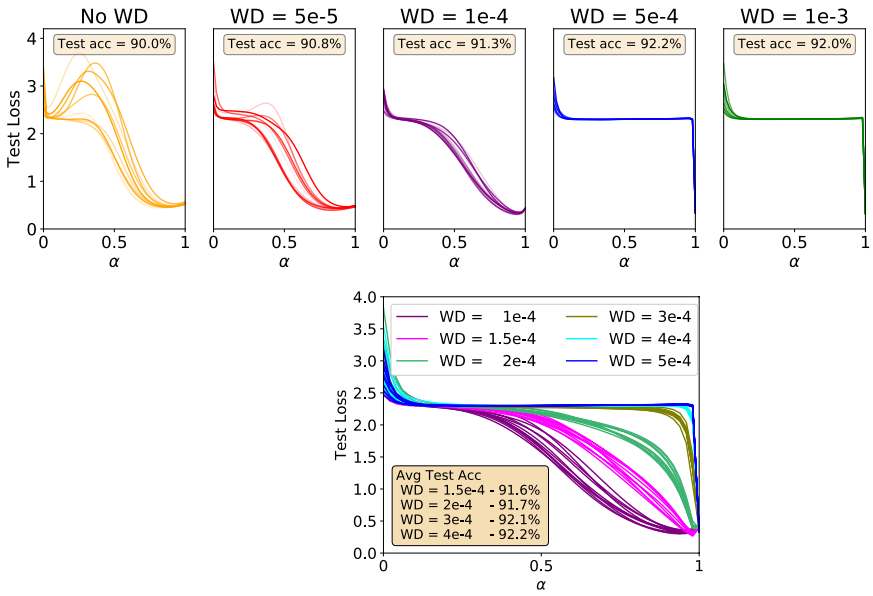

Figure 6.7: Effect of the amount of weight decay (WD) used for training ResNet-18 on CIFAR-10 on the linear path and test accuracy. The highest test accuracy is reached using WD =5e-4 (blue).

rates —both of which affect the distance travelled and final accuracy— affect the shape of loss when linearly interpolating. We find that the hypothesis of Lucas et al. [185] does not hold in general and that the level of weight decay directly controls the behaviour along the linear path.

The Effect of Weight Decay on Linear Interpolation. In Figure 6.7, we vary the amount of weight decay (WD) that is used to train ResNet-18 on CIFAR-10 data. When using little or no weight decay, the behaviour of the loss varies over different runs; it sometimes exhibits barriers and sometimes monotonic decay. When using more weight decay, the behaviour changes. At 1e-4 (purple), the loss consistently monotonically decays. The highest test accuracy is reached when training with 5e-4 (blue), for which loss consists of a plateau with a sudden drop close to the final state. When increasing beyond that, test accuracy decreases, but the loss still exhibits a plateau. The shape of the linear path is thus not a reliable indicator of test accuracy of the final model.

How do Barriers Connect to the Distance Travelled? Non-monotonic behaviour occurs when training a model both with zero and large weight decay values (Figure 6.7), while we would expect models without weight decay to travel farther. To further study this relationship, we trained a ResNet-18 at different fixed learning rates h (Figure 6.8). The linear paths for models trained using higher and lower learning rates ($h = 0.1$ and 0.001) exhibit monotonic decay, while training with an intermediate learning rate ($h = 0.01$) does not. This contradicts the hypothesis of Lucas et al. [185], under which we would expect the model with the highest learning rate (which travels the furthest) to have a barrier, not a model with a lower learning rate.[3] In addition, these results also cast doubt on a connection between monotonically decreasing loss and better accuracy; the middle learning rate (the one that induces a barrier) reaches higher test accuracy than either of the other learning rates (which do not).

Custom optimization schemes. We have observed that linear interpolation behaviour varies by layer (Figure 6.1B) and that layers/individual conv blocks have different levels of sensitivity to the choice of initialization (Figure 6.4 and Figure 6.5). This raises the question of how the use of different optimizer hyperparameter choices for different layers affects the shape of the linear path and the final test accuracy of the model. Figure 6.9 shows the effect

[3]The measure of distance is detailed in Section 6.1.10.5.

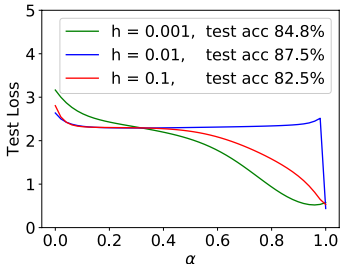

Figure 6.8: ResNet-18 trained on CIFAR-10 with fixed learning rate h. Averaged over 10 runs.

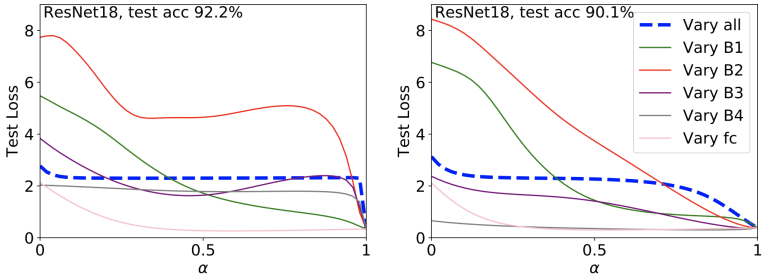

Figure 6.9: Block-wise linear interpolation for ResNet-18 trained on CIFAR-10 using different initial learning rates h_0. Left: $h_0 = 0.1$ for all, right: $h_0 = 0.01$ for layers/conv. blocks that exhibit barriers and $h_0 = 0.1$ for the rest.

of training layers that exhibit barriers with a different learning rate than those that do not. Lowering the initial learning rate used for layers that exhibit barriers removes those barriers, but also substantially lowers the test accuracy of the final model.[4].

This raises the question whether we can optimize different layers differently depending on the absence or presence of barriers. Concretely, Table 6.2 shows the result of training layers/conv blocks that exhibit barriers with a different learning rate or amount of weight decay than those that do not. Lowering the initial learning rate used for layers without barriers or eliminating weight decay has a smaller (ResNet-18) or no (VGG-11) effect on the test accuracy of the trained model, whereas doing so for layers with barriers or for all layers lowers accuracy substantially.

Table 6.2: We train layers/conv blocks that exhibit barriers (B) with a different learning rate or amount of weight decay (WD) than those that do not (NB). Test accuracy is maintained much better when training only the layers with barriers at the higher learning rate or with weight decay. Results are averaged over 10 runs.

	Test accuracy	
Initial learning rate h_0	ResNet-18	VGG-11
0.1 (all)	92.2 ±0.2%	91.9 ±0.2%
0.1 (B), 0.01 (NB)	91.8 ±0.2%	92.0 ±0.1%
0.01 (B), 0.1 (NB)	90.1 ±0.2%	90.4 ±0.1%
0.01 (all)	90.3 ±0.3%	90.9 ±0.1%
WD (B), No WD (NB)	91.5 ±0.2%	91.8 ±0.1%
No WD (B), WD (NB)	90.1 ±0.3%	90.2 ±0.1%
No WD (all)	90.1 ±0.4%	90.4 ±0.3%

Summary. The distance between initial and final state is not a reliable indicator of non-monotonic behaviour. The shape of the linear path is directly influenced by the amount of

[4]The same holds for a VGG-11 architecture, see Figure 6.16 in Section 6.1.10.5.

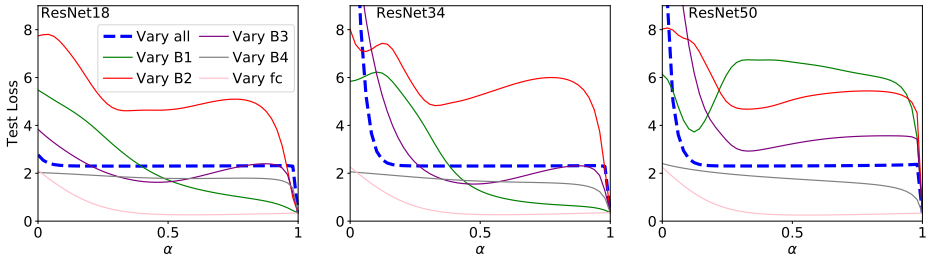

Figure 6.10: Test loss when interpolating for convolutional blocks (denoted as B) and the entire network (blue dashed line) for different ResNet architectures trained from scratch on CIFAR-10 data. Results are averaged over 10 runs.

weight decay and is not indicative of test accuracy. However, if the layer-wise linear path exhibits monotonic decay, we can train these layers with a smaller initial learning rate or no weight decay without strongly affecting test accuracy.

6.1.7 The Role of the Model

Throughout this work we focused on a ResNet-18 architecture with batch normalization. We now discuss the role of the model architecture on the shape of the linear path.

Architecture Depth. Generally, the use of deeper ResNet architectures on CIFAR-10 data generates or increases the height of convolutional block-wise barriers (Figure 6.10).

Role of Batch Normalization. Similar to Lucas et al. [185], we find that the use of batch normalization (BN) leads to non-monotonic behaviour along the linear path for ResNet and VGG architectures (see Section 6.1.10.6). However, we find that for various datasets, MLPs do exhibit barriers in the absence of BN (see e.g., Figure 6.1A). We also study the linear interpolation parameter group-wise and show that the running mean and variance BN parameters play an important role in the shape of the linear interpolation path observed for the full convolutional block (see Section 6.1.10.6).

Summary of Other Architectures. In Section 6.1.10.6, we show that the behaviour for VGG-11 is similar to that of a ResNet-18. For MLPs, we find that increasing the number of nodes in layers with barriers removes the overall presence of barriers. This effect does not transfer to ResNet architectures. For MLPs, early layers contain barriers, whereas later layers do not exhibit barriers. Similarly, Zhang et al. [307] found that the first layer is most sensitive to re-initialization.

Summary. Batch normalization encourages non-monotonic behaviour. Meanwhile, in Figure 6.9 and Table 6.2 we showed that using a large initial learning rate for layers/conv blocks that contain barriers is necessary to achieve good test accuracy. These effects together corroborate the observation by Jastrzębski et al. [126] that using a large learning rate is necessary for networks with batch normalization. We also find that the behaviour for MLPs is distinct from ResNet or VGG architectures.

6.1.8 Discussion and Future Work

Throughout this work, we explored the relationship between the shape of the linear path between the initial and final model states and the outcome of optimization. We also studied the linear path in a layer-wise fashion to illustrate that individual convolutional blocks have different sensitivities to initialization and optimizer hyperparameter settings.

What Influences the Shape of Loss when Linearly Interpolating? This one-dimensional slice of the landscape is heavily influenced by both initialization and optimiza-

tion choices. We found that the data also directly influences the linear path (Figure 6.1A and Section 6.1.5). In addition, the natural language processing literature suggests that attention-based models need adaptive optimizers (e.g. Adam) to train properly [53, 181, 308], unlike convolutional models for vision data, which can be trained well with SGD with momentum [265, 97, 303]. While we focused on the latter, we think that revisiting this study for attention-based models is an interesting direction for future work.

Does the Shape of Loss when Interpolating Provide Insight into other Aspects of Optimization? Linear interpolation is a one-dimensional view of the high-dimensional optimization landscape. The network follows a different, nonlinear path from initialization to its final weights. As such, linear interpolation inherently sacrifices information about the optimization problem; prior papers and this work consider whether the information it gleans still provides useful insights into optimization.

Despite the intuition provided by Goodfellow et al. [87] that the absence or presence of obstacles reflects difficulty of optimization, we found many cases where higher layer-wise barriers or the creation of barriers along the linear path of the full model appear alongside increased test accuracy. This evidence implies that either a more difficult optimization trajectory can lead to improved generalization, or Goodfellow et al.'s intuition about the relationship between interpolation and optimization difficulty does not apply to the networks we studied. Moreover, we found cases where monotonically decreasing loss was accompanied by higher test accuracy (e.g., pre-training). We also found others where the linear path exhibited both barriers and monotonic decay over different runs (e.g., weight decay) and where the presence or absence of barriers was not correlated with increased or decreased test accuracy. In short, for the settings we considered, the shape of the linear path between initial and final state was not a reliable indicator of test accuracy of the trained model. Our results show that caution is needed when using linear interpolation to make broader claims on the shape of the landscape and success of optimization.

Layer-wise Interpolation. Interpolating layer-wise makes it possible to study changes in behaviour that are difficult or impossible to discern when interpolating using the full network, for example changes in the data (Figure 6.6) or model (Figure 6.10). Further, independent of the shape of the linear path, we found that partial pre-training of convolutional blocks can counterintuitively lower test accuracy of the final model; this result implicates the increasingly popular direction of training individual layers in different ways [299, 163, 202, 301] and raises questions about the basis upon which we should make such decisions.

6.1.9 Conclusion

We conclude that the shape of the linear path from initial to final state is *not* a reliable indicator of test accuracy. Although focusing on one line of analysis, e.g. on the role of initialization, seems to suggest that there does exist a connection, the full picture (Table 6.1, Fig. A6.11) illustrates that this is misleading. We believe publishing this negative result is important due to the widespread use of the linear interpolation method. Further, we introduce a new line of inquiry by studying the role played by individual layers and substructures of the network. We find that certain layers require larger initial learning rates to maintain the same test accuracy. We also show the surprising adversarial effect of partial pre-training. Further exploration of the layer-wise sensitivity to choice of initialization offers an exciting direction for future work.

Acknowledgements

Tiffany Vlaar is supported by The Maxwell Institute Graduate School in Analysis and its Applications, a Centre for Doctoral Training funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016508/01), the Scottish Funding Council, Heriot-Watt University and the University of Edinburgh.

6.1.10 Supplementary Discussion

We provide further experimental details in Section 6.1.10.1. An overview figure that summarizes our findings can be found in Section 6.1.10.2. Further studies on the role of initialization, data, optimization, and model are provided in Section 6.1.10.3, Section 6.1.10.4, Section 6.1.10.5, and Section 6.1.10.6, respectively.

6.1.10.1 Implementation Details

Throughout this work we focus on a ResNet-18 architecture with batch normalization trained on CIFAR-10 data. We set batchsize to 128, use cross entropy loss, and use as data augmentation: horizontal flip, random crop, and normalization. We train the network for 100 epochs using SGD with momentum with weight decay set to 5e-4 and the momentum hyperparameter set to 0.9. We use an initial learning rate of $h = 0.1$ that is dropped by a factor of 10 every 33 epochs. For pre-trained models we use initial learning rate $h = 0.001$ that is dropped by a factor of 10 after 30 epochs. We obtain our pre-trained models from the PyTorch library, which have been trained on ImageNet. We replace the final fully connected layer to match the number of classes and (when training on CIFAR-10 data) set the first convolutional layer to have 3 input channels, 64 output channels, and a kernel size of 3×3. We perform all our experiments in PyTorch using NVIDIA DGX-1 GPUs and use standard random PyTorch initialization. Our results are all averaged across multiple runs with different random seeds on initialization and data order.

In this work we vary specific aspects of the training problem, such as the optimizer hyperparameters, data, initialization, and model, while keeping all other aspects fixed. For example, we consider deeper ResNet architectures (Figure 6.10), but use the same optimizer, data, and initialization settings as for our base model. But, as described in this work, there are a few experiments where we change multiple aspects of the training problem: we turn off weight decay for both our random label initialization experiment (Figure 6.2B) and our height of barrier initialization experiment (Figure 6.3). For the former, we wanted to study the effect of adversarial initialization that lowered the test accuracy of the final trained network. A network trained on 100% random labels as initialization typically does not affect the test accuracy of the final model, when using SGD with momentum, learning rate decay and weight decay [180]. But turning off weight decay during training in combination with pre-training on random labels does lower the test accuracy. For our height of barrier initialization experiment (Figure 6.3) we needed a setting for which the linear path exhibited clear barriers. For a ResNet-18 this is found for some seeds when weight decay is turned off (Figure 6.7). For the runs that exhibited barriers we saved the state of the model at the height of this barrier and used this as initialization for our model to produce Figure 6.3.

Further, to obtain Figure 1A we used a two hidden layer perceptron with 50 nodes in each hidden layer and ReLU activations. We used Adam with $h =$ 5e-4, without weight decay or any learning rate scheduling. We use batch size 128 for MNIST data and cross entropy loss. We use binary cross entropy loss for the binary classification spiral dataset (as provided in [163]). For our experiments on the spiral dataset we used 10000 training data, 5000 test data points and a batch size of 500.

6.1.10.2 Overview of Results

A summary of all our findings can be found in Figure 6.11.

6.1.10.3 Further Studies on the Role of Initialization

Left/right of Barrier Initialization. In Section 6.1.4 we showed the effect of initializing neural networks at the parameter configuration which corresponds to the height of the barrier along their initial to converged state. We considered a ResNet-18 architecture which was trained without weight decay on CIFAR-10 data. The linear interpolation path in this setting clearly exhibits barriers for some seeds (Section 6.1.6, Figure 6.7). For the runs that exhibited barriers, we saved the model state at the the height of the barrier and used this as initialization for our model. Although it was initialized at a higher loss than occurs at random initialization, the

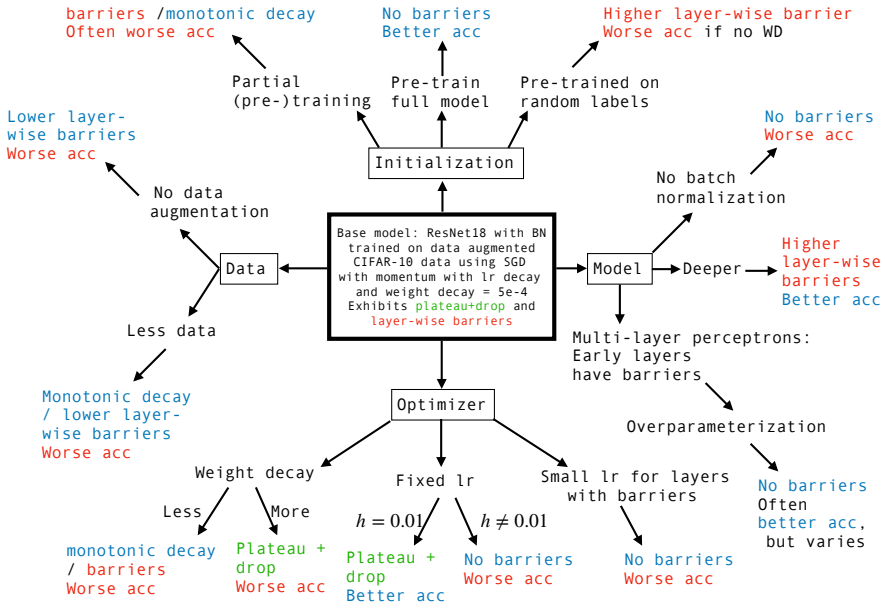

Figure 6.11: The linear path between θ_i and θ_f of our base ResNet-18 architecture exhibits a plateau at the level of random chance until close to the optimum (plateau + drop); it exhibits barriers when evaluating the linear path layer-wise (see Figure $1B$). The base model obtains 92.2% accuracy on CIFAR-10 test data. In this figure we present the change in test accuracy and shape of the linear path between θ_i and θ_f compared to this base model along four axes: the optimizer, the data, the initialization, and the model. The presence of barriers (indicated in red) frequently coincides with a better test accuracy (indicated in blue).

resulting network obtained a higher test accuracy than the network trained from scratch and its linear path exhibited monotonic decay (Figure 6.3). In Figure 6.12 and Table 6.3 we show that these effects are not restricted to initialization at the barrier: choosing an initialization along the linear path to the left or the right of the barrier leads to similar performance improvement (Table 6.3) and typically monotonic decay along the linear path (Figure 6.12).

Table 6.3: ResNet-18 on CIFAR-10 without weight decay. Test loss between θ_i and θ_f for discrete set of $\alpha \in [0, 1]$ in 50 equally spaced subintervals, when trained either from scratch (blue) or from an initialization corresponding to θ_α of the blue line at: the height of the barrier α_{barrier}, before the barrier at $\alpha = \alpha_{\text{barrier}} - 5$, after the barrier at $\alpha = \alpha_{\text{barrier}} + 5$ (red).

Initialization	Training accuracy (%)	Test accuracy (%)
Random initialization	99.44 ±0.12	90.01 ±0.39
Before barrier	99.80 ±0.04	91.05 ±0.21
Height of barrier	99.86 ±0.02	91.12 ±0.18
After barrier	99.88 ± 0.02	90.85 ±0.17

Block-wise Adversarial Initialization with Different Levels of Weight Decay. To further explore the effect of using different initializations for individual convolutional blocks on the linear path and test accuracy, we set one convolutional block to a random label adversarial initialization while using a standard random initialization for the rest of the net. We then re-train using this initialization with different amounts of weight decay (WD). We vary the WD value to study how changes in the full model linear path (Figure 6.7) are reflected by the convolutional block-wise linear path. We find that while adversarial initialization for the first convolutional blocks lowers the test accuracy, using adversarial initialization for convolutional

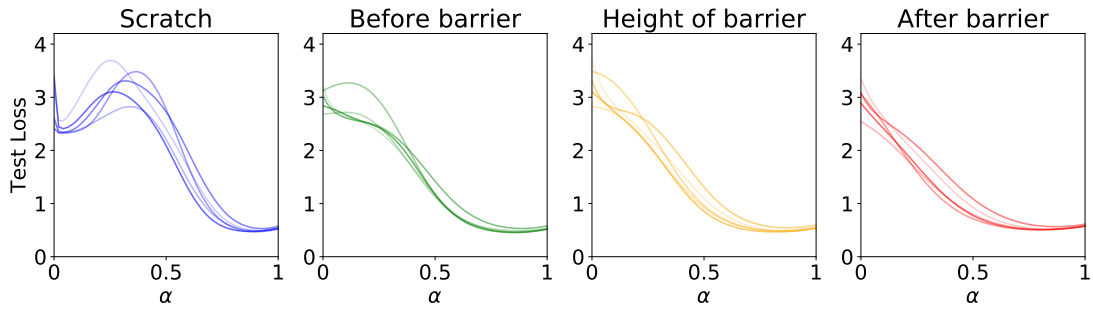

Figure 6.12: ResNet-18 on CIFAR-10 without weight decay. Test loss between θ_i and θ_f for discrete set of $\alpha \in [0, 1]$ in 50 equally spaced subintervals, when trained either from scratch (blue) or from an initialization corresponding to θ_α of the blue line at: the height of the barrier α_{barrier} (orange), just before the barrier at $\alpha = \alpha_{\text{barrier}} - 5$ (green), just after the barrier at $\alpha = \alpha_{\text{barrier}} + 5$ (red).

block 4 slightly increases the final test accuracy compared to training from scratch (compare Figure 6.13 with Figure 6.7). This is also reflected by the shape of the linear path: when using an adversarial initialization for convolutional block 4 the linear path exhibits monotonic decay while others settings exhibit barriers (no WD) and a barrier while other settings exhibit a plateau (WD = 5e-4).

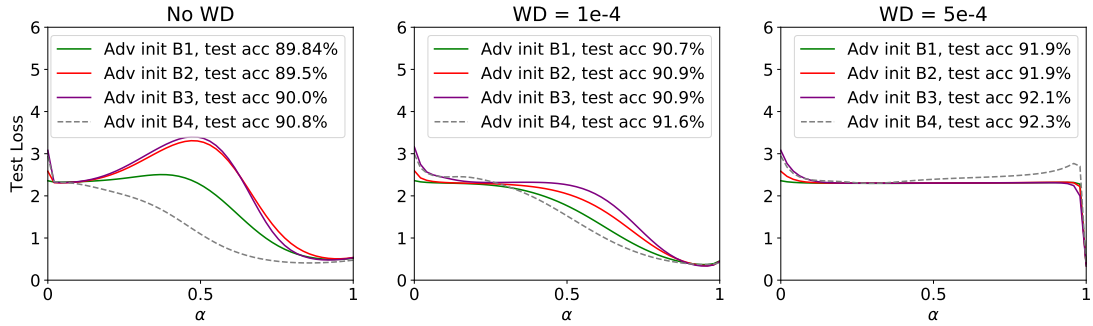

Figure 6.13: Test loss between θ_i and θ_f when one specific conv block (B) was adversarially initialized using random labels. The net was trained either without weight decay (WD) (left), with WD = 1e-4 (middle) or with WD = 5e-4 (right). Averaged over 10 runs.

6.1.10.4 Further Studies on the Role of Data

Less Training Data and No Data Augmentation. Further to our experiments in Section 6.1.5, we here show the combined effect of reducing the number of training data in addition to removing data augmentation (Figure 6.14, left). We find that although this further reduces the test accuracy and presence of block-wise barriers, the overall behaviour for the linear interpolation of the full model (blue dotted line) is maintained (unless the number of training data is further reduced, see Figure 6.6, left).

More Data Augmentation. In Figure 6.15 we show that using additional data augmentation causes the height of layer-wise barriers to slightly increase (for B2 and B3).

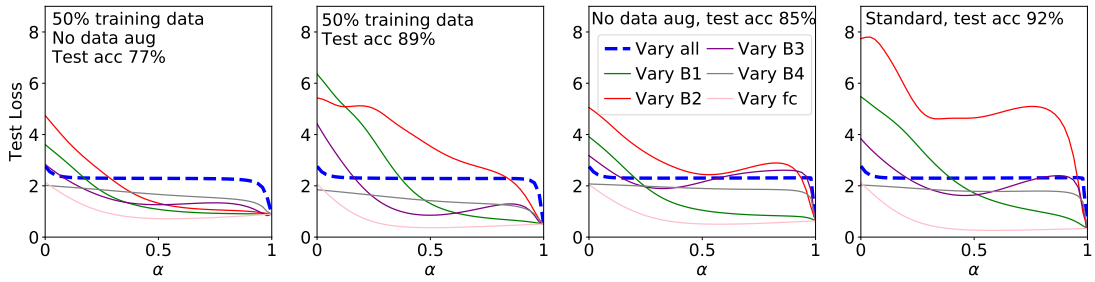

Figure 6.14: Test loss between θ_i and θ_f for ResNet-18 on CIFAR-10 when varying the amount of training data and whether data augmentation (horizontal flips and random crops) is used.

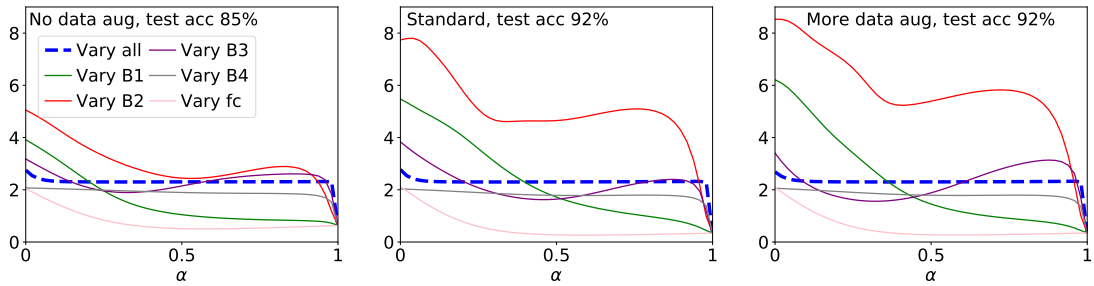

Figure 6.15: Test loss between θ_i and θ_f for ResNet-18 on CIFAR-10 when using either no data augmentation (left), standard data augmentation (horizontal flips and random crops) (middle), or additional data augmentation (brightness = 0.05) (right).

6.1.10.5 Further Studies on the Role of Optimization

Distance. In our fixed learning rate experiment (Figure 6.8) we showed that setting $h = 0.01$ leads to the largest test accuracy of the trained model and non-monotonic behaviour of the loss along the linear path between initial and final state. Meanwhile using a larger ($h = 0.1$) or smaller ($h = 0.001$) learning rate led to a lower test accuracy and monotonic decay of the loss along the linear path. We used the same initialization to compare the different settings, but averaged the results presented in Figure 6.8 over 10 runs using different random seeds. To measure the distance between initial θ_i and final θ_f models we use the following measure (as used in [180]):

$$d(\theta_f, \theta_i) = \frac{\|\theta_f - \theta_i\|_F}{\|\theta_i\|_F}, \tag{6.3}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. There exist many other ways of measuring the distance, which affect the order of magnitude. Since we focus only on the relative distances travelled for different learning rate settings the use of Eq. (6.3) suffices for this purpose. We find that the average distance travelled when using $h = 0.1$ is 1.03, whereas the average distance travelled when using $h = 0.01$ is 0.88 and when using $h = 0.001$ is 0.22. This confirms our intuition that using a larger learning rate will increase the distance travelled.

Layer-wise Sensitivity to Learning Rate. We observed that using a smaller initial learning rate for layers or convolutional blocks which exhibit barriers removes these barriers, but also lowers the test accuracy of the resulting model for a ResNet-18 architecture (Figure 6.9 or Figure 6.16*A*). We observe the same behaviour for a VGG-11 architecture (Figure 6.16*B*). We consider layers (or convolutional blocks) to not exhibit barriers if the loss along their linear path is monotonically decreasing; the exact layers are specified in the caption of Figure 6.16.

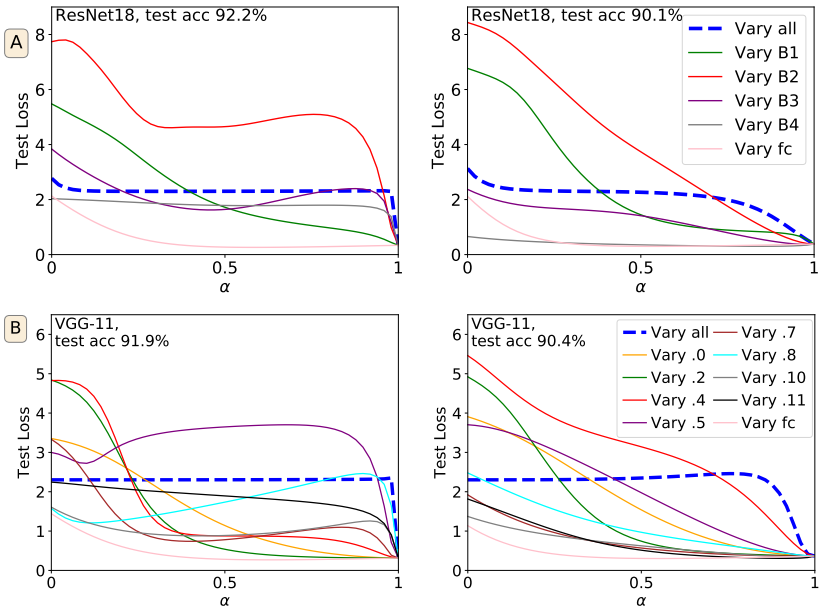

Figure 6.16: Layer-wise (or block-wise) linear interpolation for ResNet-18 *(A)* and VGG-11 *(B)* trained from scratch on CIFAR-10 data using different initial learning rates h_0. For ResNet-18 *(A)*: left: $h_0 = 0.1$ for all, right: $h_0 = 0.01$ for conv block 2, 3, and 4.0 and $h_0 = 0.1$ for rest. For VGG-11 *(B)*: left: $h_0 = 0.1$ for all, right: $h_0 = 0.01$ for layers 4-10 and $h_0 = 0.1$ for rest. Using smaller learning rates for layers that exhibit barriers (ResNet-18: conv block 2, 3, and 4.0, VGG-11: layer 4-10) can remove barriers, but lowers the test accuracy of the model.

6.1.10.6 Further Studies on the Role of the Model

Multi-layer Perceptrons. For multi-layer perceptrons (MLPs) we find that the early layers (close to the input) typically exhibit barriers, whereas later layers exhibit monotonic decay (Figure 6.17, left). By increasing the number of nodes in layers that exhibit barriers one can reduce the presence of barriers for the overall model (Figure 6.17, middle). Meanwhile, layers that do not exhibit barriers can be narrowed without removing the monotonic decay property (Figure 6.17, right).

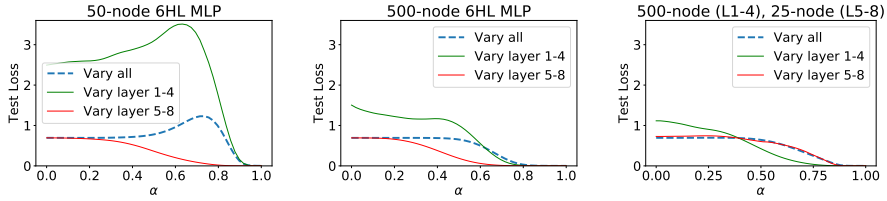

Figure 6.17: A six hidden layer MLP trained on spiral data using different amount of nodes in the hidden layers: 50-node for all layers (left), 500-node for all layers (middle), 500-node for layers that exhibit barriers (layer 1-4) and 25-node for layers that exhibit monotonic decay (layer 5-8) (right). All approaches obtain 100% test accuracy. Increasing the number of nodes in layers that exhibit barriers removes the presence of barriers for the overall model (blue dotted line).

Batch Normalization. In Figure 6.18 we show using parametergroup-wise linear interpolation that the running mean and variance batch normalization parameters appear to govern the behaviour of the full convolutional block. For the linear path of the full model, we corroborate the findings of [185] that the use of batch normalization often leads to non-monotonic

behaviour (Figure 6.19). In the absence of batch normalization, ResNet and VGG architectures often exhibit monotonic decay along their linear path from initial to final state. However, for various datasets we find that MLPs do exhibit barriers in the absence of batch normalization, see e.g., Figure 6.17.

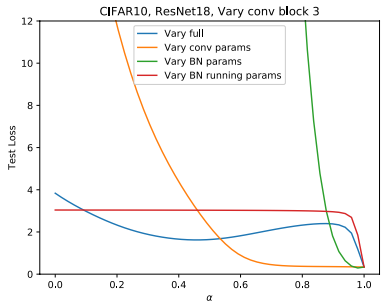

Figure 6.18: We keep all parameters of a ResNet-18 architecture fixed at their final trained position (on CIFAR-10), apart from specific parameter groups in the third convolutional block, which we vary from their initial to final position using the linear interpolation technique. The blue line is when the full third convolutional block is varied. The batch normalization parameters are split into two groups, the running mean and variance (red) and the others (green).

VGG-11. In Figure 6.19 we show the loss along the linear path for a VGG-11 architecture on CIFAR-10 data with (left) and without (right) batch normalization.

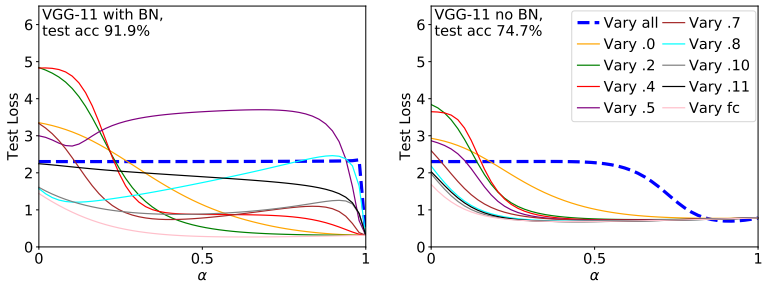

Figure 6.19: Test loss between θ_i and θ_f for VGG-11 on CIFAR-10 when varying whether batch normalization (BN) is used.

Layer width: We find for a VGG-11 architecture with batch normalization on CIFAR-10 data that layers without barriers can be narrowed without affecting the generalization performance (Table 6.4), while layers with barriers cannot. We found that this effect did not generalize to a ResNet-18 architecture, so recommend caution with using the presence or absence of layer-wise layers as an indication of which layers can be narrowed.

Table 6.4: Layer-wise linear interpolation for a VGG-11 with batch normalization trained from scratch on CIFAR-10. This architecture exhibits barriers in layers 4-10 (Figure 6.19B, left). We find that narrowing all non-barrier layers by half only marginally lowers the test accuracy (2nd row), whereas narrowing only one barrier layer (3rd row) already has a large impact. Narrowing two (4th row) or all barrier layers (final row) lowers the test accuracy even further.

Intervention	Test accuracy (%)
Standard	91.93 ±0.19
Narrow no-barrier layers	91.73 ±0.19
Narrow 1 barrier layer	91.53 ±0.20
Narrow 2 barrier layers	91.25 ±0.12
Narrow all barrier layers	91.15 ±0.13

6.2 Potential Limitations of Test Accuracy as the Sole Metric for Neural Network Evaluation

The results presented in this section appeared as a *NeurIPS* workshop paper in [276].

Abstract

We evaluate the classification of both human volunteers and various neural network models on a set of GAN-generated images that reflect the transition from one MNIST class to another. We find that models that obtain the same test accuracy on the standard MNIST test dataset exhibit different behavior on these images. Further, we find that although the number of misclassified images decreases with test accuracy, the spread in predictions over multiple runs on images that are difficult to classify (for humans) also decreases with test accuracy. Our results raise the question of how we want networks to behave on images that could plausibly belong to multiple classes and hint at the value of complementing test accuracy with other evaluation metrics.

6.2.1 Introduction

Two people manually classifying images may easily obtain the same accuracy on a test dataset. How do we determine which of the two is a better classifier? One option is to extend the challenge to a range of different datasets and tasks and evaluate their performance on each of these. Examples of such benchmarks are the Visual Task Adaptation Benchmark (VTAB) [305] for vision and SuperGLUE [283] for natural language processing models. However, the choice of these benchmark datasets may skew the direction of machine learning research as discussed in a recent preprint on "The Benchmark Lottery" by Dehghani et al. [51]. A further limitation that we want to discuss in this work is that across all these datasets the metric that we use to evaluate classifiers remains the same: the performance on a held out test set.

To be accepted in any of the leading machine learning journals, a paper that proposes a new neural network optimization scheme is generally expected to prove that its test accuracy on certain popular datasets, e.g., ImageNet for vision, matches (ideally exceeds) the state of the art. This effectively constrains the development of optimization schemes to a class of algorithms that will perform well using the metric and benchmark datasets generally accepted by the community. If we consider our search for new optimization schemes as being analogous to traversing a loss landscape, then overfitting on specific metric and benchmark choices will result in getting stuck in a local minimum, which is undesirable. Although expanding our range of datasets and metrics does not entirely resolve the problem, we argue that more research on this topic will lead to better and more robust optimizers.

In this work we study whether a model's accuracy on the standard test dataset reflects its performance on a set of images that lie along the classification boundary between two known classes. We do this by comparing the classifications of different network architectures with identical test accuracies to the classifications provided by a group of human volunteers. Our experiment raises the question of how we want networks to behave on images that could belong to either of two known classes and independently illustrates the potential limitations of using test accuracy as the sole metric for model evaluation.

6.2.2 Experiment Design

In this experiment we consider the behavior of different neural network models with identical test accuracies on images that capture the transition from one MNIST class to another. We measure their performance and compare their predictions with the labels assigned by a set of human volunteers.

Creating the Data. We trained a Deep Convolutional Generative Adversarial Network (DCGAN) [228] on MNIST data [152]. We then used the trained DCGAN to generate images, selected ten images depicting the handwritten numbers 0 to 9, and stored the corresponding

ten 100-dimensional vectors $z_0, \ldots, z_9$ which formed the input to the generator. To linearly interpolate between any given z_i and z_j (for $i \neq j$) we used the following equation

$$z_\alpha = (1 - \alpha)z_i + \alpha z_j, \quad \alpha \in [0, 1], i, j \in \{0, 1, \ldots, 9\}. \tag{6.4}$$

To create our dataset we considered 15 examples consisting of 15 pairs of (z_i, z_j), where we set $i < j$ and had each handwritten number represented three times, i.e., each z_k for $k \in \{0, 1, \ldots, 9\}$ appears three times across the 15 pairs. We then linearly interpolated between each pair using Eq. (6.4) with 11 values of α equally spaced between 0 and 1 (inclusive). This resulted in a total of 165 images (Section 6.2.6.2). For illustration we have provided the linear interpolation between a (z_i, z_j) pair in Fig. 6.20.

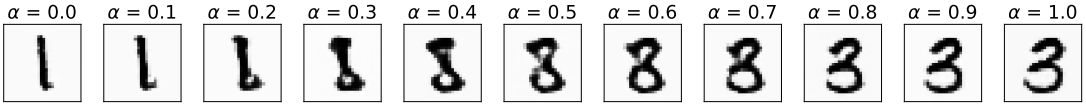

Figure 6.20: Images generated using Eq. (6.4), where $i = 1$ and $j = 3$.

Obtaining the Neural Network Baseline. We trained a multilayer perceptron (MLP), convolutional neural network (CNN) [152], and Vision Transformer (ViT) [57] architecture using cross entropy loss and the Adam [139] optimizer with different hyperparameter choices (more details can be found in Section 6.2.6.1) on the normalized MNIST training dataset. We stopped training once a model reached a certain target accuracy $\pm 0.05\%$ on the test dataset. We considered a maximum target test accuracy of 99.5% (for the CNN architecture) and 98.5% (for the MLP and ViT architectures). The resulting models were then fed the DCGAN-generated images.

Obtaining the Human Baseline. To obtain a human baseline for the DCGAN-generated images, we recruited 10 volunteers to manually label the images. The images were shuffled and then distributed to each of the labelers. The volunteers were not informed of the goal of the task, but were simply asked to label the images with numbers between 0 and 9 to the best of their ability, without consulting others. We then anonymously processed the data and re-sorted the manual labels.

6.2.3 Results

We consider the classification provided by the human volunteers to be the ground truth. This means that if all human classifiers agree on a specific label for an image we consider this to be the true label. On the other hand, if one or more of the human classifiers disagree on the label for a specific image, we consider this image to be a 'boundary case' example.[5] We will contrast the (dis-)agreement between the human classifiers with the (dis-)agreement between an equally large number of neural network (NN) models. As summarized in Table 6.5 the resulting classifications by the neural network models on the DCGAN-generated images can then be grouped into three categories:

Type A: 'Misclassification': Images on which neural network models disagree, whereas human classifiers agree.

Type B: Matching 'ground truth': Images on which neural network models (dis)agree and human classifiers also (dis)agree.[6]

Type C: Images on which neural network models agree, whereas human classifiers disagree.

[5] With a larger number of human classifiers the number of boundary case examples would likely increase, but the number of human classifiers is kept constant throughout this experiment.

[6] An exception is the (rare) images on which both neural network models agree and human classifiers agree but the label assigned is different for the two groups: these images are misclassified by the neural networks.

Table 6.5: The different types that images belong to: type A are misclassified images on which networks disagree; for type B images the networks' predictions match the ground truth (indicated with a ✓); and finally type C is indicated with a "?" as it is unclear how networks should behave in this setting.

	NNs disagree	NNs agree
Human classifiers disagree	✓	?
Human classifiers agree	Misclassification	Human label = NN label: ✓ Human label ≠ NN label: Misclassification

To compare the disagreement amongst a group of neural network models we need to decide which models to group together. We focus on the two extreme cases: the exact same model trained with different random seeds and a diverse set of models (MLP, CNN, and ViT) trained with different optimization schemes (full details in Section 6.2.6.1). The former experiment illustrates the differences between model architectures, whereas the latter is particularly useful for comparing neural networks with humans. Each human is unique and hence we want to contrast the disagreements within the human classifiers with the disagreements amongst an equally diverse set of neural network models.

Do Models with the Same Test Accuracy Misclassify the same Images? We consider the diverse set of neural network models with equivalent test accuracies. Table 6.6 shows that the average % of images on which 2 models disagree is much lower than the % of images on which all 9 models disagree. Further, when focusing on the images on which the human classifiers agree, we see that the actual % of misclassified images, i.e. on which the label predicted by the neural network is different from the unanimous label assigned by human classifiers, is also much smaller per model than when measured across all 9 models. This illustrates that models that were trained to the same test accuracy behave differently on the DCGAN-generated dataset and misclassify (and disagree on) different images.

Table 6.6: Three MLPs, CNNs, and ViTs were trained until a prespecified test accuracy using Adam (more details in Section 6.2.6.1). The "2 models disagree" class reflects the average number of images on which any 2 models (out of the 9) disagree. The "9 models disagree" class reflects the number of images for which at least one model of the 9 has a different label (a more nuanced view on the level of 'disagreement' is discussed in Section 6.2.6.3).

	% of misclassified images		% of images where	
Test accuracy	per model	across 9 models	2 models disagree	9 models disagree
97.5%	5.9 ±1.7%	15.2%	11.3 ±2.9%	28.5%
98%	4.3 ±1.1%	11.5%	8.5 ±2.3%	21.8%
98.5%	3.5 ±0.8%	7.3%	7.4 ±1.8%	18.2%

Do Neural Networks Disagree on the same Images as Human Classifiers? A human classifier disagrees with another human classifier on average on 9.3 ±2.2% of the dataset. As shown in Table 6.6 this falls within the same range as the number of disagreements between two neural network models (4th column), which raises the question of whether the neural networks disagree on the same images as the humans, namely the complicated 'boundary-case' examples. In Table 6.7 we compare the disagreements among the 9 neural network models with the disagreements between 9 humans. There exist a significant number of images on which either humans agree but neural network models do not (type A) or on which humans disagree whereas neural network models agree (type C). It is surprising that for such a diverse set of neural network models there is relatively little disagreement on the for humans difficult to classify 'boundary-case' examples.

Table 6.7: Same setting as in Table 6.6, where type A and type C are as detailed in Table 6.5.

	% of dataset		
Test accuracy	on which NN models disagree	that is type A	that is type C
97.5%	28.5%	15.2%	6.1%
98%	21.8%	11.5%	9.1%
98.5%	18.2%	7.3%	8.5%

How Does the Maximum Softmax Value Across Models Vary with the Class? In Table 6.8 we show that the maximum softmax value across models is much lower for type A images than for type C images. Further, the maximum softmax value for images on which both humans and models disagree is higher than for images on which models disagree but humans do not (type A). Instead we would expect the 'boundary case' examples on which the humans also disagree to have similar maximum softmax values as the other examples on which the models disagree.

Table 6.8: Maximum softmax values across all neural network predictions on different image classes.

		Maximum softmax for test accuracy $\mathcal{A}$		
		$\mathcal{A} = 97.5\%$	$\mathcal{A} = 98\%$	$\mathcal{A} = 98.5\%$
(type A)	NNs disagree & Humans agree	0.83 ±0.178	0.85 ±0.172	0.83 ±0.187
(type B)	NNs disagree & Humans disagree	0.87 ±0.149	0.86 ±0.152	0.89 ±0.138
	NNs agree & Humans agree	0.99 ±0.050	0.99 ±0.051	0.99 ±0.038
(type C)	NNs agree & Humans disagree	0.97 ±0.067	0.96 ±0.091	0.97 ±0.082

How Does the Image Type Vary with Respect to Test Accuracy of the Considered Models? In Figure 6.21 we consider the (dis)agreements between the predictions of a network across 10 random seeds that was trained each time to the same test accuracy. Standard deviations are obtained by repeating this 10 times (i.e., over 100 different seeds in total). Type A images form a clear representation of a specific handwritten digit as all the human volunteers agreed upon their label. We consider type A images to be misclassified by the neural network models and hence we expect the number of type A images to decrease with test accuracy of the considered models. For all three model architectures considered (MLPs, CNNs, and ViTs) we indeed observe that the percentage of type A images decreases with test accuracy (Figure 6.21, left).

Although our expectation for images on which human classifiers agree is clear, it is debatable what the desired behavior of a set of neural networks is for images on which human classifiers disagree. However, if a set of models is initially split across multiple labels for a (for humans) difficult to classify image, but then transitions to predict a unanimous label, we consider this to be an undesirable change as there does not exist a single true answer in this setting. In Figure 6.21 (right) we study how the number of type C images (on which humans disagree, but neural networks agree) changes with the test accuracy of the considered models. We find that the number of difficult to classify images that neural networks unanimously agree on increases with test accuracy.

Finally, Figure 6.21 clearly shows the variations across different network architectures, which were trained using the same optimizer until the same test accuracy was reached. An MLP architecture (blue) misclassifies the lowest number of images (type A) for any given test accuracy, but also predicts 'boundary-case' images with the largest 'certainty' (type C). Test accuracy is therefore not an accurate indicator of the performance of these models on the considered dataset.

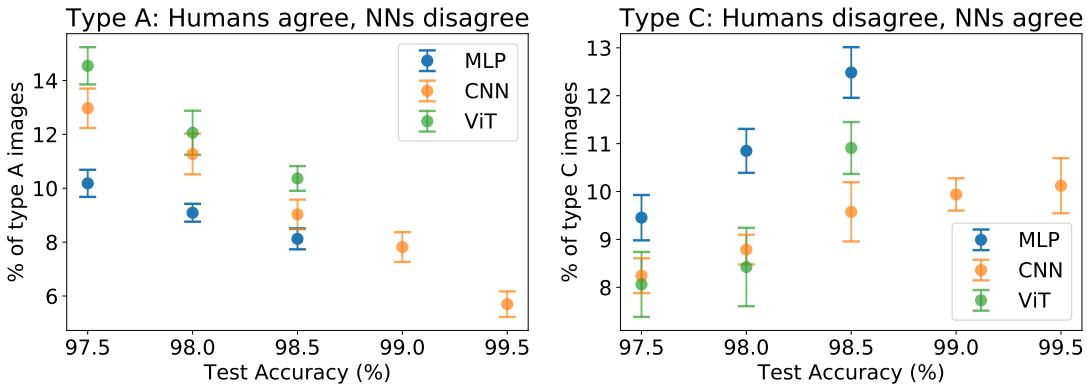

Figure 6.21: The % of type A (left) and type C (right) images with respect to the test accuracy of the neural network models considered. We measure the disagreements between the same neural network architectures trained with 10 different seeds. All models were trained using Adam with h = 5e-4.

6.2.4 Conclusion

To show the limitations of using test accuracy as the only metric for model evaluation we created a set of DCGAN-generated images that capture the transition from one MNIST class to another. We found that models with the same test accuracy misclassify (and disagree on) different sets of images. Further, we find that although a higher test accuracy corresponds with neural network models becoming more unanimous in their label prediction, this may be undesirable for those images on which humans disagree as they do not have a clear class label associated with them. Our experiments also highlight the differences between human and model predictions, as neural networks disagree amongst themselves on a different set of examples than where human classifiers disagree and when in disagreement have higher maximum softmax values on those images on which humans disagree.

6.2.5 Directions for Future Research

The study performed in this work was restricted to 10 human volunteers, who each labeled a total of 165 images. It would be valuable to recreate this experiment with a larger amount of human classifiers and images, and to consider other datasets. This would result in a more accurate measure of human certainty and allow richer comparison between which examples humans and networks find 'difficult'. It would be interesting to then take the label distribution provided by the human labelers to train a model and compare this with knowledge distillation.

To further expand the scope of the experiment, a meticulous study of how the architecture and optimizer choices affect the behavior of the network could potentially lead to insights into inductive biases. For example, in Figure 6.21 the chosen MLP architecture misclassifies the lowest number of images for any given test accuracy, but is also more unanimous in its label prediction on images on which humans disagree. Establishing that this observation holds more generally and if so, understanding *why* it holds, would be valuable.

Furthermore, in this work, we investigated models trained exclusively on MNIST data. It would be interesting to perform the same experiments with CLIP [230], an image-text bi-encoder pretrained on a large corpus of image-text pairs. CLIP is capable of classifying images without finetuning, and has been shown to possess remarkable generalization capabilities. As such, we might find that the behaviour of CLIP meaningfully differs from the behavior of the models we investigated here and more closely aligns with human behavior.

Finally, we believe there is value in exploring other metrics for model evaluation that could complement test accuracy. For the study performed in this work, a first step towards this goal would be to establish what the desired behavior of an ensemble of networks would be on images that could plausibly belong to multiple classes. More generally, it would be valuable to seek

out additional settings in which complementing test accuracy with different metrics might be beneficial.

Acknowledgements

We want to thank the human volunteers for their time and the reviewers for helpful comments. Tiffany Vlaar is supported by the Maxwell Institute Graduate School in Analysis and its Applications, a Centre for Doctoral Training funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016508/01), the Scottish Funding Council, Heriot-Watt University, and the University of Edinburgh.

6.2.6 Supplementary Discussion

6.2.6.1 Experimental Details

Generating the Data. As the generator of the DCGAN architecture [228] we use five convolutional transpose layers with feature map size 64 with ReLU activation and batch normalization. For the discriminator we use five convolutional layers with Leaky ReLU activation and batch normalization. The weights are randomly initialized from $\mathcal{N}(0,0.02)$ and both models are trained using Adam [139] with h = 2e-4 and $\beta_1 = 0.5$ as suggested by Radford et al. [228]. All other details on the data generation are provided in Section 6.2.2.

Model Architectures. We consider three different neural network architectures: a multilayer perceptron (MLP), a convolutional neural network (CNN) [152], and a Vision Transformer (ViT) [57]. As MLP architecture we used a hidden layer with 500 input and output nodes and ReLU activation. Our CNN architecture has 4 convolutional layers and 3 fully connected layers, ReLU activation, batch normalization, max pooling, and dropout ($p = 0.4$) on the middle fully connected layer. Finally, the Vision Transformer (ViT) architecture is adapted from Dosovitskiy et al. [57], where we use 6 transformer blocks, 8 heads in the multi-head attention layer, and a MLP size of 128.

Training. We train the models using PyTorch [219] on the normalized MNIST [152] training dataset with the Adam optimizer. Training is stopped when a model has reached the target test accuracy $\pm 0.05\%$ on the MNIST test dataset. To create the diverse set of models we train three models of each architecture type: for MLP1, CNN1, and ViT1 we use Adam with step size h = 5e-4, batch size (BS) = 128 and no weight decay (WD). For MLP2, CNN2, and ViT2 we use AdamW with h = 5e-4, BS = 128, and WD = 1e-05. To introduce additional variation we train MLP3 using h = 1e-3, BS = 256, WD = 1e-05; CNN3 without weight decay but using additional data augmentation; and ViT3 using WD = 5e-05. To create Figure 6.21 we train the same network across 10 random seeds to the same test accuracy using a batch size of 128 and the Adam optimizer with h = 5e-4. We repeat this process 10 times (so using a total of 100 random seeds) to obtain standard deviations.

6.2.6.2 Dataset

In Figure 6.22 we provide the full 165 DCGAN-generated images considered in this work. These images were shuffled before being given to the human volunteers. On top of each image we provide a ✓ if all 10 human classifiers agreed on the label for this image. If for an image there was disagreement between the human classifiers, we state the different labels proposed.

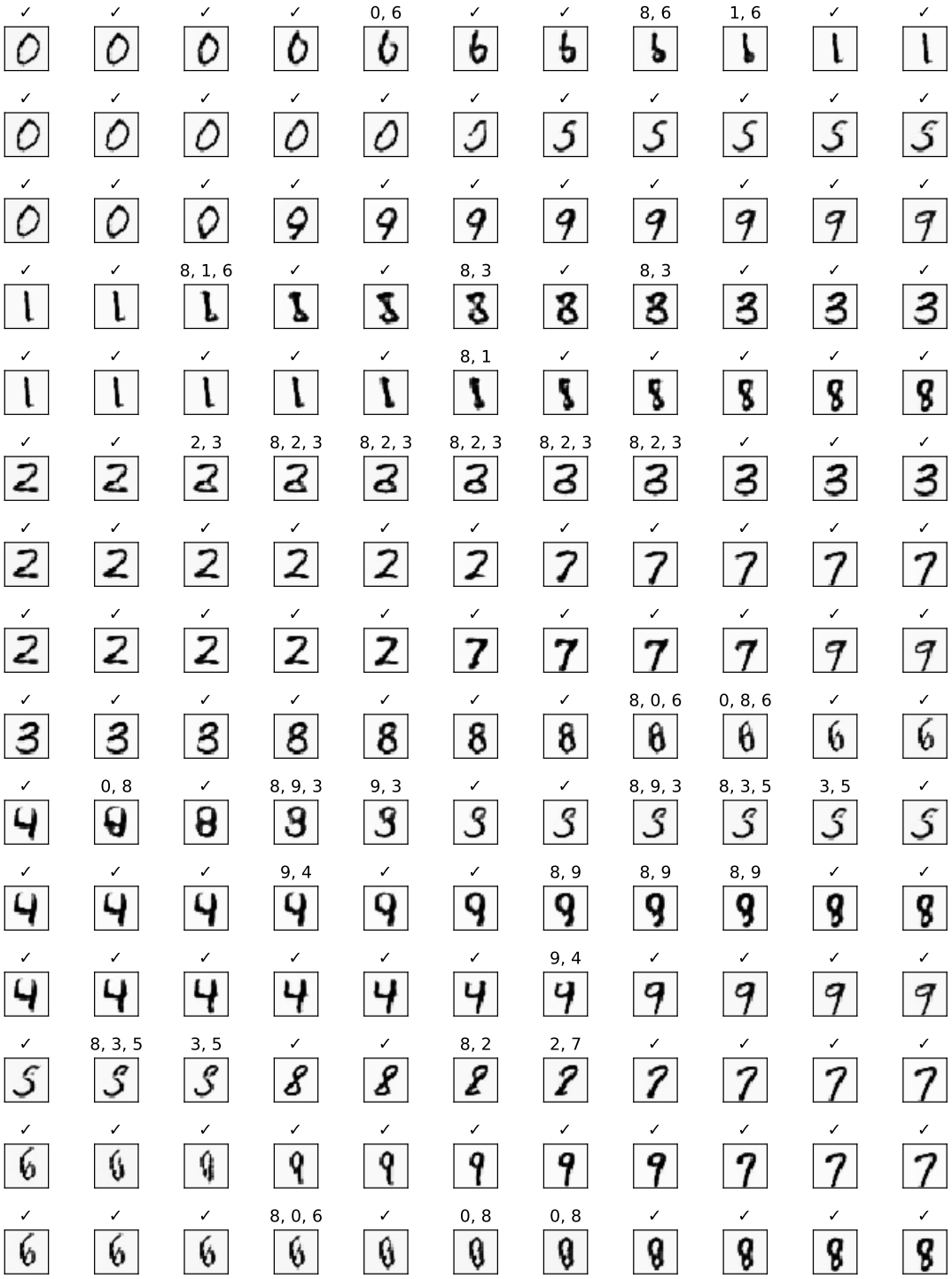

Figure 6.22: The dataset used in this work. The data generation is described in Section 6.2.2. The labels provided by the human volunteers are listed on top of each image (✓ if there was no disagreement).

6.2.6.3 Further Results

In this work we focused on showing the limitations of test accuracy as the sole metric for model evaluation. The results in this section further support this statement and try to answer some follow-up questions the reader may have.

Is an Increased Disagreement Among Human Classifiers Reflected by the Behavior Across Neural Network Models? Throughout this work we have categorized the human classifications into two groups: images on which the humans agree and images on which they disagree. We now want to take a closer look at the latter category. We consider the number of humans disagreeing with the majority vote as a measure of human certainty, i.e., if 2 humans disagree with the majority vote the humans are 80% certain. Correspondingly, we can measure the level of 'certainty' of neural networks by studying how many networks disagree with the most prevalent label assigned to a specific image across all 10 networks. The results are summarized in Table 6.9. It is clear that there is a lot of variation across models that have the same test accuracy. Further, there is not a clear correlation between the human certainty on specific images and the model certainty. CNNs seem to have the lowest overall certainty (but still much higher than humans) on images where humans are $\leq$80% certain on. Since we only have 10 human volunteers in this study, it may be worth repeating this experiment with a much larger sample size to give a more accurate representation of human certainty. Further, the number of images in each "human certainty" class is rather limited and so scaling up along this axis is also worth exploring in future experiments.

Table 6.9: We compute the model certainty of a model across 10 random seeds on images that humans are hc% certain on. The number of images in each class are: 3 ($hc = 90\%$), 9 ($hc = 80\%$), 7 ($hc = 70\%$), 6 ($hc = 60\%$), and 6 ($hc = 50\%$). We exclude the class on which humans are 40% certain as there only exists a single sample point. The standard deviations for the models are obtained by repeating the experiment 10 times (so using 100 random seeds in total).

		Model certainty on images that humans are hc% certain on				
Test acc	Model	$hc = 90\%$	$hc = 80\%$	$hc = 70\%$	$hc = 60\%$	$hc = 50\%$
97.5%	MLP	81.7 ±5.6%	86.6 ±3.0%	90.9 ±2.4%	80.0 ±3.8%	94.7 ±1.5%
	CNN	96.3 ±2.8%	83.3 ±2.1%	81.4 ±2.6%	79.2 ±4.2%	87.3 ±2.9%
	ViT	85.0 ±5.8%	87.2 ±2.8%	82.7 ±3.1%	80.5 ±3.8%	93.5 ±3.1%
98%	MLP	84.0 ±2.0%	86.8 ±3.7%	94.9 ±1.6%	86.5 ±4.7%	93.5 ±1.7%
	CNN	98.0 ±1.6%	83.3 ±2.9%	83.3 ±2.5%	79.2 ±3.8%	88.0 ±2.2%
	ViT	89.0 ±5.4%	85.6 ±2.5%	85.9 ±3.6%	79.3 ±4.5%	91.5 ±1.9%
98.5%	MLP	86.7 ±3.7%	87.2 ±2.4%	94.3 ±2.6%	88.0 ±3.6%	94.2 ±2.0%
	CNN	99.0 ±2.1%	84.6 ±3.5%	81.6 ±3.5%	81.2 ±6.6%	87.8 ±2.0%
	ViT	88.3 ±3.7%	88.8 ±2.2%	90.3 ±2.5%	81.7 ±5.0%	92.7 ±2.6%

Can we Visualize how the Human and Network (dis-)Agreements are Distributed Across the Dataset? In Figure 6.23 we visualize using a heatmap on which images humans disagree amongst themselves, but CNNs do not (in red) and on which images CNNs disagree amongst themselves, but humans do not (in blue). The images on which both networks and humans agree are indicated in gray, while images on which both networks and humans disagree are indicated in white. The significant number of red and blue squares clearly shows that CNNs disagree on a different set of images than the human classifiers. Further, we observe that both the number of blue squares (on which CNNs disagree, but humans do not) and the number of white squares (on which both CNNs and humans disagree) decrease with test accuracy of the considered models, which corroborates the results shown in Figure 6.21.

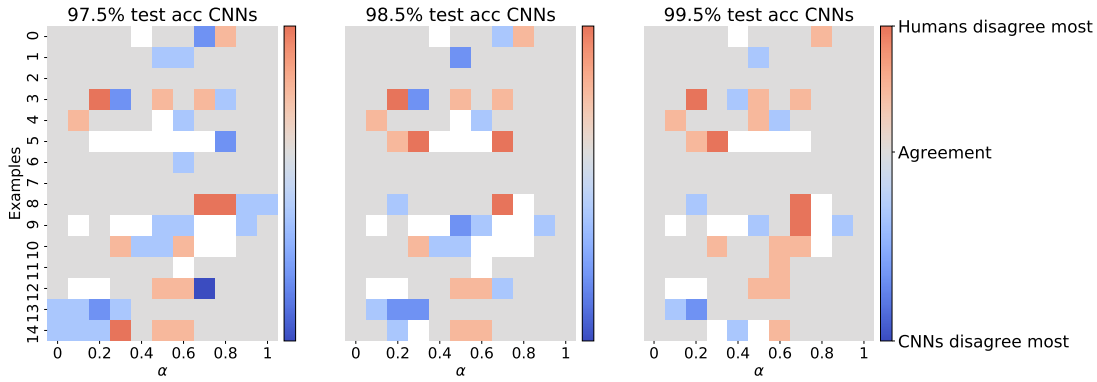

Figure 6.23: The disagreement between 10 human classifiers on the dataset (Figure 6.22) is indicated in red, where the humans are either split over two classes (light red) or three classes (dark red). The disagreement between 10 CNN models with identical test accuracies (obtained using 10 different random seeds) is indicated in blue, where a darker color again indicates that the models were split over more classes. In gray we indicate images on which all human classifiers and all networks agree (although they may be misclassified by the networks). Finally, in white we indicate images on which both the humans and networks disagree. We observe that both the number of blue squares and (to a smaller extent) the number of white squares decreases with test accuracy, thus collaborating the results from Figure 6.21 that CNNs agree more as test accuracy increases.

Conclusion

In this thesis we have studied essential properties of neural network parameterization through a mathematical lens. We have exploited the inherent interdisciplinary nature of deep learning by drawing inspiration from molecular dynamics for the development of new techniques. Published works presented in Chapter 3 and Chapter 4 use discretized stochastic differential equations for the training of neural networks. There exists a rich literature on stochastic differential equations, upon which we can build to provide mathematical underpinnings for our algorithms. This is of paramount importance for deep learning, where there are many open questions on generalization and the nature of the training problem. In Chapter 3 we show the advantages of taking a Bayesian perspective. The use of small temperatures (cold posteriors) allows our algorithms to robustly explore a useful range of parameters and enhances generalization.

In Chapter 3 we also introduced the use of partitioned integrators to exploit the natural layer structure of neural networks. Partitioned algorithms appear throughout this thesis and can be used to obtain enhanced generalization through the use of different optimizers (Chapter 3), enhanced regularization through the use of layer-wise constraints (Chapter 4), or computational speed-up in a transfer learning setting when using multirate techniques (Chapter 5). It is well-known that different neural network layers play different roles, i.e. they learn different features of the data [298, 94, 213], have different levels of sensitivity to re-initialization [307], and the intrinsic dimensionalities of the data representation varies across layers [7]. It therefore makes sense to also train layers differently. We establish several benefits of taking a more fine-grained view on neural network parameterization in this thesis.

There are a number of aspects of the neural network training procedure that remain not well-understood. One way to make the training process more robust is through the Bayesian perspective. Another aspect that we have addressed in this thesis is the standard use of explicit (penalty-based) regularization to mitigate overfitting in neural network training schemes. It is not transparent how such soft constraints directly affect the generalization performance of the model. In Chapter 4 we instead propose to use hard constraints to obtain direct control of the parameter space. This allows us to directly study the effect of specific constraints on the generalization performance of the model.

For the development of the general constrained stochastic differential equations framework used in Chapter 4 we drew inspiration from molecular dynamics, where constraints can be used to increase the maximum usable time step (stability threshold) of the numerical scheme. Molecular dynamics also partly inspired our work on multirate training of neural networks, which is presented in Chapter 5. Similar to applications of multiple time-stepping techniques in molecular dynamics, where the fast dynamics is often cheap to compute compared to the slow dynamics, we purposefully choose the slow part of the neural network system to be computationally expensive. This allows us to obtain significant computational speed-up for transfer learning applications using our multirate approach, without reducing the model's generalization performance. Further, multirate methods can gather information on different features of the data in settings where fixed learning rate approaches fail. Although we focused on vision and natural language processing (NLP) applications in this thesis, we believe multirate methods to contain significant promise for training neural networks on multiscale data.

The final part of the thesis studies existing metrics for deep learning. The high-dimensionality and non-convexity of neural network loss landscapes makes them exceedingly difficult to study. However, enhancing our understanding of neural network loss landscapes may contain the key to training more generalizable neural networks. Therefore, in recent years there has been an effort towards loss landscape visualization. All these low-dimensional visualization techniques

inherently sacrifice information about the high-dimensional landscape, but may still contain useful information about the success of optimization. A simple, yet popular technique studies the loss along the linear path between the initial and final parameters of a neural network after training. Despite its popularity, in Section 6.1 we show that the shape of this linear path is not a reliable indicator of test accuracy. We believe publishing negative results is important when addressing a widespread practice in the community. In Section 6.2 we show the potential limitations of using test accuracy as the sole metric for model evaluation for classification problems. Although the latter work is limited in scope, we hope it opens up a conversation about settings where complementing test accuracy with different metrics might be beneficial.

This thesis opens up several directions for future work. Building on our contribution in Chapter 3 and the recent work by Sekkat and Stoltz [250]: Can the use of Adaptive Langevin dynamics correct for the bias caused by mini-batching for practical deep learning settings? Can we obtain a better heuristic for the layer-wise partitioning by studying the layer-wise diffusion rates? Can we connect the desired additive noise level with local features of the loss landscape? Expanding on our contribution in Chapter 4: Can we use our general constraint framework to obtain desired sparsity levels for our neural network model? Can we extend the theoretical results for our constrained method? Can we use orthogonality constraints to improve the learning of long-term dependencies of recurrent neural networks? For our work on multirate training of neural networks in Chapter 5 it would be interesting to explore different choices of partitioning and to build a multirate version of dropout. Our work in Section 6.1 again reinforces the benefits of taking a more fine-grained (layer-wise) view on neural network training and shows the different levels of sensitivity of different layers to the choice of initialization. It would be interesting to try to understand and exploit this further in future work. For all the techniques proposed in this thesis, we believe it would be valuable to study their applicability for datasets arising from the physical sciences. An overwhelming majority of the machine learning community currently focuses and benchmarks new algorithms on applications in vision and NLP. Diversifying the datasets considered will change the nature of the training problem and lead to interesting challenges where for example, the use of different types of constraints for our constraint framework would be valuable. We believe we should continue to actively seek out and identify the limitations of deep learning, alongside furthering the mathematical foundations of deep learning.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. *12th USENIX symposium on operating systems design and implementation*, pages 265–283, 2016.

[2] A. Abdulle, G. Vilmart, and K. C. Zygalakis. Long time accuracy of Lie-Trotter splitting methods for Langevin dynamics. *SIAM Journal on Numerical Analysis*, 53(1):1–16, 2015.

[3] S. Ahn, A. Korattikara, and M. Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. *International Conference on Machine Learning*, 2012.

[4] L. Aitchison. A statistical theory of cold posteriors in deep neural networks. *International Conference on Learning Representations*, 2020.

[5] H. Andersen. RATTLE: A "velocity" version of the SHAKE algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52:24–34, 1983.

[6] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[7] A. Ansuini, A. Laio, J. H. Macke, and D. Zoccolan. Intrinsic dimension of data representations in deep neural networks. *Advances in Neural Information Processing Systems*, 2019.

[8] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[9] A. Avati, K. Jung, S. Harman, L. Downing, A. Ng, and N. Shah. Improving palliative care with deep learning. *BMC Medical Information and Decision Making*, 18(122), 2018.

[10] J. Ba, G. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using fast weights to attend to the recent past. *Advances in Neural Information Processing Systems*, 2016.

[11] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *Advances in Neural Information Processing Systems Deep Learning Symposium, arXiv:1607.06450*, 2016.

[12] M. Baity-Jesi, L. Sagun, M. Geiger, S. Spigler, G. Ben Arous, C. Cammarota, Y. LeCun, M. Wyart, and G. Biroli. Comparing dynamics: Deep neural networks versus glassy systems. *International Conference on Machine Learning*, 2018.

[13] D. Bakry and M. Émery. Diffusions hypercontractives. In J. Azéma and M. Yor, editors, *Séminaire de Probabilités XIX 1983/84*, pages 177–206, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. ISBN 978-3-540-39397-9.

[14] D. Bakry, I. Gentil, and M. Ledoux. *Analysis and Geometry of Markov Diffusion Operators*, volume 348. Springer Science & Business Media, 2013.

[15] A. J. Ballard, R. Das, S. Martiniani, D. Mehta, L. Sagun, J. D. Stevenson, and D. J. Wales. Energy landscapes for machine learning. *Phys. Chem. Chem. Phys.*, 19:12585–12603, 2017.

[16] N. Bansal, X. Chen, and Z. Wang. Can we gain more from orthogonality regularizations in training deep CNNs? In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4266–4276, 2018.

[17] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[18] P. L. Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

[19] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[20] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 2020. ISSN 0027-8424.

[21] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[22] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. *IEEE International Conference*, 1993.

[23] B. A. Berg. *Markov Chain Monte Carlo Simulations and Their Statistical Analysis*. World Scientific, 2004.

[24] M. Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv: 1701.02434*, 2017.

[25] R. N. Bhattacharya. On the functional central limit theorem and the law of the iterated logarithm for Markov processes. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 60(2):185–201, 1982.

[26] J. J. Biesiadecki and R. D. Skeel. Dangers of multiple time step methods. *Journal of Computational Physics*, 109:318–328, 1993.

[27] D.M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[28] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *International Conference on Machine Learning*, pages 1613–1622, 2015.

[29] L. Bottou. Stochastic gradient descent tricks. In G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 421–436. Springer, 2012.

[30] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[31] N. Bou-Rabee and H. Owhadi. Long-run accuracy of variational integrators in the stochastic context. *SIAM Journal on Numerical Analysis*, 48(1):278–297, 2010.

[32] A. Brock, T. Lim, J. M. Ritchie, and N. J. Weston. Neural photo editing with introspective adversarial networks. *International Conference on Learning Representations*, 2017.

[33] N. Brosse, A. Durmus, and E. Moulines. The promises and pitfalls of stochastic gradient Langevin dynamics. *Advances in Neural Information Processing Systems*, pages 8268–8278, 2018.

[34] A. Brünger, C. B. Brooks, and M. Karplus. Stochastic boundary conditions for molecular dynamics simulations of ST2 water. *Chemical Physics Letters*, 105:495–500, 1984.

[35] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. *International Conference on Knowledge discovery and Data mining. ACM.*, 2006.

[36] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *IEEE International Symposium on Circuits & Systems*, 2017.

[37] E. A. Carter, G. Ciccotti, J. T. Hynes, and R. Kapral. Constrained reaction coordinate dynamics for the simulation of rare events. *Chem. Phys. Lett.*, 156(5):472–477, 1989.

[38] A. Cauchy. Méthode générale pour la résolution de systèmes d'équations simultanées. *Compte rendu des séances de l'académie des sciences*, pages 536–538, 1847.

[39] N. S. Chatterji, B. Neyshabur, and H. Sedghi. The intriguing role of module criticality in the generalization of deep networks. *International Conference on Learning Representations*, 2020.

[40] P. Chaudhari and S. Soatto. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *International Conference on Learning Representations*, 2018.

[41] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. *International Conference on Learning Representations*, 2017.

[42] G. Chen, P. Chen, Y. Shi, C. Hsieh, B. Liao, and S. Zhang. Rethinking the usage of batch normalization and dropout in the training of deep neural networks. *arXiv:1905.05928*, 2019.

[43] T. Chen, E. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. *International Conference on Machine Learning*, 2014.

[44] X. Cheng, N. S. Chatterji, P. L. Bartlett, and M. I. Jordan. Underdamped Langevin MCMC: A non-asymptotic analysis. *arXiv:1707.03663*, 2017.

[45] A. Choromanska, M. Henaff, M. Mathieu, G. Arous, and Y. LeCun. The loss surfaces of multilayer networks. *Journal of Machine Learning Research*, 38:192–204, 2015.

[46] E. M. Constantinescu and A. Sandu. Extrapolated multirate methods for differential equations with multiple time scales. *Journal of Scientific Computing*, 56(1):28–44, 2013.

[47] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. *Advances in Neural Information Processing Systems*, 2015.

[48] S. d'Ascoli, L. Sagun, J. Bruna, and G. Biroli. Finding the needle in the haystack with convolutions: On the benefits of architectural bias. *Advances in Neural Information Processing Systems*, 2019.

[49] Y. Dauphin, R. Pascanu, C. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in Neural Information Processing Systems*, 2014.

[50] Y. Dauphin, H. de Vries, and Y. Bengio. Equilibrated adaptive learning rates for non-convex optimization. *Advances in Neural Information Processing Systems*, pages 1504–1512, 2015.

[51] M. Dehghani, Y. Tay, A. A. Gritsenko, Z. Zhao, N. Houlsby, F. Diaz, D. Metzler, and O. Vinyals. The benchmark lottery. *arXiv:2107.07002*, 2021.

[52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li. ImageNet: A large-scale hierarchical image database. *Conference on Computer Vision and Pattern Recognition*, 2009.

[53] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL*, 2019.

[54] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven. Bayesian sampling using stochastic gradient thermostats. In *Advances in neural information processing systems*, pages 3203–3211, 2014.

[55] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. *International Conference on Machine Learning*, 2017.

[56] J. Dolbeault, C. Mouhot, and C. Schmeiser. Hypocoercivity for kinetic equations with linear relaxation terms. *C. R. Math. Acad. Sci. Paris*, 347(9-10):511–516, 2009.

[57] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2021.

[58] F. Draxler, K. Veschgini, M. Salmhofer, and F. A. Hamprecht. Essentially no barriers in neural network energy landscape. *arXiv:1803.00885*, 2018.

[59] S. S. Du, C. Jin, J. D. Lee, M. I. Jordan, A. Singh, and B. Poczos. Gradient descent can take exponential time to escape saddle points. *Advances in Neural Information Processing Systems*, page 1067–1077, 2017.

[60] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[61] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.

[62] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[63] A. Durmus and E. Moulines. Non-asymptotic convergence analysis for the unadjusted Langevin algorithm. *The Annals of Applied Probability*, 27:1551–1587, 2017.

[64] W. E, W. Ren, and E. Vanden-Eijnden. String method for the study of rare events. *Physical Review B*, 66(5):052301, 2002.

[65] W. E, W. Ren, and E. Vanden-Eijnden. Simplified and improved string method for computing the minimum energy paths in barrier-crossing events. *Journal of Chemical Physics*, 126(16):164103, 2007.

[66] Ch. Engstler and Ch. Lubich. Multirate extrapolation methods for differential equations with different time scales. *Computing*, 58:173–185, 1997.

[67] E. Faou and T. Lelièvre. Conservative stochastic differential equations: Mathematical and numerical analysis. *Mathematics of computation*, 78(268):2047–2074, 2009.

[68] J. A. Feldman. Dynamic connections in neural networks. *Biological Cybernetics*, 46(1): 27–39, 1982.

[69] S. Fort and S. Jastrzębski. Large scale structure of neural network loss landscapes. *International Conference on Machine Learning*, 2019.

[70] S. Fort, H. Hu, and B. Lakshminarayanan. Deep ensembles: A loss landscape perspective. *Bayesian Deep Learning workshop, Advances in Neural Information Processing Systems*, 2019.

[71] V. Fortuin, A. Garriga-Alonso, F. Wenzel, G. Rätsch, R. Turner, M. van der Wilk, and L. Aitchison. Bayesian neural network priors revisited. *Advances in Approximate Bayesian Inference*, 2021.

[72] J. Frankle, D. J. Schwab, and A. S. Morcos. The early phase of neural network training. *International Conference on Learning Representations*, 2020.

[73] Y. Gal. Uncertainty in deep learning. *PhD thesis, University of Cambridge*, 2016.

[74] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv:1506.02142*, 2015.

[75] C. Gardiner. *Handbook of Stochastic Methods for Physics, Chemistry, and the Natural Sciences*. 3rd edn. Springer, New York, 2004.

[76] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. *Advances in Neural Information Processing Systems*, 31:8789–8798, 2018.

[77] C. W. Gear. Multirate methods for ordinary differential equations. *Technical Report, University of Illinois*, 1974.

[78] C. W. Gear and D. R. Wells. Multirate linear multistep methods. *BIT*, 24:484–502, 1984.

[79] M. Geiger, S. Spigler, S. d'Ascoli, L. Sagun, M. Baity-Jesi, G. Biroli, and M. Wyart. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. *Physical Review E*, 100(1):012115, 2019.

[80] M. Geiger, L. Petrini, and M. Wyart. Landscape and training regimes in deep learning. *Physics Reports*, 924:1–18, 2021.

[81] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[82] C. J. Geyer. Markov Chain Monte Carlo maximum likelihood. *Computer Science and Statistics*, 1991.

[83] B. Ginsburg, P. Castonguay, O. Hrinchuk, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, H. Nguyen, and J. M. Cohen. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *arXiv:1905.11286*, 2019.

[84] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings*, pages 315–323, 2011.

[85] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *International Conference on Machine Learning*, 2013.

[86] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.

[87] I. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *International Conference on Learning Representations*, 2015.

[88] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[89] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *CoRR*, 2017.

[90] A. Graves. Practical variational inference for neural networks. *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[91] A. Gray, E. Abbena, and S. Salamon. *Modern Differential Geometry of Curves and Surfaces with MATHEMATICA, 3rd edition*. Chapman and Hall, 2006.

[92] M. Günther and P. Rentrop. Multirate row methods and latency of electric circuits. *Appl. Numer. Math.*, 13:83–102, 1993.

[93] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. *International Conference on Machine Learning*, pages 1321–1330, 2017.

[94] Y. Hao, L. Dong, F. Wei, and K. Xu. Visualizing and understanding the effectiveness of BERT. *EMNLP*, 2019.

[95] W. K. Hastings. Monte Carlo sampling methods using Markov Chains and their applications. *Biometrika*, 57:97–109, 1970.

[96] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[97] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[98] K. He, R. Girshick, and P. Dollár. Rethinking ImageNet pre-training. *ICCV*, 2019.

[99] D. P. Herzog. Exponential relaxation of the Nosé-Hoover equation under Brownian heating. *Communications in Mathematical Sciences*, 16(8):2231–2260, 2018.

[100] G. E. Hinton and D. C. Plaut. Using fast weights to deblur old memories. *In Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

[101] G. E. Hinton and D. van Camp. Keeping the neural networks simple by minimizing the description length of the weights. *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13, 1993.

[102] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[103] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[104] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

[105] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[106] A. Hoerl and R. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[107] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[108] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1), 2013.

[109] W. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A.*, 31(3):1695–1697, 1985.

[110] J. Howard and S Ruder. Universal language model fine-tuning for text classification. *ACL*, 2018.

[111] W. Hu, C. J. Li, L. Li, and J.-G. Liu. On the diffusion approximation of nonconvex stochastic gradient descent. *Annals of Mathematical Sciences and Applications*, 4(1), 2019.

[112] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get M for free. *International Conference on Learning Representations*, 2017.

[113] L. Huang, X. Liu, B. Lang, A. Wei Yu, and B. Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[114] W. R. Huang, Z. Emam, M. Goldblum, L. Fowl, J. K. Terry, F. Huang, and T. Goldstein. Understanding generalization through visualizations. *arXiv: 1906.03291*, 2019.

[115] Z. Huang, Z. Zeng, B. Liu, D. Fu, and J. Fu. Pixel-BERT: Aligning image pixels with text by deep multi-modal transformers. *CoRR*, 2020.

[116] M. Huh, P. Agrawal, and A. A. Efros. What makes ImageNet good for transfer learning? *arXiv:1608.08614*, 2016.

[117] L. Hörmander. Hypoelliptic second order differential equations. *Acta Mathematica*, 119: 147–171, 1967.

[118] L. Hörmander. *The Analysis of Linear Partial Differential Operators I–IV*. Springer, 1985.

[119] Y. Ida, Y. Fujiwara, and S. Iwamura. Adaptive learning rate via covariance matrix based preconditioning for deep neural networks. *International Joint Conferences on Artificial Intelligence*, 2017.

[120] D. J. Im, M. Tao, and K. Branson. An empirical analysis of deep network loss surfaces. *arXiv:1612.04010*, 2016.

[121] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 2015.

[122] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *Uncertainty in Artificial Intelligence*, 2018.

[123] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson. What are Bayesian neural network posteriors really like? *International Conference on Machine Learning*, 2021.

[124] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? *ICCV*, 2009.

[125] S. Jastrzębski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. Three factors influencing minima in SGD. *ICANN*, 2018.

[126] S. Jastrzębski, M. Szymczak, S. Fort, D. Arpit, J. Tabor, K. Cho, and K. Geras. The break-even point on optimization trajectories of deep neural networks. *International Conference on Learning Representations*, 2020.

[127] K. Jia, S. Li, Y. Wen, T. Liu, and D. Tao. Orthogonal deep neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[128] A. Jones and B. Leimkuhler. Adaptive stochastic methods for sampling driven molecular systems. *The Journal of Chemical Physics*, 135(8):084125, 2011.

[129] D. Justus, J. Brennan, S. Bonner, and A. S. McGough. Predicting the computational cost of deep learning models. *CoRR*, 2018.

[130] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

[131] I. Karatzas and S. Shreve. *Brownian Motion and Stochastic Calculus*. Springer, second edition, 1998.

[132] A. Karpathy. Connecting images and natural language. *PhD Thesis*, 2016.

[133] K. Kawaguchi. Deep learning without poor local minima. *Advances in Neural Information Processing Systems*, 2016.

[134] A. Kendall and Y. Gal. What uncertainties do we need in Bayesian deep learning for computer vision? *Advances in Neural Information Processing Systems*, 2017.

[135] N. S. Keskar and R. Socher. Improving generalization performance by switching from Adam to SGD. *arXiv:1712.07628*, 2017.

[136] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*, 2017.

[137] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 1952.

[138] D. E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[139] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2015.

[140] C. Kipnis and S. R. S. Varadhan. Central limit theorem for additive functionals of reversible Markov processes and applications to simple exclusions. *Communications in Mathematical Physics*, 104(1):1–19, 1986.

[141] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[142] A. Der Kiureghian and O. Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2):105–112, 2009.

[143] W. Kliemann. Recurrence and invariant measures for degenerate diffusions. *The annals of probability*, pages 690–707, 1987.

[144] P. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Applications of Mathematics. Springer, 1992.

[145] A. Kristiadi, M. Hein, and P. Hennig. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. *International Conference on Machine Learning*, 2020.

[146] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[147] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 2012.

[148] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[149] H. Kushner and G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*, volume 35. Springer Science & Business Media, 2003.

[150] J. Lafond, N. Vasilache, and L. Bottou. Diagonal rescaling for neural networks. *arXiv:1705.09319*, 2017.

[151] J. Lan, R. Liu, H. Zhou, and J. Yosinski. LCA: Loss change allocation for neural network training. *arXiv: 1909.01440*, 2019.

[152] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[153] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, page 9–50. Springer, 1998.

[154] J. M. Lee. *Introduction to Riemannian manifolds*, volume 2. Springer, 2018.

[155] T. K. Leen and G. B. Orr. Optimal stochastic search and adaptive momentum. *Advances in Neural Information Processing Systems*, 1993.

[156] B. Leimkuhler and C. Matthews. Rational construction of stochastic numerical methods for molecular sampling. *Applied Mathematics Research eXpress*, 2013(1):34–56, 2013.

[157] B. Leimkuhler and C. Matthews. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Interdisciplinary Applied Mathematics. Springer, 2015.

[158] B. Leimkuhler and C. Matthews. Efficient molecular dynamics using geodesic integration and solvent–solute splitting. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 472(2189):20160138, 2016.

[159] B. Leimkuhler and S. Reich. A reversible averaging integrator for multiple time-scale dynamics. *Journal of Computational Physics*, 171(1):95–114, 2001.

[160] B. Leimkuhler and S. Reich. *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics, volume 14. Cambridge University Press, 2005.

[161] B. Leimkuhler and X. Shang. Adaptive thermostats for noisy gradient systems. *SIAM Journal on Scientific Computing*, 38(2):A712–A736, 2016.

[162] B. Leimkuhler, C. Matthews, and G. Stoltz. The computation of averages from equilibrium and nonequilibrium Langevin molecular dynamics. *IMA Journal of Numerical Analysis*, 36(1):13–79, 2016.

[163] B. Leimkuhler, C. Matthews, and T. Vlaar. Partitioned integrators for thermodynamic parameterization of neural networks. *Foundations of Data Science*, 1(4):457–489, 2019.

[164] B. Leimkuhler, M. Sachs, and G. Stoltz. Hypocoercivity properties of adaptive Langevin dynamics. *SIAM Journal on Applied Mathematics*, 80(3):1197–1222, 2020.

[165] B. Leimkuhler, T. Vlaar, T. Pouchon, and A. Storkey. Better training using weight-constrained stochastic dynamics. *International Conference on Machine Learning (ICML)*, 2021.

[166] T. Lelièvre, M. Rousset, and G. Stoltz. Langevin dynamics with constraints and computation of free energy differences. *Mathematics of computation*, 81(280):2071–2125, 2012.

[167] T. Lelièvre, G. Stoltz, and W. Zhang. Multiple projection MCMC algorithms on submanifolds. *arXiv:2003.09402*, 2020.

[168] T. Lelièvre and G. Stoltz. Partial differential equations and stochastic methods in molecular dynamics. *Acta Numerica*, 25:681–880, 2016.

[169] T. Lelièvre, G. Stoltz, and M. Rousset. *Free Energy Computations: A Mathematical Perspective*. Imperial College Press, 2010.

[170] C. Li, C. Chen, D. Carlson, and L. Carin. Preconditioned stochastic gradient Langevin dynamics for deep neural networks. *AAAI Conference on Artificial Intelligence*, 2016.

[171] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the intrinsic dimension of objective landscapes. *International Conference on Learning Representations*, 2018.

[172] F.-F. Li, A. Karpathy, and J. Johnson. CS231n: Convolutional neural networks for visual recognition, 2021. URL `https://cs231n.github.io/convolutional-networks/`.

[173] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems*, 2018.

[174] H. Li, P. Chaudhari, H. Yang, M. Lam, A. Ravichandran, R. Bhotika, and S. Soatto. Rethinking the hyperparameters for fine-tuning. *International Conference on Learning Representations*, 2020.

[175] Q. Li, S. Haque, C. Anil, J. Lucas, R. Grosse, and J. Jacobsen. Preventing gradient attenuation in Lipschitz constrained convolutional networks. *Advances in Neural Information Processing Systems*, 2019.

[176] Q. Li, C. Tai, and W. E. Stochastic modified equations and dynamics of stochastic gradient algorithms I: Mathematical foundations. *Journal of Machine Learning Research*, 20(40):1–47, 2019.

[177] X. Li, S. Chen, X. Hu, and J. Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[178] Y. Li, C. Wei, and T. Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *Advances in Neural Information Processing Systems*, 2019.

[179] Z. Li, S. Malladi, and S. Arora. On the validity of modeling SGD with stochastic differential equations (SDEs). *Advances in Neural Information Processing Systems*, 2021.

[180] S. Liu, D. Papailiopoulos, and D. Achlioptas. Bad global minima exist and SGD can reach them. *Advances in Neural Information Processing Systems*, 2020.

[181] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*, 2019.

[182] I. Loshchilov and F. Hutter. Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*, 2017.

[183] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations*, 2019.

[184] J. Lucas, J. Bae, M. Zhang, J. Ba, R. Zemel, and R. Grosse. On monotonic linear interpolation of neural network parameters. *OptML Advances in Neural Information Processing Systems workshop*, 2020.

[185] J. Lucas, J. Bae, M. Zhang, S. Fort, R. Zemel, and R. Grosse. Analyzing monotonic linear interpolation in neural network loss landscapes. *International Conference on Machine Learning*, 2021.

[186] L. Luo, Y. Xiong, Y. Liu, and X. Sun. Adaptive gradient methods with dynamic bound of learning rate. *International Conference on Learning Representations*, 2019.

[187] P. Luo, X. Wang, W. Shao, and Z. Pen. Towards understanding regularization in batch normalization. *International Conference on Learning Representations*, 2019.

[188] Q. Ma, J. Izaguirre, and R. Skeel. Verlet-I/r-RESPA/Impulse is limited by nonlinear instabilities. *SIAM Journal on Scientific Computing*, 24(6):1951–1973, 2003.

[189] D. J. C. MacKay. Bayesian methods for adaptive models. *PhD thesis, California Institute of Technology*, 1991.

[190] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[191] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[192] S.S. Mannelli, E. Vanden-Eijnden, and L. Zdeborová. Optimization and generalization of shallow neural networks with quadratic activation functions. *Advances in Neural Information Processing Systems*, 2020.

[193] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[194] E. Marinari and G. Parisi. Simulated tempering: A new Monte Carlo scheme. *Europhysics Letters*, 1992.

[195] 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.

[196] S. McCandlish, J. Kaplan, D. Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv:1812.06162*, 2018.

[197] P. Mehta, M. Bukov, C. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *arXiv: 1803.08823*, 2018.

[198] S. Mei and A. Montanari. The generalization error of random features regression: Precise asymptotics and double descent curve. *Communications on Pure and Applied Mathematics*, 2021.

[199] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. *International Conference on Learning Representations*, 2017.

[200] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21, 1953.

[201] S. P. Meyn and R. L. Tweedie. Stability of Markovian processes II: Continuous-time processes and sampled chains. *Advances in Applied Probability*, 25(3):487–517, 1993.

[202] D. Murfet, S. Wei, M. Gong, H. Li, J. Gell-Redman, and T. Quella. Deep learning is singular, and that's good. *arXiv:2010.11560*, 2020.

[203] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[204] Z. Nado, J. M. Gilmer, C. J. Shallue, R. Anil, and G. E. Dahl. A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes. *arXiv:2102.06356*, 2021.

[205] V. Nair and G. E. Hinton. Rectified Linear Units improve restricted Boltzmann machines. *International Conference on Machine Learning*, 2010.

[206] R. M. Neal. Bayesian learning for neural networks. *PhD thesis, University of Toronto*, 1995.

[207] R. M. Neal. MCMC using Hamiltonian Dynamics. In S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, editors, *Handbook of Markov Chain Monte Carlo*, pages 113–162. Chapman & Hall / CRC Press, 2011.

[208] R. M. Neal. *Bayesian Learning for Neural Networks*, volume 118. Springer Science & Business Media, 2012.

[209] Y. Nesterov. A method of solving a convex programming problem with convergence rate O(1/k2). *Soviet Mathematics Doklady*, 27:372–376, 1983.

[210] B. Neyshabur, R. Tomioka, and N. Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *Proceeding of the International Conference on Learning Representations workshop track*, 2015.

[211] B. Neyshabur, R. Tomioka, and N. Srebro. Norm-based capacity control in neural networks. *Proceeding of the 28th Conference on Learning Theory*, 2015.

[212] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. *Advances in Neural Information Processing Systems*, pages 5949–5958, 2017.

[213] B. Neyshabur, H. Sedghi, and C. Zhang. What is being transferred in transfer learning? *Advances in Neural Information Processing Systems*, 2020.

[214] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[215] L. Noci, K. Roth, G. Bachmann, S. Nowozin, and T. Hofmann. Disentangling the roles of curation, data-augmentation and the prior in the cold posterior effect. *Advances in Neural Information Processing Systems*, 2021.

[216] S. Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of Chemical Physics*, 81(1):511–519, 1984.

[217] I. Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[218] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[219] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. 2017.

[220] G. A. Pavliotis. *Stochastic Processes and Applications: Diffusion Processes, the Fokker-Planck and Langevin Equations*. Texts in Applied Mathematics, volume 60. Springer, 2014.

[221] J. Pennington, S. Schoenholz, and S. Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: Theory and practice. In *Advances in Neural Information Processing Systems*, pages 4785–4795, 2017.

[222] J. Pennington, S. Schoenholz, and S. Ganguli. The emergence of spectral universality in deep networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1924–1932, 2018.

[223] P. Persson. The level set method. Lecture notes MIT 16.920J / 2.097J / 6.339J, Numerical Methods for Partial Differential Equations, October 2006.

[224] G. Philipp, D. Song, and J. G. Carbonell. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv:1712.05577*, 2018.

[225] E. Pollak, A. Auerbach, and P. Talkner. Observations on rate theory for rugged energy landscapes. *Biophysical Journal*, 95:4258–4265, 2008. URL `https://doi.org/10.1529/biophysj.108.136358`.

[226] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[227] H. Qi, E. R. Sparks, and A. Talwalkar. PALEO: A performance model for deep neural networks. *International Conference on Learning Representations*, 2017.

[228] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations*, 2016.

[229] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *OpenAI*, 2018.

[230] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. *International Conference on Machine Learning*, 2021.

[231] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio. Transfusion: Understanding transfer learning for medical imaging. *Advances in Neural Information Processing Systems*, 2019.

[232] S. Reich and C. Cotter. *Probabilistic Forecasting and Bayesian Data Assimilation*. Cambridge University Press, 2015.

[233] J. R. Rice. Split Runge-Kutta methods for simultaneous equations. *Journal of Research of the National Institute of Standards and Technology*, 60, 1960.

[234] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[235] D. A. Roberts, S. Yaida, and B. Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022. URL `https://arxiv.org/abs/2106.10165`.

[236] G. O. Roberts and R. L. Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

[237] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca. Regularizing CNNs with locally constrained decorrelations. *International Conference on Learning Representations*, 2017.

[238] P. J. Rossky, J. D. Doll, and H. L. Friedman. Brownian dynamics as smart Monte Carlo simulation. *The Journal of Chemical Physics*, 69(10):4628–4633, 1978.

[239] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, page 318–362, 1986.

[240] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[241] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[242] J.-P. Ryckaert, G. Ciccotti, and H. J. Berendsen. Numerical integration of the Cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341, 1977.

[243] M. Sachs, B. Leimkuhler, and V. Danos. Langevin dynamics with variable coefficients and nonconservative forces: From stationary states to numerical methods. *Entropy*, 19 (12):647, 2017.

[244] L. Sagun, L. Bottou, and Y. LeCun. Singularity of the Hessian in deep learning. *International Conference on Learning Representations*, 2017.

[245] L. Sagun, U. Evci, U. Güney, Y. Dauphin, and L. Bottou. Empirical analysis of the Hessian of over-parametrized neural networks. *International Conference on Learning Representations*, 2018.

[246] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *Energy Efficient Machine Learning and Cognitive Computing Workshop, Advances in Neural Information Processing Systems*, 2019.

[247] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? *Advances in Neural Information Processing Systems*, 2018.

[248] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[249] K. T. Schütt, F. Arbabzadah, K. R. Müller S. Chmiela, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature Communications*, 2017.

[250] I. Sekkat and G. Stoltz. Removing the mini-batching error in Bayesian inference using Adaptive Langevin dynamics. *arXiv:2105.10347*, 2021.

[251] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[252] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20:1–49, 2019.

[253] X. Shang, Z. Zhu, B. Leimkuhler, and A. J. Storkey. Covariance-controlled adaptive Langevin thermostat for large-scale Bayesian sampling. In *Advances in Neural Information Processing Systems*, pages 37–45, 2015.

[254] S. Shen, L. H. Li, H. Tan, M. Bansal, A. Rohrbach, K.-W. Chang, Z. Yao, and K. Keutzer. How much can CLIP benefit vision-and-language tasks? *CoRR*, 2021.

[255] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[256] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[257] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor. Layer-specific adaptive learning rates for deep networks. *International Conference on Machine Learning*, 2015.

[258] L. N. Smith. Cyclical learning rates for training neural networks. *Worshop on Application of Computer Vision*, 2017.

[259] S. L. Smith and Q. V. Le. A Bayesian perspective on generalization and stochastic gradient descent. *International Conference on Learning Representations*, 2018.

[260] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *Conference on Empirical Methods in Natural Language Processing*, 2013.

[261] N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In *International Conference on Computational Learning Theory*, pages 545–560. Springer, 2005.

[262] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[263] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning*, 2013.

[264] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.

[265] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Computer Vision and Pattern Recognition*, 2015.

[266] Y. W. Teh, A. H. Thiery, and S. J. Vollmer. Consistency and fluctuations for Stochastic Gradient Langevin Dynamics. *Journal of Machine Learning Research*, 17:1–33, 2016.

[267] F. Thalmann and J. Farago. Trotter derivation of algorithms for Brownian and dissipative particle dynamics. *The Journal of Chemical Physics*, 127:124109, 2007.

[268] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B*, 58(1):267–288, 1996.

[269] T. Tieleman and G. Hinton. Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[270] M. E. Tuckerman, B. J. Berne, and G. J. Martyna. Molecular dynamics algorithm for multiple time scales: Systems with long range forces. *The Journal of Chemical Physics*, 94(10):6811–6815, 1991.

[271] M. E. Tuckerman, B. J. Berne, and G. J. Martyna. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics*, 97(3):1990–2001, 1992.

[272] V. N. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for consistency of the method of empirical risk minimization. *Pattern Recognition and Image Analysis*, 1(3):284–305, 1991.

[273] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[274] L. Verlet. Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review*, 159(1):98–103, 1967.

[275] C. Villani. Hypocoercivity. *Memoirs of the American Mathematical Society*, 202(950), 2009.

[276] T. Vlaar. Neural network behavior at the classification boundary. *Advances in Neural Information Processing Systems, I (Still) Can't Believe It's Not Better workshop*, 2021.

[277] T. Vlaar and J. Frankle. What can linear interpolation of neural network loss landscapes tell us? *arXiv:2106.16004*, 2021.

[278] T. Vlaar and B. Leimkuhler. Multirate training of neural networks. *arXiv:2106.10771*, 2021.

[279] T. Vlaar, B. Leimkuhler, and C. Matthews. Partitioned integrators for thermodynamic parameterization of neural networks. *Advances in Neural Information Processing Systems, Machine Learning and the Physical Sciences workshop*, 2019.

[280] S. J. Vollmer, K. C. Zygalakis, and Y. W. Teh. Exploration of the (non-) asymptotic bias and variance of Stochastic Gradient Langevin Dynamics. *Journal of Machine Learning Research*, 17(159):1–48, 2016.

[281] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3570–3578. Journal of Machine Learning Research, 2017.

[282] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using DropConnect. *International Conference on Machine Learning*, 2013.

[283] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems*, 2019.

[284] C. Wei, S. Kakade, and T. Ma. The implicit and explicit regularization effects of dropout. *arXiv:2002.12915*, 2020.

[285] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, pages 681–688, 2011.

[286] Y. Wen, K. Luk, M. Gazeau, G. Zhang, H. Chan, and J. Ba. An empirical study of large-batch stochastic gradient descent with structured covariance noise. *International Conference on Artificial Intelligence and Statistics*, 2020.

[287] F. Wenzel, K. Roth, B. S. Veeling, J. Swiatkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin. How good is the Bayes posterior in deep neural networks really? *International Conference on Machine Learning*, 2020.

[288] P. Williams. Bayesian regularization and pruning using a Laplace prior. *Neural Computation*, 7:117–143, 1995.

[289] A. 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*, pages 4148–4158, 2017.

[290] J. Wu, W. Hu, H. Xiong, J. Huan, V. Braverman, and Z. Zhu. On the noisy gradient descent that generalizes as SGD. *International Conference on Machine Learning*, 2020.

[291] L. Wu, Z. Zhu, and W. E. Towards understanding generalization of deep learning: Perspective of loss landscapes. *International Conference on Machine Learning*, 2017.

[292] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

[293] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402, 2018.

[294] D. Xie, J. Xiong, and S. Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6176–6185, 2017.

[295] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *CoRR, arXiv: 1505.00853*, 2015.

[296] Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 2007.

[297] Z. Yao, A. Gholami, K. Keutzer, and M. Mahoney. PyHessian: Neural networks through the lens of the Hessian. *IEEE BigData, arXiv:1912.07145*, 2020.

[298] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems*, 2014.

[299] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. *arXiv:1708.03888*, 2017.

[300] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer. ImageNet training in minutes. *Proceedings of the 47th International Conference on Parallel Processing*, 2018.

[301] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C. Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. *International Conference on Learning Representations*, 2020.

[302] A. W. Yu, Q. Lin, R. Salakhutdinov, and J. Carbonell. Block-normalized gradient method: An empirical study for training deep neural networks. *arXiv:1707.04822*, 2018.

[303] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, 2016.

[304] M. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, 2012.

[305] X. Zhai, J. Puigcerver, A. Kolesnikov, P. Ruyssen, C. Riquelme, M. Lucic, J. Djolonga, A. S. Pinto, M. Neumann, A. Dosovitskiy, L. Beyer, O. Bachem, M. Tschannen, M. Michalski, O. Bousquet, S. Gelly, and N. Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv:1910.04867*, 2019.

[306] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations*, 2017.

[307] C. Zhang, S. Bengio, and Y. Singer. Are all layers created equal? *arXiv:1902.01996*, 2019.

[308] J. Zhang, S. Karimireddy, A. Veit, S. Kim, S. Reddi, S. Kumar, and S. Sra. Why are adaptive methods good for attention models? *Advances in Neural Information Processing Systems*, 2020.

[309] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. *Advances in Neural Information Processing Systems*, 2015.

[310] Y. Zhang, P. Liang, and M. Charikar. A hitting time analysis of stochastic gradient Langevin dynamics. *Conference on Learning Theory*, pages 1980–2022, 2017.

[311] J. Zhou, M.N. Do, and J. Kovacevic. Special paraunitary matrices, Cayley transform, and multidimensional orthogonal filter banks. *IEEE Transactions on Image Processing*, 15(2):511–519, 2006.

[312] P. Zhou, J. Feng, C. Ma, C. Xiong, S. Hoi, and W. E. Towards theoretically understanding why SGD generalizes better than Adam in deep learning. *Advances in Neural Information Processing Systems*, 2020.

[313] Z. Zhu, J. Wu, B. Yu, L. Wu, and J. Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. *International Conference on Machine Learning*, 2019.

[314] R. Zwanzig. Diffusion in a rough potential. *Proceedings of the National Academy of Sciences*, 87:2029–2030, 1988.