

Contributions to Efficient Machine Learning

Original

Contributions to Efficient Machine Learning / Franzese, Giulio. - (2021 Mar 01), pp. 1-155.

Availability:

This version is available at: 11583/2875759 since: 2021-03-23T09:48:16Z

Publisher:

Politecnico di Torino

Published

DOI:

Terms of use:

Altro tipo di accesso

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)



ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR



Doctoral Dissertation
Doctoral Program in Electronic and Telecommunications Engineering (33rd cycle)

Contributions to Efficient Machine Learning

Giulio Franzese

* * * * *

Supervisor

Prof Monica Visintin

Doctoral Examination Committee:

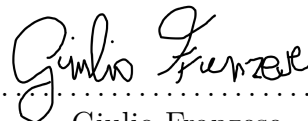
Prof. Giuseppe Caire, Referee, Technische Universitat Berlin

Prof. Maria Alejandra Zuluaga Valencia, Referee, EURECOM

Politecnico di Torino

This thesis is licensed under a Creative Commons License, Attribution - Noncommercial-NoDerivative Works 4.0 International: see www.creativecommons.org. The text may be reproduced for non-commercial purposes, provided that credit is given to the original author.

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.



.....
Giulio Franzese
Turin,

Summary

In recent years, the words machine learning and artificial intelligence have become common words in news and pop culture.

Although the field traces back its origins into the 50's of past century, the research area has seen highs and lows, the latter colloquially referred as *AI winter*. Experts and researchers in the field have partially attributed the recent renaissance to the increase of computational power that has unlocked the possibility of performing large scale experiments and consider more complex models. Recently, however, researchers has started to warn that the trend of simply throwing more computational power to be able to achieve better results is no longer sustainable both from a computational [89] and an environmental point of view [88]. In this work we focus in particular on a usually neglected aspect of the computational cost balance: the hyperparameters and hardware topology optimization loop. We do argue, in fact, that comparing efficiency of methods as, for example, training time required to reach a target accuracy is not a completely correct procedure. Actually, in fact, the training/design complexity should be computed as the cost of training a given model with a given topology times the number of different trials that are necessary to choose a good performing training scheme. This multiplicative factor, seldom reported in scientific publications, can easily be in the order of the hundreds, if not more. Experience and the access to automatic searching tools can lessen the burden of the number of different trials one has to do, but the current status is far from being solved.

We argue that an interesting direction, from a practical point of view, is the one in which we potentially sacrifice optimality of performance in favour of more efficient schemes. In the first part of the thesis, we focus on the large domain of stochastic gradient Markov Chain Montecarlo methods (SG-MCMC), a class of algorithms designed to collect samples from complex probability distributions, in which many hyperparameters need to be selected. We consider an alternative algorithm that, on a sound theoretical basis, lessens the burden of the need to find accurate schedules for learning rates. In the second part of the thesis, we explore instead the realm of classification trees and consider a novel algorithm that builds "reversed trees" based on the information theoretic concept of Information Bottleneck. The proposed algorithm is extremely simple, robust to architectural choices and built

using modular, parallelizable and easy to scale basic blocks.

Acknowledgements

And I would like to acknowledge Professor Pietro Michiardi for the helpful collaboration.

Contents

List of Tables	x
List of Figures	xI
1 Introduction	1
I Stochastic Gradient MonteCarlo integration	5
2 Introduction to Part I	7
2.1 Overview of Part I	8
2.2 Notation and Calculations (informal)	9
3 Langevin Dynamics and Fokker Planck equations	13
3.1 Wiener process, white noise, and Langevin dynamics	14
3.1.1 Stochastic integral: (informal) introduction	15
3.2 Fokker Planck equations associated to Langevin dynamics	17
3.2.1 Ito's Lemma	18
3.2.2 Infinitesimal generator	20
3.2.3 Proof of Fokker Planck Equations	22
3.3 Ito vs Stratonovich: discretisation choice, noise induced drift and conversion between the two	23
3.3.1 Equivalence conditions	27
4 Bayesian Machine Learning: the Montecarlo Integration approach	31
4.1 Overview	31
4.2 Montecarlo integration	32
4.2.1 Montecarlo Integrals are unbiased and consistent	34
4.3 Bayesian problems in Machine learning	34
4.3.1 Confidence characterization	38
4.3.2 Out of distribution detection	38
4.4 The problem of sampling from $p_{\mathbf{x} \mathcal{D}}(\hat{\mathbf{x}} \hat{\mathcal{D}})$: Markov Chain Monte Carlo	38

4.4.1	Simulated Langevin dynamics	39
4.4.2	Gradient descent with noise to sample from posterior	40
5	SG-MCMC methods	43
5.1	The computational problem	43
5.2	Stochastic Gradient	44
5.3	Gradient methods without momentum	46
5.3.1	Conditions for convergence to desired potential	47
5.3.2	Methods based on SGD	47
5.4	Gradient methods with momentum	48
5.4.1	Conditions for convergence to the desired potential (II)	49
5.4.2	Methods based on m-SGD	51
6	Isotropic SGD: Practical Bayesian Posterior Sampling	53
6.1	Multilayer Neural Networks	54
6.2	ISGD	54
6.3	An ideal method.	55
6.4	A practical method: Isotropic SGD.	56
6.4.1	Estimation of $\lambda^{(l)}$	58
6.4.2	A simple toy example	60
6.5	Assumptions and convergence to the true posterior.	61
6.6	Experimental Results	62
6.6.1	A disclaimer on performance characterization	62
6.6.2	UCI regression tasks, with a multi layer perceptron.	63
6.6.3	Classification tasks, with deeper models.	65
7	Conclusions of Part I	69
 II Deep Information Networks		 71
8	Introduction to Part II	73
8.1	Overview	74
8.2	Notations	75
9	Information Theory and Decision Trees	77
9.1	Information Theory	78
9.2	Information Bottleneck (IB)	81
9.2.1	Sufficient Statistic	82
9.2.2	Proof of IB solution	85
9.3	Inference for categorical datasets	89
9.3.1	Decision trees	90
9.3.2	Simple algorithm for growing (learning) a tree	91

9.3.3	Ensemble of trees: random forests	93
10	Deep Information Networks	97
10.1	Introduction	97
10.2	The information network	98
10.2.1	The information node	99
10.2.2	The network	100
10.2.3	The testing/running phase	103
10.3	Analysis	103
10.3.1	The probabilistic point of view	103
10.3.2	The information theory point of view	104
10.4	The Kidney Disease experiment	106
10.5	Conclusions	108
11	Probabilistic Deep Information Networks	109
11.1	The DIN architecture and its training	111
11.1.1	The input node	111
11.1.2	The information node	113
11.1.3	The combiner	114
11.1.4	The tree architecture	116
11.1.5	A note on computational complexity and memory requirements	118
11.2	The running phase	119
11.2.1	The DIN ensemble	120
11.3	The probabilistic point of view	121
11.3.1	Assumption of conditionally independent features	121
11.3.2	The overall probability matrix	122
11.4	Experiments	123
11.4.1	UCI Congressional Voting Records dataset	123
11.4.2	UCI Kidney Disease dataset	124
11.4.3	UCI Mushroom dataset	125
11.4.4	Misclassification probability analysis	125
11.4.5	The impact of number of iterations of Blahut-Arimoto on the performance	126
11.4.6	The role of β : underfitting, optimiality and overfitting	126
11.4.7	A synthetic multiclass experiment	127
11.5	Conclusions	128
12	Conclusions of Part II	131
13	General Conclusions	133
	Bibliography	135

List of Tables

6.1	RMSE results for regression on UCI data-sets.	65
6.2	MNLL results for regression on UCI data-sets	66
6.3	Results for classification on MNIST data-set.	68
10.1	Results obtained with DINs on the Kidney Disease dataset; averages over 1000 runs; $\beta = 5$, the specified value of $N_{out}^{(i)}$ is valid for $i =$ $0,1,2,3$, the last layer has $N_{out}^{(4)} = N_{classes} = 2$	107
11.1	Mean misclassification probability (over 100 random experiments) for the three datasets with the considered classifiers.	125

List of Figures

2.1	Mindmap of the thesis	10
4.1	Montecarlo estimation of π using $N_{MC} = 1000$	33
4.2	Montecarlo estimation of π performing 1000 experiments as a function of N_{MC}	35
6.1	Single hidden layer Neural Network	54
6.2	True and I-SGD predictive posterior distributions on a simple example.	61
6.3	Convergence speed of SGHMC and I-SGD for the WINE dataset. To improve readability the metrics are shifted vertically with respect to the value of the SGHMC method at time instant 0.	66
9.1	A decision tree (left). A soft decision tree (right), the events H, I are short hand for healthy and ill respectively.	91
10.1	Schematic representation of an information node, showing the input and output vectors, with alphabets of cardinality N_{in} and N_{out} , respectively, and the target vector \mathbf{y} which is available during the training phase.	99
10.2	Architecture of a simple information node network for $D = 8$: each info node is represented by a circle, the numbers inside the circle identify the node, the triangles identify the combiners, $N_{in}^{(k)}$ is the number of values taken by the input of the info node at layer k , $N_{out}^{(k)}$ is the number of values taken by the output of the info node at layer k	101
10.3	Definition of matrices, vectors and their dimensions	102
10.4	Sub-network used for the evaluation of the probability matrix; $X_{in,a}$, $X_{out,a}$, $X_{in,b}$, $X_{out,b}$, $X_{in,c}$, $X_{out,c}$ are all random variables; N_0 is the number of values taken by $X_{in,a}$ and $X_{in,b}$, N_1 is the number of values taken by $X_{out,a}$ and $X_{out,b}$, N_2 is the number of values taken by $X_{out,c}$.	104
10.5	Mutual information in the network: circles correspond to inputs of the combiners, triangles to their outputs, each dotted line represents the maximum value of the mutual information at the input of the combiners of a given layer. Results obtained during one run of the training phase with $N_{train} = 200$ (half ill, half healthy), $N_{out}^{(i)} = 3$. . .	107

11.1	Schematic representation of an input node: the inputs are two vectors, the outputs are matrices that statistically describe the random variables X_{in} and Y .	112
11.2	Schematic representation of an information node, showing the input and output matrices.	115
11.3	Sub-network: $X_{in,a}$, $X_{out,a}$, $X_{in,b}$, $X_{out,b}$, $X_{in,c}$, $X_{out,c}$ are all random variables; N_0 is the number of values taken by $X_{in,a}$ and $X_{in,b}$, N_1 is the number of values taken by $X_{out,a}$ and $X_{out,b}$, N_2 is the number of values taken by $X_{out,c}$.	116
11.4	Example of a DIN for $D = 8$: the input nodes are represented as rectangles, the info nodes as circles, the combiners as triangles. The numbers inside each circle identify the node (position inside the layer and layer number), $N_{in}^{(k)}$ is the number of values taken by the input of the info node at layer k , $N_{out}^{(k)}$ is the number of values taken by the output of the info node at layer k . In this example the info nodes at a given layer all have the same input and output cardinalities.	117
11.5	Misclassification probability versus number of iterations(average over 10 different trials) for the considered UCI datasets.	126
11.6	Misclassification probability versus β (average over 20 different trials) for the considered UCI datasets.	127
11.7	Varying of classification accuracy for different values of control parameter ρ	129

Chapter 1

Introduction

This manuscript is focused on contributions to efficient Machine Learning. We could define Machine learning as the process of computers changing the way they perform tasks of interest by learning from data without the need for a human to give direct instructions. Usually the comparison, and thus the selection, among different algorithms is performed in terms of performance for the considered task and time and complexity of computations required to reach such performance. However, comparing different methods only on such bases, does not provide the full picture of the true cost of an algorithm. In fact almost never in realistic deployment of machine learning algorithms a unique trial is performed, but several nested loops across system design choices are performed, greatly increasing the effective overall cost. Our goal in this dissertation will be the one of describing and proposing efficient algorithms, basing our exploration on sound theoretical foundations.

We will explore methods that are suited for different types of data. This term, that refers generically to any kind of information stored on a computer, is the core of all machine learning algorithms. The class of machine learning problems can be divided into two broad classes based on the kind of data on which they operate. The first class is the one in which models are built to work considering continuous inputs, that is, belonging to the space of real numbers, and models that are specific for discrete inputs. In this dissertation we consider and discuss about efficiency problems concerning both classes. We focus for the two class of methods, based on continuous and discrete data, on two distinct problems: quantification of uncertainty and hardware topology design and optimization respectively. In particular, we study the efficiency of various methods considering the difficulty of finding good hyperparameters for the first class of problems and the simplicity of implementation for the second class. The document is divided into two distinct parts, Part I and Part II respectively, that can be read independently. The theoretical foundation for the two modules is self contained in the respective parts.

The first part is focused on the analysis of methods tailored for machine learning models based on continuous inputs. This class of parametric models, of which

neural networks are currently the most famous example, currently lacks efficient, robust and scalable methodologies for the estimation of uncertainty linked to the parameters of interest. While many methods exist and have been proposed, the problem is far from being solved. In particular, one of the major drawback of current methodologies is that most of the best performing methods need careful hand tuning of important hyperparameters. Without computationally scalable and easy methodologies for uncertainty estimation, it won't be possible to extend the usage of these models to scenarios in which uncertainty estimation is of fundamental importance, such as self driving cars, aided medical diagnosis and many others. We present a new algorithm, that we call Isotropic Stochastic Gradient Descent (I-SGD), and explore carefully how the proposed method lessen the burden of finding such hyperparameters. We first present an idealised version, with theoretical guarantees, that shed light on the working principle of the proposed method and allows to draw important comparisons with respect to other concurrent methods. Then we leave the idealised world and present a practical implementation, performing an empirical analysis of the performance of the method, showing that the proposed method is competitive with the state of the art.

The second part of the thesis is concerned with the study of classification methods for discrete datasets. The most widely adopted class of algorithms to perform classification in these settings is the class of decision trees. As far as efficiency is concerned, in classical decision trees, nodes utilization depend on the single input, forcing us to make system design choices calibrated for the worst case scenario. Most of the time, a good portion of the nodes are not used, providing a clear example of inefficiency. In this second part of the thesis, we propose a novel classification algorithm, called Deep Information Networks (DIN), that is based on the Information Bottleneck principle. The considered principle is essentially a variational framework for the study of interplay between generalization and approximation from an Information theoretic perspective. We investigate a first variant of the proposed method, based on hardware capable generating discrete random variables. The main building blocks are discussed, and their possible hardware implementation is considered. We then consider a variant of the proposed method, that we call Probabilistic DIN. Probabilistic DIN extend the DIN framework considering ensembles of multiple networks and a modification of the random variable generation process. Both variants are carefully explored, providing theoretical insights and performing numerous experimental validations. Differently from classical decision trees, the total computational time and cost for all branches is fixed per input and known in advance. This knowledge, combined with the flexibility and modularity of the method, can help in taking system design choices. The experimental campaigns suggest that the considered algorithm is a good alternative to current solution and worthy of further exploration.

Both parts of the thesis are built on top of sound theoretical bases, that we

present in a didactic way by reducing, as much as possible, the previous knowledge requirements. In particular, the reader familiar with the basic knowledge of multivariate calculus and probability should be able, by the end of the thesis, to read and comprehend the formalism for the areas of Langevin Dynamics, Stochastic Integrals, Fokker Planck equations, Montecarlo Integration, Stochastic Gradient methods, Information Theory, Data compression, Information Bottleneck and Minimal Statistical Sufficiency.

Part I

Stochastic Gradient MonteCarlo integration

Chapter 2

Introduction to Part I

In recent years, complex, large scale machine learning models have seen a rapid expansion thanks to the widespread availability of performing hardware and the growing capabilities of collecting enormous datasets. In particular, the class of parametric models, that covers from the simple linear model to the extremely deep neural network, is currently used for many real world applications both in academic research and industry. The community has acknowledged as one of the biggest drawback of current deployment the unavailability of robust, easy to deploy mechanisms for uncertainty quantification at inference time of such models. This first part of the thesis is focused exactly on this problem and covers an extremely popular approach to solve such problems, namely stochastic gradient (SG) based sampling methods.

The contribution of this first part is twofold: on one side we present a novel practical algorithm (I-SGD) that allows to collect samples from desired probability distributions, and on the other we provide a complete tutorial of the stochastic gradient based sampling methods, trying to use a unified notation that is as simple as possible. As will be clear after having read the thesis, in fact, SG sampling methods are techniques that have their roots at the fringe between Stochastic calculus, being based on Stochastic Differential Equations, Statistical Physics and Chemistry, being Langevin dynamics originally a purely physical phenomenological description of certain relaxation processes, and (scalable) Optimization Theory, being the success of the methods due to the usage of modified (stochastic) gradient descent. While experts on these subjects easily fill the gaps in their knowledge and quickly become proficient when attacking the research problems linked to SG-MCMC methods, novice students can encounter some difficulties in understanding the fine differences in terms of notations and perspective that emerge across the different disciplines, such as the choice of discretisation scheme, the (unusual) role of implicit noise and many others. The humble target of this first part of the thesis is then also to provide a solid theoretical introduction to the topic, while requiring the minimal amount of background knowledge.

2.1 Overview of Part I

We start our discussion by providing a pictorial representation of the thesis by using the mindmap depicted in Figure 2.1. The four fundamental concepts treated in the various chapters can be summarised as follows:

- **Chapter 3** provides the theoretical background on Langevin Dynamics, Stochastic Integrals and Fokker Planck equations.
- **Chapter 4** treats the problem of Montecarlo integration and its important application to the computation of integrals in the Bayesian Machine Learning Framework.
- **Chapter 5** is built on top of the link drawn between Langevin Dynamics and computational efficiency of Gradient Descent. A full overview of the literature is presented and new light is shed on different tradeoffs.
- **Chapter 6** contains a new proposed sampling algorithm called Isotropic SGD, built considering the theory of Chapter 3 and competitive with many state of the art competitors.

The gluing element of all chapters is the study of Langevin Dynamics 3.1, grey in the mindmap of Figure 2.1, and their statistical properties. They can be understood as a Stochastic generalization of differential equations. Their precise definition requires particular care and the cleanest way to introduce the problem is through Stochastic Integrals 3.1.1. The definition of Stochastic Integrals allows to study the Fokker Planck equations, 3.2,3.2.3, that describe the time evolution of the probability density functions of the Langevin Dynamics. The chapter is written such that it is self contained and any reader with basic knowledge of probability theory and signal processing can understand it. Of particular interest for the reader whose background is a discipline different from Machine Learning such as physics or chemistry, is section 3.3. The focus is on the difference between Ito discretisation (the one used in this thesis) and Stratonovich one (the one common for example in physics). Using the two conventions provides different results and one cannot blindly apply approaches derived with one discretisation and apply it to the other without incurring in errors. Finally section 3.3.1 underlines some particular cases of Stochastic Differential Equations that will be useful for the rest of the thesis.

The motivating problem of the thesis is presented in Section 4.3, where the Bayesian approach to Machine Learning is described. The main idea is to consider parametric machine learning models and the Bayesian a posteriori probability of the parameters given an observation dataset to enhance the uncertainty quantification capabilities of the considered models with respect to the classical ML approach. While intriguing and elegant, the main drawback of this approach is the (usual) analytical insolvability of probability integrals of interest. In Section 4.2,

Montecarlo integration techniques are introduced, whose main idea is to estimate, instead of exactly computing, integrals by averaging functions over samples drawn from particular distributions. Section 4.4 explains how to get samples to be used for the aforementioned technique, and the particular case of algorithm based on Langevin Dynamics is introduced.

Langevin dynamics are defined as a continuous time random processes, thus any attempt of exact simulation on a digital computer must rely on some form of discretisation. Interestingly, it turns out that for the subclass of dynamics of interest there is a strong duality between the Euler-Maryama discretisation scheme for Stochastic Differential Equations and a variant of Gradient Descent Algorithm. In an infinite computational power world, a straightforward modification of gradient descent would allow to solve the problem on a sound theoretical basis. However, obviously, since scalability is an important practical issue with modern large scale problems we describe (Section 5.2) variants of the discretisation scheme based on mini-batch (i.e. random subsampling of the whole dataset) gradient methods, called Stochastic Gradient Descent SGD. Sections 5.3 and 5.4 contain an exhaustive overview of the methods based on variants of SGD and SGD with momentum. Thanks to our unified notation it is possible to fully appreciate the various assumptions and tradeoffs of the various methods by investigating the link between noise, learning rates and preconditioning matrices.

The whole Chapter 6 is devoted to Isotropic SGD, a simple and scalable variant of SGD that we propose. Our main objective is to derive a simple, scalable and easy to tune algorithm. We describe an idealised version of the proposed algorithm in Section 6.3. By leveraging the theory described in Chapter 3 it is possible to prove that the proposed algorithm guarantees to obtain samples from the true posterior. We then carefully analyse a practical version in Section 6.4, based on a well grounded set of assumptions. Finally, a large experimental campaign comparing the proposed method and many competitors is presented in Section 6.6.

Conclusions are presented in Chapter 7, where the summary of the whole work is presented.

2.2 Notation and Calculations (informal)

In this brief section we introduce the necessary notations and calculations.

We will often work with systems of equations, that we describe using matricial notation. We indicate scalars with lowercase symbols, e.g. a , vectors (always inteded column) with bold lowercase symbols, \mathbf{a} . When considering the single j th element of the vector \mathbf{a} , we use the notation $(\mathbf{a})_j = a_j$. Transposition is indicated with the symbol \top . Matrices are indicated with bold, uppercase letters, \mathbf{A} .

We indicate with p the *probability density function* (p.d.f.), of a real random variable x , i.e. $x \sim p_x(\hat{x})$. Some text refer to the random density of the random

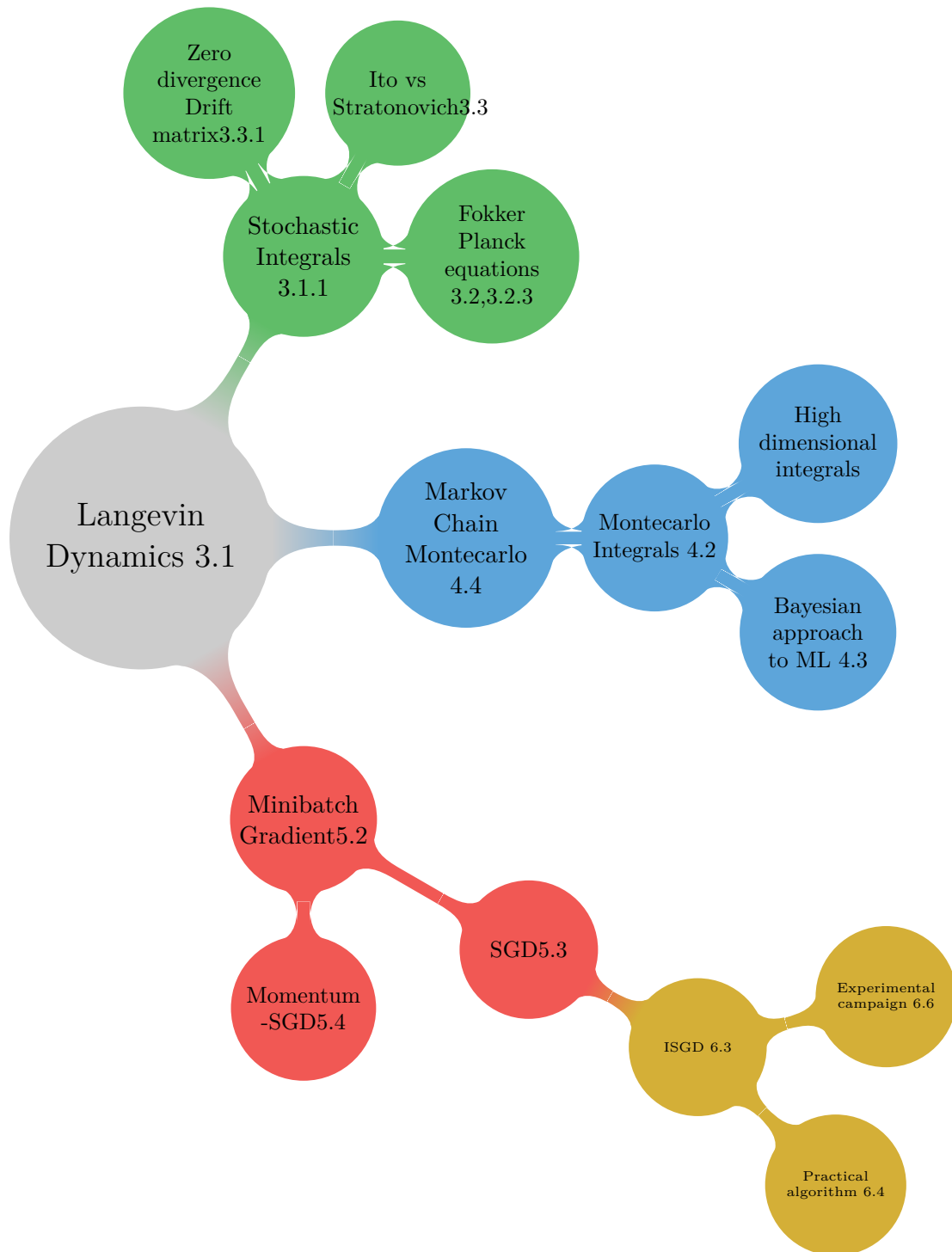


Figure 2.1: Mindmap of the thesis

variable x simply as $p(x)$. While this is a commonly adopted choice in many disciplines, we do choose the extended notation to avoid ambiguities in distinguishing between the random variable and a value assumed by the variable. To help the reader throughout the text, we usually choose as the arbitrary variable inside the density function the same symbol as the corresponding random variable with an hat ($\hat{\cdot}$) symbol on top.

When dealing with random quantities, we indicate the expected value using the symbol $E(\cdot)$. If the considered quantity is multivariate, i.e. the object is determined by multiple random variables, and there is the need to average out only some of the components, we append the subscript corresponding to the averaged out variable. We make the concept clearer with an example

$$\begin{aligned} \mathbf{z} &= [x, y] \sim p_{x,y}(\hat{x}, \hat{y}) \\ y &= E_x(\mathbf{z}) \sim \int p_{x,y}(\hat{x}, \hat{y}) d\hat{x}. \end{aligned}$$

We indicate the multivariate Gaussian probability density function using the following notation

$$\log(p(\hat{\mathbf{x}})) = -\frac{1}{2} \left(\log(|\text{Det}(2\pi\mathbf{\Sigma})|) + (\hat{\mathbf{x}} - \mu)^\top \mathbf{\Sigma}^{-1} (\hat{\mathbf{x}} - \mu) \right) \iff \mathbf{x} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$$

The indicator function $\mathbb{1}(\cdot)$ is a non differentiable, piecewise constant function that maps events into $\{0,1\}$. It takes value 1 if the considered event is verified and 0 otherwise. For example

$$\mathbb{1}(\exp(x) < 1) = \begin{cases} 1 & \text{if } \exp(x) < 1 \\ 0 & \text{if } \exp(x) \geq 1 \end{cases} = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x \geq 0 \end{cases}$$

The Dirac delta is a generalized function. If one desires to be formally completely correct, the Dirac delta is an object that can be used (and makes sense) only inside an integral. In the general literature of signal processing and physics however the usage is relaxed and it is considered as a valid object even when existing alone. There are several different ways of defining the Dirac delta as limit of classical functions, two of the most common being

$$\delta(x) = \lim_{h \rightarrow 0} \frac{1}{\sqrt{\pi|h|}} \exp\left(-\frac{x^2}{h^2}\right)$$

and

$$\delta(x) = \lim_{h \rightarrow 0} \frac{1}{h} \mathbb{1}\left(-\frac{h}{2} \leq x \leq \frac{h}{2}\right)$$

The multidimensional Dirac delta can be defined as $\delta(\mathbf{x}) = \prod_i \delta(x_i)$.

We often deal with time varying quantities (signals). We distinguish between continuous time signals, i.e. signals whose domain is the real axis, and discrete time ones, signals whose domain is the set of integers. We indicate continuous time signal arguments with the classical round parentheses as $x(t)$, while discrete time ones using the square brackets as $x[n]$.

Throughout the work we make heavy use of differential operators. In the following we define their properties and some useful equalities when dealing with linear algebra-like operations. Notice that this presentation is carried at an informal level, it is assumed that the reader is robustly familiar with all the described concepts and thus the main purpose of the chapter is to set a common notation to lighten the reading of the thesis. It is not then, by any means, a complete and correct presentation, and the interested reader is referred to [12] for a complete and precise discussion.

Consider an open subset of \mathbb{R}^N . The partial differential operator $\frac{\partial}{\partial x_i}$, sometimes simply ∂_i , maps every smooth function $f(\cdot)$, in the usual sense of partial derivative $\frac{\partial f(\mathbf{x})}{\partial x_i}$, i.e. $\frac{\partial}{\partial x_i} : f \rightarrow \mathbb{R}$.

The ∇ operator is simply an $N \times 1$ vector containing all partial derivatives, i.e.

$$\nabla = [\partial_0, \partial_1, \dots, \partial_{N-1}]^\top \quad (2.1)$$

When applied to a function, it produces a vector field

$$\nabla f(\mathbf{x}) = [\partial_0 f, \partial_1 f, \dots, \partial_{N-1} f]^\top \quad (2.2)$$

The first important property we notice is that the ∇ operator is not commutative, $\nabla f \neq f \nabla$.

When applied to a multivariate function $\mathbf{a}(\mathbf{x}) \in \mathbb{R}^N$, the linear algebra rules for the dimensions must be respected, i.e. $\nabla \mathbf{a}^\top$ is valid, $\nabla^\top \mathbf{a}$ is valid, while $\nabla \mathbf{a}$, $\nabla^\top \mathbf{a}^\top$ are not. The explicit form can be expressed as

$$(\nabla \mathbf{a}^\top)_{i,j} = \partial_i a_j, \quad \nabla^\top \mathbf{a} = \sum_i \partial_i a_i. \quad (2.3)$$

We define moreover the Δ operator as $\Delta_{i,j} = \partial_i \partial_j$, or in vectorial form $\Delta = \nabla \nabla^\top$. All the properties needed in this thesis during calculations can be obtained as particular cases of the following following quantity: $\nabla^\top (f \mathbf{M} \mathbf{a})$, where $\mathbf{M} \in \mathbb{R}^{N \times M}$, $\mathbf{a} \in \mathbb{R}^M$, $f \in \mathbb{R}$, from which other particular cases are easily derived:

$$\begin{aligned} \nabla^\top (f \mathbf{M} \mathbf{a}) &= \sum_{i,j} \partial_i (f M_{i,j} a_j) = \sum_{i,j} \partial_i (f) M_{i,j} a_j + f \partial_i (M_{i,j}) a_j + f M_{i,j} \partial_i (a_j) = \\ &\nabla^\top (f) \mathbf{M} \mathbf{a} + f \nabla^\top (\mathbf{M}) \mathbf{a} + f \text{Tr}(\mathbf{M}^\top \nabla \mathbf{a}^\top) \end{aligned}$$

Finally, whenever the quantities a and b belong to a space \mathcal{S} where scalar product is defined, we indicate their scalar product as $\langle a, b \rangle$.

Chapter 3

Langevin Dynamics and Fokker Planck equations

In this chapter, we build the mathematical tools that are needed to understand the basic idea behind stochastic dynamics simulations for Montecarlo integration. To generate samples from a given probability distribution, simulate a stochastic system whose stationary distribution is the desired probability, collect samples in the stationary regime and use them to compute the Montecarlo integrals of interest. The most important mathematical tool we will develop in this chapter is the Fokker Planck equation (FPE). Named after Adriaan Fokker [24] and Max Planck [68], it is a partial differential equation that describes the time evolution of the first order probability distribution for a large class of random processes. Having its roots in physics, it describes the velocity of a particle under the influence of drag forces and stochastic forces, but can be easily conceptually generalized to different settings.

We introduce in Section 3.1 the concepts of Wiener process, white noise and Langevin dynamics, the central topic of the thesis. Being stochastic generalizations of classical differential equations, their formal definition is tricky. We propose to introduce the reader to the general topic through Stochastic Integrals, described in Section 3.1.1. Having set the roots, we can expand the discussion and present the mathematical workhorse of the thesis, the Fokker Planck equations, Sections 3.2, 3.2.3, that allow to study the time evolution of the probability density functions of the Langevin Dynamics. Since the target audience of this thesis is broader than the machine learning community, we include a whole section dedicated to the technicalities linked to different discretisation schemes that appear in different scientific fields such as physics or chemistry. Indeed, in section 3.3, the focus is on the difference between Ito discretisation (the one used in this thesis) and Stratonovich one (the one common for example in physics). It is not possible, besides from special cases, to blindly convert between the two notations without incurring in some form of error, and thus a precise set of conversion rules are

presented. Finally, in Section 3.3.1, we present some of the particular cases of Stochastic Differential Equations that will be useful for the rest of the thesis.

3.1 Wiener process, white noise, and Langevin dynamics

The Wiener process [22] is the stochastic process $\mathbf{w}(t)$ that satisfies the following properties:

- $\mathbf{w}(0) = \mathbf{0}$
- $\mathbf{w}(t)$ is composed of independent increments. The increment of the stochastic process from $t \geq 0$ to $t + u \geq t$, i.e. $\delta\mathbf{w} = \mathbf{w}(t + u) - \mathbf{w}(t)$ is statistically independent from all the past of $\mathbf{w}(t)$. Formally, the following property holds:

$$p_{\mathbf{w}(t+u)-\mathbf{w}(t)|\mathbf{w}(s)}(\hat{\mathbf{w}}_1|\hat{\mathbf{w}}_0) = p_{\mathbf{w}(t+u)-\mathbf{w}(t)}(\hat{\mathbf{w}}_1) \quad \forall t \geq 0, u \geq 0, s < t \quad (3.1)$$

- $\mathbf{w}(t)$ is composed of Gaussian increments. In particular $\mathbf{w}(t + u) - \mathbf{w}(t) \sim \mathcal{N}(0, u\mathbf{I})$

Of particular interest are the infinitesimal increments $d\mathbf{w}(t) = \mathbf{w}(t + dt) - \mathbf{w}(t)$. Considering the aforementioned properties, it is easy to show that $d\mathbf{w}(t) \sim \mathcal{N}(0, dt\mathbf{I})$. Some readers, whose background is stronger in engineering or physics than mathematics, will be more familiar with the concept of white noise instead of Wiener process. Formally it is possible to obtain a Gaussian white noise process $\mathbf{n}(t)$ as

$$\mathbf{n}(t) = \frac{d\mathbf{w}(t)}{dt} \quad (3.2)$$

A Gaussian process $\mathbf{n}(t)$ is defined as white if its autocorrelation function is the Dirac delta,

$$E[\mathbf{n}(t)\mathbf{n}^\top(t + \tau)] = \mathbf{I}\delta(\tau). \quad (3.3)$$

From (3.2) we notice that $\mathbf{n}(t) \sim \mathcal{N}(0, \frac{1}{dt}\mathbf{I})$. Moreover, due to the independence of increments property, we have that, for any $\tau > 0$, $\mathbf{n}(t + \tau)$ and $\mathbf{n}(t)$ are independent. Combining together we can write that

$$E[\mathbf{n}(t)\mathbf{n}^\top(t + \tau)] = \begin{cases} \frac{1}{dt}\mathbf{I} & \text{if } \tau = 0 \\ \mathbf{0} & \text{if } \tau \neq 0 \end{cases} \quad (3.4)$$

At least informally, we can then state the equivalence between (3.3) and (3.4). In fact, one of the possible representations of the Dirac delta is

$$\delta(z) = \lim_{h \rightarrow 0} \frac{1}{h} \mathbf{1} \left(-\frac{h}{2} \leq z \leq \frac{h}{2} \right) \quad (3.5)$$

from which the informal equivalence is evident. In the applied mathematics literature the Wiener process formalism is preferred to the white noise one, but often in other fields of engineering or physics both approaches are considered, depending on which object is simpler to treat [45].

Having introduced the concept of Wiener process, we are able to define the concept of Langevin dynamic. It is best understood using physical analogies. Consider a system whose internal state at time instant t can be described by an N -dimensional random variable $\mathbf{x}(t)$. We say that the time evolution of the random variable $\mathbf{x}(t)$ is ruled by an over-damped Langevin dynamic if the infinitesimal increments of system state, $d\mathbf{x}(t) = \mathbf{x}(t+dt) - \mathbf{x}(t)$, are described by the following stochastic differential equation (Ito's convention, more on that in section 3.1.1)

$$d\mathbf{x}(t) = \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t))d\mathbf{w}(t) \quad (3.6)$$

where $\mathbf{s}(\cdot) : \mathbb{R}^N \rightarrow \mathbb{R}^N$ is the so called drift or driving force, while $\mathbf{D}(\cdot) : \mathbb{R}^N \rightarrow \mathbb{R}^{N \times M}$ is called diffusion matrix. The quantity $\mathbf{w}(t)$ is an M -dimensional Wiener process. The equation can be understood as a generalization of a classical deterministic differential equation $\frac{d\mathbf{x}(t)}{dt} = \mathbf{s}(\mathbf{x}(t))$ for the time evolution of a physical system, to the case where external, random forces acts as well. The role of random forces is taken by the Wiener process and the diffusion matrix $\mathbf{D}(\mathbf{x}(t))$ describes the (possible) dependence of the random forces strength and correlations on the system internal state $\mathbf{x}(t)$.

3.1.1 Stochastic integral: (informal) introduction

Introducing correctly from a formal point of view the formulation of an SDE (Stochastic Differential Equation) is a topic worth by itself a dissertation. In the spirit of practicality, we simply underline that one of the key elements is the definition of the stochastic integral. discretisation, or the infinitesimal time grid spacing that is selected to build the SDE (and the corresponding integral), has an impact on the result. Starting from a generic SDE without discretisation prescription specified (also called pre-equation)

$$d\mathbf{x} = \mathbf{s}dt + \sqrt{2}\mathbf{D}d\mathbf{w} \quad (3.7)$$

we can naively write the corresponding integral equation as

$$\mathbf{x}(T) - \mathbf{x}(0) = \int_0^T \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2} \int_0^T \mathbf{D}(\mathbf{x}(t))d\mathbf{w}(t) \quad (3.8)$$

and consider the matter solved. Actually, we are missing an extremely important piece of information: how to compute $\int_0^T \mathbf{D}(\mathbf{x}(t))d\mathbf{w}(t)$?

The problem is better explained in terms of the classical Riemann-Stieltjes integral (1-d for simplicity of exposition). Consider two functions f, g , defined on the interval $[a, b]$, and define the integral

$$\int_a^b f(t)dg(t) = \lim_{\|\mathcal{P}\| \rightarrow 0} \sum_{i=0}^{n-1} f(\tau_i)(g(t_i) - g(t_{i-1})), \quad (3.9)$$

where $\mathcal{P} = \{t_0 = a, t_1, \dots, t_n = b\}$, $t_{i+1} > t_i$, $\|\mathcal{P}\| = \max_i |t_{i+1} - t_i|$ and $\tau_i \in [t_i, t_{i+1}]$. A fundamental result [46] states that if f, g have **bounded variation**¹ then the integral exists and does not depend on the choice of the τ_i . When computing $\int \mathbf{D}(\mathbf{x}(t))d\mathbf{w}(t)$, however, the quantity $\mathbf{w}(t)$ does not have bounded variation. The idea is that if a function has bounded variation, then it is everywhere differentiable. Since the Wiener process $\mathbf{w}(t)$ (whose derivative is white noise) is almost surely nowhere differentiable, then $\mathbf{w}(t)$ does not have bounded variation. This is fundamentally, the reason why the choice of the τ_i influences the result.

Implicitly and without being rigorous, when we introduced Langevin dynamics in equation (3.6) we chose the following discretisation scheme:

$$t_n = ndt \quad \tau_n = t_n. \quad (3.10)$$

This choice, called Ito discretisation, is however, up to this point, arbitrary, and different choices with different resulting dynamics are possible.

This apparent paradox and the problem of choosing different discretisation schemes starting from a mathematical model, have been the subject of an intense academical discussion [84],[72]. In the realm of pure mathematics the problem is *unsolved*, since there is no reason to favour one definition against others, as long as the built stochastic calculus is consistent. When dealing with real modelling problems, in applied sciences, the consensus is that depending on the considered scenario, some discretisations are more natural than others.

Luckily for us, we do not need to consider this problem: the discretisation scheme is in fact imposed by the nature of gradient algorithms considered in this thesis. With respect to a physical, chemical or economical modelling problem in which the correct discretisation scheme has to be chosen to avoid producing wrong results, we here face the backward problem: we start from a discretisation scheme (corresponding to gradient descent and variants) and describe its properties using the continuous time limit. It is important however, to take particular care when manipulating the quantities of interest, since in the considered formalism unintuitive rules of calculus hold (the classical chain rule is not valid).

¹The total variation of a function f is defined as $TV(f) = \sup_{\mathcal{P}} \sum_{i=0}^{n-1} |f(t_{i+1}) - f(t_i)|$, a function is said to have bounded variation if its total variation is finite.

3.2 Fokker Planck equations associated to Langevin dynamics

The knowledge of the probability density function distribution $p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t)$ for the random process $\mathbf{x}(t)$ governed by the a Langevin dynamic is obviously of paramount importance². There are multiple possibilities for deriving the considered probability distribution, from path integral formulations to particular techniques for simple systems, such as ones driven by linear forces and whose diffusion is state independent. In this work, we do consider the Fokker Planck formalism. Our aim is to prove that for a stochastic system ruled by an equation of the form (3.6), the Fokker Planck equation

$$\frac{\partial p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t)}{\partial t} = \text{Tr} \left\{ \nabla \left[-\mathbf{s}(\hat{\mathbf{x}})^\top p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t) + \nabla^\top \left(\boldsymbol{\Sigma}(\hat{\mathbf{x}}) p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t) \right) \right] \right\} \quad (3.11)$$

describes the time evolution for the probability distribution of system state, where $\boldsymbol{\Sigma}(\hat{\mathbf{x}}) = \mathbf{D}(\hat{\mathbf{x}})\mathbf{D}(\hat{\mathbf{x}})^\top$. We will give particular attention to the stationary distribution, that is $\rho_{ss}(\hat{\mathbf{x}}) = \lim_{t \rightarrow \infty} p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t)$, that satisfies the following:

$$0 = \text{Tr} \left\{ \nabla \left[-\mathbf{s}(\hat{\mathbf{x}})^\top \rho_{ss}(\hat{\mathbf{x}}) + \nabla^\top \left(\boldsymbol{\Sigma}(\hat{\mathbf{x}}) \rho_{ss}(\hat{\mathbf{x}}) \right) \right] \right\}. \quad (3.12)$$

An extremely important property that will be useful in next chapters is that an SDE with $\mathbf{s}(\hat{\mathbf{x}}) = -\nabla f(\hat{\mathbf{x}})$ and $\boldsymbol{\Sigma}(\hat{\mathbf{x}}) = \mathbf{I}$, has stationary distribution that satisfies:

$$\rho_{ss}(\hat{\mathbf{x}}) \propto \exp(-f(\hat{\mathbf{x}})). \quad (3.13)$$

as is easily checked by substitution in (3.12):

$$\begin{aligned} 0 &= \text{Tr} \left\{ \nabla \left[\nabla^\top f(\hat{\mathbf{x}}) \rho_{ss}(\hat{\mathbf{x}}) + \nabla^\top (\rho_{ss}(\hat{\mathbf{x}})) \right] \right\} \\ 0 &= \text{Tr} \left\{ \nabla \left[\nabla^\top f(\hat{\mathbf{x}}) \exp(-f(\hat{\mathbf{x}})) + \nabla^\top (\exp(-f(\hat{\mathbf{x}}))) \right] \right\} \\ 0 &= \text{Tr} \left\{ \nabla \left[\nabla^\top f(\hat{\mathbf{x}}) \exp(-f(\hat{\mathbf{x}})) - \exp(-f(\hat{\mathbf{x}})) \nabla^\top f(\hat{\mathbf{x}}) \right] \right\} \\ 0 &= 0 \end{aligned}$$

Similarly, if $\mathbf{s}(\hat{\mathbf{x}}) = \nabla f(\hat{\mathbf{x}})$, then

$$\rho_{ss}(\hat{\mathbf{x}}) \propto \exp(f(\hat{\mathbf{x}})). \quad (3.14)$$

²With a slight abuse of notation we write the density function as $p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t)$ instead of simply $p_{\mathbf{x}(t)}(\hat{\mathbf{x}})$. The time variable t is not random and thus is not necessary to write it twice. However, we choose this convention to be more uniform w.r.t. the literature on applied Fokker Planck equations.

3.2.1 Ito's Lemma

Starting from a generic Langevin dynamic (3.6):

$$d\mathbf{x}(t) = \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t))d\mathbf{w}(t),$$

the stochastic process $\mathbf{x}(t)$ is implicitly defined. The first fundamental type of investigation we need to perform in order to characterize and understand this class of processes is the study of the behaviour of a generic function $h : \mathbb{R}^N \rightarrow \mathbb{R}$ applied to the stochastic process $\mathbf{x}(t)$. The study of this transformation will be instrumental to understand the new rules of calculus that are valid when using the Ito prescription for SDE and later used as building tool for the proof of the Fokker Planck equation.

A naive application of chain rule would suggest the following:

$$dh(t) = \nabla_{\hat{\mathbf{x}}}^\top h(\hat{\mathbf{x}})|_{\hat{\mathbf{x}}=\mathbf{x}(t)}d\mathbf{x}(t) = \nabla^\top h(\mathbf{x}(t))d\mathbf{x}(t), \quad (3.15)$$

where the second equality is used to introduce a simpler abuse of notation. It turns out that (3.15) is **wrong**, and needs to be adjusted with second order derivatives. As discussed previously, due to the unbounded nature of Wiener processes, strange phenomena can happen. The correct equality is usually called Ito's Lemma [44], and describes the main difference between ordinary and (Ito) stochastic calculus.

We start by considering the differential of the new function

$$dh(t) = h(\mathbf{x}(t + dt)) - h(\mathbf{x}(t)) \quad (3.16)$$

The first term on the right of (3.16) can be expanded as

$$\begin{aligned} h(\mathbf{x}(t + dt)) &= h(\mathbf{x}(t) + d\mathbf{x}(t)) = h(\mathbf{x}(t)) + \nabla^\top h(\mathbf{x}(t))d\mathbf{x}(t) + \\ &\frac{1}{2}d\mathbf{x}(t)^\top \Delta h(\mathbf{x}(t))d\mathbf{x}(t) + \dots \end{aligned} \quad (3.17)$$

and consequently

$$dh(t) = \nabla^\top h(\mathbf{x}(t))d\mathbf{x}(t) + \frac{1}{2}d\mathbf{x}(t)^\top \Delta h(\mathbf{x}(t))d\mathbf{x}(t) + \dots \quad (3.18)$$

To avoid clutter, in this section, whenever unambiguous, we remove the explicit dependencies of the considered quantities (for example $d\mathbf{x}(t) \rightarrow d\mathbf{x}$ or $\Delta h(\mathbf{x}(t)) \rightarrow \Delta h$).

Since we are working with infinitesimals, and from (3.6) it is clear that $d\mathbf{x}$ is of order $\mathcal{O}(dt)$, normally one would truncate (3.18) at the first term. We show instead that, counterintuitively, the second term is a quantity of order $\mathcal{O}(dt)$ and needs thus to be kept. Focusing on the second term, in fact

$$\begin{aligned} d\mathbf{x}^\top \Delta h d\mathbf{x} &= \text{Tr} \left(\Delta h d\mathbf{x} d\mathbf{x}^\top \right) = \\ &\text{Tr}(\Delta h s s^\top dt^2) + 2\sqrt{2}\text{Tr}(\Delta h s d\mathbf{w}^\top \mathbf{D}^\top dt) + 2\text{Tr}(\Delta h \mathbf{D} d\mathbf{w} d\mathbf{w}^\top \mathbf{D}^\top) \end{aligned} \quad (3.19)$$

The first term, $\text{Tr}(\Delta h \mathbf{s} \mathbf{s}^\top dt^2) = \mathbf{s}^\top \Delta h \mathbf{s} dt^2$, is a quantity of order $\mathcal{O}(dt^2)$ and can thus be safely neglected. The second term, $\text{Tr}(\Delta h \mathbf{s} \mathbf{d}\mathbf{w}^\top \mathbf{D}^\top dt)$, is a zero mean Gaussian random variable with variance $\mathcal{O}(dt^3)$ (due to $\mathbf{d}\mathbf{w} dt$), and thus in the order dt any fluctuation from the expected value can be neglected, considering de facto the term equal to zero. Finally the last term, $\text{Tr}(\Delta h \mathbf{D} \mathbf{d}\mathbf{w} \mathbf{d}\mathbf{w}^\top \mathbf{D}^\top)$, is instead surprisingly of order $\mathcal{O}(dt)$.

Consider in fact the random matrix \mathbf{M} , $M_{i,j} = dw_i dw_j$. When $i \neq j$ it is easy to show that $E[M_{i,j}] = 0$, $\text{var}(M_{i,j}) = dt^2$. This implies that $M_{i,j}$ is a zero mean random variable with variance quadratic in dt . Since dt is an infinitesimal, and we are considering perturbation up to order $\mathcal{O}(dt)$ we can safely consider to be deterministically determined as $M_{i,j} = 0$. When $i = j$, instead the expected value is $E[M_{i,i}] = dt$ and the variance is $\text{var}(M_{i,i}) = E[(dw_i^2)^2] - E[(dw_i^2)]^2 = 3dt^2 - dt^2 = 2dt^2$. Again, we can interpret $M_{i,i}$ as the deterministic quantity dt perturbed by a random variable whose variance is order dt^2 (thus the perturbation is negligible). Loosely speaking we thus have the following equality: $M_{i,j} = \delta_{i,j} dt$. Consequently, we can substitute in the third term of (3.19) $\mathbf{d}\mathbf{w} \mathbf{d}\mathbf{w}^\top = dt \mathbf{I}$, and obtain $dt \text{Tr}(\Delta h \mathbf{D} \mathbf{D}^\top) = dt \text{Tr}(\Delta h \mathbf{\Sigma})$. The full differential is then

$$dh = \nabla^\top h \mathbf{d}\mathbf{x} + \text{Tr}(\Delta h \mathbf{\Sigma}) dt$$

and since $\mathbf{d}\mathbf{x} = \mathbf{s} dt + \sqrt{2} \mathbf{D} \mathbf{d}\mathbf{w}$, we can write

$$dh = \nabla^\top h \mathbf{s} dt + \sqrt{2} \nabla^\top h \mathbf{D} \mathbf{d}\mathbf{w} + \text{Tr}(\Delta h \mathbf{\Sigma}) dt$$

Rearranging, and reintroducing the explicit dependencies, we have

$$\begin{aligned} dh(t) &= \left(\nabla^\top h(\mathbf{x}(t)) \mathbf{s}(\mathbf{x}(t)) + \text{Tr}(\Delta h(\mathbf{x}(t)) \mathbf{\Sigma}(\mathbf{x}(t))) \right) dt \\ &+ \sqrt{2} \nabla^\top h(\mathbf{x}(t)) \mathbf{D}(\mathbf{x}(t)) \mathbf{d}\mathbf{w}(t). \end{aligned} \quad (3.20)$$

The result (3.20) is usually called Ito's Lemma. The expected value of $dh(t)$ conditioned on $\mathbf{x}(t) = \hat{\mathbf{x}}$ is

$$E[dh(t) | \mathbf{x}(t) = \hat{\mathbf{x}}] = \left(\nabla^\top h(\hat{\mathbf{x}}) \mathbf{s}(\hat{\mathbf{x}}) + \text{Tr}(\Delta h(\hat{\mathbf{x}}) \mathbf{\Sigma}(\hat{\mathbf{x}})) \right) dt \quad (3.21)$$

Notice that other conventions for SDE are possible, such as the Stratonovich one [72], for which different calculus rules hold. On one side, the Ito's convention explicitly describes the time evolution of the stochastic process and many results are easily proven thanks to the Markov property of the SDE. On the other side, the Stratonovich one has a more natural interpretation in physical applications and the usual rule of calculus are valid. In this work, we present results in terms of Ito's convention, mainly due to this discretisation scheme and the classical gradient descent training methods used in machine learning. However, as we will show in Section 3.3, it is always possible to translate between the two conventions. This is useful, in particular, in understanding peculiar effects such as noise induced drifts.

3.2.2 Infinitesimal generator

In the theory of stochastic processes, the quantity described by (3.21) is linked to the so called infinitesimal generator. Many important properties of the stochastic system can be described having just the knowledge of the infinitesimal generator, and it is thus of fundamental importance.

The infinitesimal generator is an operator \mathcal{A} that acts on continuous, vanishing at infinity set of functions $h : \mathbb{R}^N \rightarrow \mathbb{R}$, as follows

$$\mathcal{A}h(\hat{\mathbf{x}}) = \lim_{\delta t \rightarrow 0} \frac{E(h(\mathbf{x}(\delta t)) | \mathbf{x}(0) = \hat{\mathbf{x}}) - h(\hat{\mathbf{x}})}{\delta t} \quad (3.22)$$

If we restrict ourselves to Markov processes, we can rewrite the previous definition as

$$\mathcal{A}h(\hat{\mathbf{x}}) = \lim_{\delta t \rightarrow 0} \frac{E(h(\mathbf{x}(t + \delta t)) | \mathbf{x}(t) = \hat{\mathbf{x}}) - h(\hat{\mathbf{x}})}{\delta t}. \quad (3.23)$$

We recast the definition of infinitesimal generator into the following equivalent, but easier to be treated, form:

$$\lim_{\delta t \rightarrow 0} [\delta t \mathcal{A}h(\hat{\mathbf{x}}) - (E(h(\mathbf{x}(t + \delta t)) | \mathbf{x}(t) = \hat{\mathbf{x}}) - h(\hat{\mathbf{x}}))] = 0, \quad (3.24)$$

and since $h(\hat{\mathbf{x}}) = E(h(\mathbf{x}(t)) | \mathbf{x}(t) = \hat{\mathbf{x}})$, we can further manipulate the expression as

$$\lim_{\delta t \rightarrow 0} [\delta t \mathcal{A}h(\hat{\mathbf{x}}) - E(h(\mathbf{x}(t + \delta t)) - h(\mathbf{x}(t)) | \mathbf{x}(t) = \hat{\mathbf{x}})] = 0. \quad (3.25)$$

We recognize, in the second term of (3.25), the expected value of the differential described by (3.21). We then write, by substituting $\delta t \rightarrow dt$,

$$\begin{aligned} dt \mathcal{A}h(\hat{\mathbf{x}}) - E(dh(\mathbf{x}(t)) | \mathbf{x}(t) = \hat{\mathbf{x}}) = \\ dt \mathcal{A}h(\hat{\mathbf{x}}) - dt \left(\nabla^\top h(\hat{\mathbf{x}}) \mathbf{s}(\hat{\mathbf{x}}) + \text{Tr}(\Delta h(\hat{\mathbf{x}}) \Sigma(\hat{\mathbf{x}})) \right) = 0. \end{aligned}$$

We thus derive that

$$\mathcal{A}h(\hat{\mathbf{x}}) = \left(\nabla^\top h(\hat{\mathbf{x}}) \mathbf{s}(\hat{\mathbf{x}}) + \text{Tr}(\Delta h(\hat{\mathbf{x}}) \Sigma(\hat{\mathbf{x}})) \right), \quad (3.26)$$

Notice that \mathcal{A} is clearly a linear (in the function h) differential operator:

$$\mathcal{A}h(\hat{\mathbf{x}}) = \sum_i s_i(\hat{\mathbf{x}}) \frac{\partial}{\partial \hat{x}_i} h(\hat{\mathbf{x}}) + \sum_{i,j} \Sigma_{i,j}(\hat{\mathbf{x}}) \frac{\partial^2}{\partial \hat{x}_i \partial \hat{x}_j} h(\hat{\mathbf{x}}) \quad (3.27)$$

Having defined the infinitesimal generator \mathcal{A} , it is possible to derive its adjoint, i.e, the operator \mathcal{A}^\dagger acting on the same set of functions for which $\langle \mathcal{A}h(\hat{\mathbf{x}}), g(\hat{\mathbf{x}}) \rangle = \langle \mathcal{A}^\dagger g(\hat{\mathbf{x}}), h(\hat{\mathbf{x}}) \rangle$, where $g(\hat{\mathbf{x}})$ is again in the set of continuous functions vanishing

at infinity. The adjoint of a linear differential \mathcal{A}^\dagger operator is linear in the various components of \mathcal{A} , i.e. if $\mathcal{A} = \sum_k \mathcal{A}_{(k)}$, then $\mathcal{A}^\dagger = \sum_k \mathcal{A}_{(k)}^\dagger$. In fact

$$\begin{aligned} \langle \mathcal{A}h(\hat{\mathbf{x}}), g(\hat{\mathbf{x}}) \rangle &= \langle \sum_k \mathcal{A}_{(k)}h(\hat{\mathbf{x}}), g(\hat{\mathbf{x}}) \rangle = \sum_k \langle \mathcal{A}_{(k)}h(\hat{\mathbf{x}}), g(\hat{\mathbf{x}}) \rangle = \\ &= \sum_k \langle \mathcal{A}_{(k)}^\dagger g(\hat{\mathbf{x}}), h(\hat{\mathbf{x}}) \rangle = \langle \sum_k \mathcal{A}_{(k)}^\dagger g(\hat{\mathbf{x}}), h(\hat{\mathbf{x}}) \rangle = \langle \mathcal{A}^\dagger g(\hat{\mathbf{x}}), h(\hat{\mathbf{x}}) \rangle, \end{aligned}$$

and clearly from the last equality $\mathcal{A}^\dagger = \sum_k \mathcal{A}_{(k)}^\dagger$. To find the adjoint of a generic operator, it is sufficient then to find the adjoint of the various components. Consider

$$\mathcal{D}h(\hat{\mathbf{x}}) = q(\hat{\mathbf{x}}) \frac{\partial^N}{\partial \hat{x}_{i_0} \partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}). \quad (3.28)$$

The inner product $\langle \mathcal{D}h(\hat{\mathbf{x}}), g(\hat{\mathbf{x}}) \rangle$ is then written as

$$\begin{aligned} \int g(\hat{\mathbf{x}}) \mathcal{D}h(\hat{\mathbf{x}}) d\hat{\mathbf{x}} &= \int g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}}) \frac{\partial^N}{\partial \hat{x}_{i_0} \partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{\mathbf{x}} = \\ &= \int \left(\int g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}}) \frac{\partial}{\partial \hat{x}_{i_0}} \frac{\partial^{N-1}}{\partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{x}_{i_0} \right) d\hat{\mathbf{x}}_{/i_0} \end{aligned}$$

where all the integration limits are $\pm\infty$. If we focus on the inner integral, and thanks to integration by parts

$$\begin{aligned} I(\hat{\mathbf{x}}_{/i_0}) &= \int g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}}) \frac{\partial}{\partial \hat{x}_{i_0}} \frac{\partial^{N-1}}{\partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{x}_{i_0} = \\ &= g(\hat{\mathbf{x}}_{/i_0}, x_{i_0}) q(\mathbf{x}_{/i_0}, \mathbf{x}_{i_0}) \frac{\partial^{N-1}}{\partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}_{/i_0}, x_{i_0}) \Big|_{x_{i_0}=-\infty}^{x_{i_0}=\infty} - \\ &= \int \frac{\partial}{\partial \hat{x}_{i_0}} (g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}})) \frac{\partial^{N-1}}{\partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{x}_{i_0}. \end{aligned}$$

Since we are considering vanishing functions, $g(\hat{\mathbf{x}}_{/i_0}, +\infty) = g(\hat{\mathbf{x}}_{/i_0}, -\infty) = 0$ and thus

$$I(\hat{\mathbf{x}}_{/i_0}) = - \int \frac{\partial}{\partial \hat{x}_{i_0}} (g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}})) \frac{\partial^{N-1}}{\partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{x}_{i_0}. \quad (3.29)$$

Consequently, we can rewrite the inner product as

$$\int g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}}) \frac{\partial^N}{\partial \hat{x}_{i_0} \partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{\mathbf{x}} = - \int \frac{\partial}{\partial \hat{x}_{i_0}} (g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}})) \frac{\partial^{N-1}}{\partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{\mathbf{x}}.$$

Repeating the same procedure N times, by induction, we write that

$$\int g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}}) \frac{\partial^N}{\partial \hat{x}_{i_0} \partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} h(\hat{\mathbf{x}}) d\hat{\mathbf{x}} = (-1)^N \int \frac{\partial^N}{\partial \hat{x}_{i_0} \partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} (g(\hat{\mathbf{x}}) q(\hat{\mathbf{x}})) h(\hat{\mathbf{x}}) d\hat{\mathbf{x}}. \quad (3.30)$$

By looking at (3.30), it is easy to derive that the adjoint of the considered operator (3.28) is

$$\mathcal{D}^\dagger(\hat{\mathbf{x}}) = (-1)^N \frac{\partial^N}{\partial \hat{x}_{i_0} \partial \hat{x}_{i_1} \dots \partial \hat{x}_{i_{N-1}}} (q(\hat{\mathbf{x}})h(\hat{\mathbf{x}})). \quad (3.31)$$

Due to linearity properties, we can then derive the adjoint of the infinitesimal generator (written as in form (3.26), or (3.27)):

$$\mathcal{A}^\dagger h(\hat{\mathbf{x}}) = - \sum_i \frac{\partial}{\partial \hat{x}_i} s_i(\hat{\mathbf{x}})h(\hat{\mathbf{x}}) + \sum_{i,j} \frac{\partial^2}{\partial \hat{x}_i \partial \hat{x}_j} \Sigma_{i,j}(\hat{\mathbf{x}})h(\hat{\mathbf{x}}) \quad (3.32)$$

or

$$\mathcal{A}^\dagger h(\hat{\mathbf{x}}) = -\nabla^\top (\mathbf{s}(\hat{\mathbf{x}})h(\hat{\mathbf{x}})) + \text{Tr}(\nabla \nabla^\top (\boldsymbol{\Sigma}(\hat{\mathbf{x}})h(\hat{\mathbf{x}}))) \quad (3.33)$$

3.2.3 Proof of Fokker Planck Equations

We present here a proof of Fokker Planck equations based on the adjoint of infinitesimal generator.

Consider again a smooth, vanishing at infinity function $h(\hat{\mathbf{x}})$. The expected value of $h(\mathbf{x}(t))$ conditioned on initial conditions can be written as

$$E[h(\mathbf{x}(t))|\mathbf{x}(0) = \hat{\mathbf{x}}_0] = \int h(\hat{\mathbf{x}})p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)d\hat{\mathbf{x}}. \quad (3.34)$$

The expected value of the differential $dh(\mathbf{x}(t))$ can be written, if we consider (3.34), as

$$E[dh(\mathbf{x}(t))|\mathbf{x}(0) = \hat{\mathbf{x}}_0] = \int h(\hat{\mathbf{x}})dp_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)d\hat{\mathbf{x}}, \quad (3.35)$$

where we defined $dp_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0) = p_{\mathbf{x}(t+dt)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t+dt|\hat{\mathbf{x}}_0, 0) - p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)$.

It also holds that, due to the Markovian nature of the process,

$$E[dh(\mathbf{x}(t))|\mathbf{x}(0) = \hat{\mathbf{x}}_0] = \int E[dh(\mathbf{x}(t))|\mathbf{x}(t) = \hat{\mathbf{x}}]p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)d\hat{\mathbf{x}}. \quad (3.36)$$

It holds, combining (3.35),(3.36) and (3.26), that

$$\begin{aligned} \int h(\hat{\mathbf{x}})dp_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)d\hat{\mathbf{x}} &= dt \int \mathcal{A}h(\hat{\mathbf{x}})p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)d\hat{\mathbf{x}} = \\ dt \int h(\hat{\mathbf{x}})\mathcal{A}^\dagger p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)d\hat{\mathbf{x}} \end{aligned} \quad (3.37)$$

Since (3.37) must hold for all smooth, vanishing functions h , it is evident that

$$dp_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0) = dt\mathcal{A}^\dagger p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0). \quad (3.38)$$

Consequently, recognizing the time derivative of the density function in the previous expression

$$\begin{aligned} \frac{\partial p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0)}{\partial t} &= \mathcal{A}^\dagger p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0) = \\ \text{Tr} \left\{ \nabla \left[-\mathbf{s}(\hat{\mathbf{x}})^\top p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0) + \nabla^\top \left(\boldsymbol{\Sigma}(\hat{\mathbf{x}}) p_{\mathbf{x}(t)|\mathbf{x}(0)}(\hat{\mathbf{x}}, t|\hat{\mathbf{x}}_0, 0) \right) \right] \right\} \end{aligned} \quad (3.39)$$

where we used (3.33). Dependence on initial conditions is usually omitted from the notation and thus (3.39) is completely equivalent to (3.11).

3.3 Ito vs Stratonovich: discretisation choice, noise induced drift and conversion between the two

As anticipated in the previous Sections, many different discretisation schemes are possible. Together with Ito SDE, a Stratonovich SDE, whose expression is

$$d\mathbf{x}(t) = \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t)) \circ d\mathbf{w}(t), \quad (3.40)$$

are the most common discretisation schemes. Notice the usage of symbol \circ to distinguish from other discretisations. Functionally, the above Stratonovich SDE corresponds to an Ito SDE of the following form

$$d\mathbf{x}(t) = \mathbf{s} \left(\frac{\mathbf{x}(t) + \mathbf{x}(t+dt)}{2} \right) dt + \sqrt{2}\mathbf{D} \left(\frac{\mathbf{x}(t) + \mathbf{x}(t+dt)}{2} \right) d\mathbf{w}(t), \quad (3.41)$$

or, since $\mathbf{x}(t) + \frac{d\mathbf{x}(t)}{2} = \frac{\mathbf{x}(t) + \mathbf{x}(t+dt)}{2}$, to

$$d\mathbf{x}(t) = \mathbf{s}(\mathbf{x}(t) + \frac{d\mathbf{x}(t)}{2})dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t) + \frac{d\mathbf{x}(t)}{2})d\mathbf{w}(t). \quad (3.42)$$

Slightly modifying the results of the previous section, it is trivial to prove that, for an usual smooth bounded function h ,

$$\begin{aligned} h(\mathbf{x}(t) + \alpha d\mathbf{x}(t)) &= h(\mathbf{x}(t)) + \left(\alpha \nabla^\top h(\mathbf{x}(t)) \mathbf{s}(\mathbf{x}(t)) + \alpha^2 \text{Tr}(\Delta h(\mathbf{x}(t)) \boldsymbol{\Sigma}(\mathbf{x}(t))) \right) dt \\ &+ \alpha \sqrt{2} \nabla^\top h(\mathbf{x}(t)) \mathbf{D}(\mathbf{x}(t)) d\mathbf{w}(t) \end{aligned} \quad (3.43)$$

Using this property, our goal is to derive the needed adjustments to an equation in the form of (3.42) to write it again as an equivalent Ito SDE. The first equality we need to derive is the expression for the p -th element of the new drift,

$$\begin{aligned} s_p(\mathbf{x}(t) + \alpha d\mathbf{x}(t))dt &= s_p(\mathbf{x}(t))dt + \left(\alpha \nabla^\top s_p(\mathbf{x}(t)) \mathbf{s}(\mathbf{x}(t)) + \alpha^2 \text{Tr}(\Delta s_p(\mathbf{x}(t)) \boldsymbol{\Sigma}(\mathbf{x}(t))) \right) dt^2 \\ &+ \alpha \sqrt{2} \nabla^\top s_p(\mathbf{x}(t)) \mathbf{D}(\mathbf{x}(t)) d\mathbf{w}(t)dt = s_p(\mathbf{x}(t))dt + \mathcal{O}(dt^2), \end{aligned} \quad (3.44)$$

where the last equality is derived with similar arguments to the previous section. Switching our attention to the p -th element of the noise component, i.e. $\sqrt{2} \sum_q D_{p,q}(\mathbf{x}(t) + \alpha d\mathbf{x}(t)) dw_q(t)$, the single term of the sum is expanded as

$$\begin{aligned}
 D_{p,q}(\mathbf{x}(t) + \alpha d\mathbf{x}(t)) dw_q(t) &= D_{p,q}(\mathbf{x}(t)) dw_q(t) + (\dots) dt dw_q(t) + \\
 \alpha \sqrt{2} \nabla^\top D_{p,q}(\mathbf{x}(t)) \mathbf{D}(\mathbf{x}(t)) d\mathbf{w}(t) dw_q(t) &= D_{p,q}(\mathbf{x}(t)) dw_q(t) + \\
 \alpha \sqrt{2} \sum_i \partial_i (D_{p,q}(\mathbf{x}(t))) \sum_r D_{i,r}(\mathbf{x}(t)) dw_r(t) dw_q(t) &= \\
 D_{p,q}(\mathbf{x}(t)) dw_q(t) + \alpha \sqrt{2} \sum_{i,r} \partial_i (D_{p,q}(\mathbf{x}(t))) D_{i,r}(\mathbf{x}(t)) \delta_{r,q} dt &= \\
 D_{p,q}(\mathbf{x}(t)) dw_q(t) + \alpha \sqrt{2} \sum_i \partial_i (D_{p,q}(\mathbf{x}(t))) D_{i,q}(\mathbf{x}(t)) dt & \quad (3.45)
 \end{aligned}$$

It is then possible to write

$$\begin{aligned}
 \sqrt{2} \sum_q D_{p,q}(\mathbf{x}(t) + \alpha d\mathbf{x}(t)) dw_q(t) &= \\
 \sqrt{2} \sum_q D_{p,q}(\mathbf{x}(t)) dw_q(t) + 2\alpha dt \sum_{i,q} \partial_i (D_{p,q}(\mathbf{x}(t))) D_{i,q}(\mathbf{x}(t)). & \quad (3.46)
 \end{aligned}$$

A more compact form in vectorial notation can be obtained by manipulating the second term of the above equation. We omit for simplicity the dependency on $\mathbf{x}(t)$. Starting with the basic application of chain rule

$$\sum_{i,q} \partial_i (D_{p,q}) D_{i,q} = \sum_{i,q} \partial_i (D_{p,q} D_{i,q}) - \sum_{i,q} \partial_i (D_{i,q}) D_{p,q} \quad (3.47)$$

Since $(\mathbf{D}\mathbf{D}^\top)_{i,p} = \sum_q D_{i,q} D_{p,q}$, then $(\nabla^\top (\mathbf{D}\mathbf{D}^\top))_p = \sum_i \partial_i (\mathbf{D}\mathbf{D}^\top)_{i,p} = \sum_{i,q} \partial_i (D_{i,q} D_{p,q})$. Similarly, $(\nabla^\top \mathbf{D})_q = \sum_i \partial_i D_{i,q}$, then $(\nabla^\top (\mathbf{D})\mathbf{D}^\top)_p = \sum_{i,q} \partial_i D_{i,q} D_{p,q}$. We can then write in vectorized form the term in (3.47) as

$$\sum_{i,q} \partial_i (D_{p,q}) D_{i,q} = (\nabla^\top (\mathbf{D}\mathbf{D}^\top))_p - (\nabla^\top (\mathbf{D})\mathbf{D}^\top)_p \quad (3.48)$$

Finally, we have from (3.44) that $\mathbf{s}(\mathbf{x}(t) + \frac{d\mathbf{x}(t)}{2}) dt = \mathbf{s}(\mathbf{x}(t)) dt$. From (3.46), (3.48) and since $\alpha = \frac{1}{2}$ we instead have that $\sqrt{2} \mathbf{D}(\mathbf{x}(t) + \frac{d\mathbf{x}(t)}{2}) d\mathbf{w}(t) = \sqrt{2} \mathbf{D}(\mathbf{x}(t)) d\mathbf{w}(t) + ((\nabla^\top (\mathbf{D}\mathbf{D}^\top)) - (\nabla^\top (\mathbf{D})\mathbf{D}^\top))^\top dt$. We can thus recast a Stratonovich SDE into Ito form as follows

$$\begin{aligned}
 d\mathbf{x}(t) &= \mathbf{s}(\mathbf{x}(t)) dt + \sqrt{2} \mathbf{D}(\mathbf{x}(t)) \circ d\mathbf{w}(t) \\
 &\quad \Downarrow \\
 d\mathbf{x}(t) &= [\mathbf{s}(\mathbf{x}(t)) + ((\nabla^\top (\mathbf{D}\mathbf{D}^\top)) - (\nabla^\top (\mathbf{D})\mathbf{D}^\top))^\top] dt + \sqrt{2} \mathbf{D}(\mathbf{x}(t)) d\mathbf{w}(t). \quad (3.49)
 \end{aligned}$$

It is easily show, just by reversing the order of argumentations, that the converse holds: starting from an Ito SDE, by correcting the dynamics we can find the equivalent Stratonovich SDE

$$\begin{aligned} \mathbf{dx}(t) &= \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\mathbf{dw}(t) \\ &\quad \Updownarrow \\ \mathbf{dx}(t) &= [\mathbf{s}(\mathbf{x}(t)) - ((\nabla^\top(\mathbf{D}\mathbf{D}^\top)) - (\nabla^\top(\mathbf{D})\mathbf{D}^\top))^\top]dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t)) \circ \mathbf{dw}(t). \end{aligned} \tag{3.50}$$

It has been argued that for several physical models the Stratonovich interpretation of the SDE is the correct one: arguments range from the celebrated Wong-Zakai theorem [96], that roughly speaking argues in favour of Stratonovich integral whenever the white noise in the physical system is an idealisation of a real, bandlimited noise process, to the Stratonovich integral being more natural since (as we will show shortly after) the classical rules of calculus hold. In our case it is not, strictly speaking, possible to argue similarly since as already mentioned before, we start from a discretisation scheme and study its properties in continuous time. Some of the strange effects, as the so called noise induced drift, can be interpreted by thinking of the Ito discretisation as a *wrong* model for a real physical system. It is fundamental to stress that this is just an interpretation.

In the remainder of this section, we skim through technical details to show that for Stratonovich discretisation the classical chain rule holds. We proceed as follows:

- we start from a generic Stratonovich SDE

$$\mathbf{dx}(t) = \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t)) \circ \mathbf{dw}(t)$$

- we write the equivalent Ito SDE

$$\mathbf{dx}(t) = [\mathbf{s}(\mathbf{x}(t)) + ((\nabla^\top(\mathbf{D}\mathbf{D}^\top)) - (\nabla^\top(\mathbf{D})\mathbf{D}^\top))^\top]dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\mathbf{dw}(t)$$

- we write the differential of a generic function for the Ito SDE

$$dh = (\nabla^\top h \mathbf{s} + \nabla^\top(\boldsymbol{\Sigma})\nabla h - \nabla^\top(\mathbf{D})\mathbf{D}^\top\nabla h + \text{Tr}(\Delta h \boldsymbol{\Sigma}))dt + \sqrt{2}\nabla^\top h \mathbf{D} \mathbf{dw}$$

- show the equivalence

$$\begin{aligned} \text{Tr}(\Delta h \boldsymbol{\Sigma}) + \nabla^\top(\boldsymbol{\Sigma})\nabla h - \nabla^\top(\mathbf{D})\mathbf{D}^\top\nabla h = \\ \partial_h(\nabla^\top h \boldsymbol{\Sigma} \nabla h) - \partial_h(\nabla^\top h \mathbf{D})\mathbf{D}^\top\nabla h \end{aligned}$$

- rewrite the Ito SDE as

$$dh = (\nabla^\top h \mathbf{s} dt + \partial_h(\nabla^\top h \boldsymbol{\Sigma} \nabla h) - \partial_h(\nabla^\top h \mathbf{D})\mathbf{D}^\top\nabla h)dt + \sqrt{2}\nabla^\top h \mathbf{D} \mathbf{dw}$$

- Switch back to a Stratonovich SDE

$$dh = (\nabla^\top h s dt) dt + \sqrt{2} \nabla^\top h \mathbf{D} \circ d\mathbf{w},$$

that is $dh = \nabla^\top h d\mathbf{x}$, i.e. chain rule holds for Stratonovich

We start with a couple of equalities, easily derived using the chain rule: $\frac{\partial h}{\partial h} = 1 = \sum_i \frac{\partial \hat{x}_i}{\partial h} \frac{\partial h}{\partial x_i}$, $\delta_{ij} = \frac{\partial \hat{x}_i}{\partial h} \frac{\partial h}{\partial x_j}$, $\frac{\partial}{\partial h} \frac{\partial h}{\partial \hat{x}_i} = \sum_j \frac{\partial \hat{x}_j}{\partial h} \frac{\partial^2 h}{\partial x_i \partial x_j}$, and for a generic function $D_{p,q}$, $\frac{\partial D_{p,q}}{\partial h} = \sum_j \frac{\partial \hat{x}_j}{\partial h} \frac{\partial D_{p,q}}{\partial x_j}$.

We rewrite in index form $\nabla^\top h \Sigma \nabla h = \sum_{i,j} \frac{\partial h}{\partial \hat{x}_i} \frac{\partial h}{\partial \hat{x}_j} \Sigma_{i,j}$. Consequently

$$\begin{aligned} \frac{\partial}{\partial h} \sum_{i,j} \left(\frac{\partial h}{\partial \hat{x}_i} \frac{\partial h}{\partial \hat{x}_j} \Sigma_{i,j} \right) &= \sum_{i,j} \frac{\partial}{\partial h} \left(\frac{\partial h}{\partial \hat{x}_i} \right) \frac{\partial h}{\partial \hat{x}_j} \Sigma_{i,j} + \sum_{i,j} \frac{\partial}{\partial h} \left(\frac{\partial h}{\partial \hat{x}_j} \right) \frac{\partial h}{\partial \hat{x}_i} \Sigma_{i,j} + \sum_{i,j} \frac{\partial}{\partial h} (\Sigma_{i,j}) \frac{\partial h}{\partial \hat{x}_i} \frac{\partial h}{\partial \hat{x}_j} = \\ &= \sum_{i,j,p} \frac{\partial h}{\partial \hat{x}_j} \frac{\partial \hat{x}_p}{\partial h} \frac{\partial^2 h}{\partial x_p \partial x_i} \Sigma_{i,j} + \sum_{i,j,p} \frac{\partial h}{\partial \hat{x}_i} \frac{\partial \hat{x}_p}{\partial h} \frac{\partial^2 h}{\partial x_p \partial x_j} \Sigma_{i,j} + \sum_{i,j,p} \frac{\partial \hat{x}_p}{\partial h} \frac{\partial \Sigma_{i,j}}{\partial x_p} \frac{\partial h}{\partial \hat{x}_i} \frac{\partial h}{\partial \hat{x}_j} = \\ &= \sum_{i,j,p} \delta_{jp} \frac{\partial^2 h}{\partial x_p \partial x_i} \Sigma_{i,j} + \sum_{i,j,p} \delta_{ip} \frac{\partial^2 h}{\partial x_p \partial x_j} \Sigma_{i,j} + \sum_{i,j,p} \delta_{ip} \frac{\partial \Sigma_{i,j}}{\partial x_p} \frac{\partial h}{\partial \hat{x}_j} = \\ &= 2 \sum_{i,j} \frac{\partial^2 h}{\partial x_i \partial x_j} \Sigma_{i,j} + \sum_{i,j} \frac{\partial \Sigma_{i,j}}{\partial x_i} \frac{\partial h}{\partial \hat{x}_j} = \\ &= 2 \text{Tr}(\Delta h \Sigma) + \nabla^\top (\Sigma) \nabla h \end{aligned}$$

Similarly, we compute $\partial_h (\nabla^\top h \mathbf{D}) \mathbf{D}^\top \nabla h$ as

$$\begin{aligned} \sum_{i,r,q} \frac{\partial}{\partial h} \left(\frac{\partial h}{\partial \hat{x}_i} D_{iq} \right) \frac{\partial h}{\partial \hat{x}_r} D_{rq} &= \sum_{i,r,q,p} \frac{\partial \hat{x}_p}{\partial h} \frac{\partial^2 h}{\partial \hat{x}_i \partial \hat{x}_p} D_{iq} \frac{\partial h}{\partial \hat{x}_r} D_{rq} + \sum_{i,r,q,p} \frac{\partial h}{\partial \hat{x}_i} \frac{\partial \hat{x}_p}{\partial h} \frac{\partial D_{iq}}{\partial \hat{x}_p} \frac{\partial h}{\partial \hat{x}_r} D_{rq} = \\ &= \sum_{i,r,q,p} \delta_{pr} \frac{\partial^2 h}{\partial \hat{x}_i \partial \hat{x}_p} D_{iq} D_{rq} + \sum_{i,r,q,p} \delta_{pi} \frac{\partial D_{iq}}{\partial \hat{x}_p} \frac{\partial h}{\partial \hat{x}_r} D_{rq} = \\ &= \sum_{i,r,q} \frac{\partial^2 h}{\partial \hat{x}_i \partial \hat{x}_r} D_{iq} D_{rq} + \sum_{i,r,q} \frac{\partial h}{\partial \hat{x}_r} \frac{\partial D_{iq}}{\partial \hat{x}_i} D_{rq} = \text{Tr}(\Delta h \Sigma) + \nabla^\top (\mathbf{D}) \mathbf{D}^\top \nabla h \end{aligned}$$

We can then finally show that

$$\begin{aligned} \partial_h (\nabla^\top h \Sigma \nabla h) - \partial_h (\nabla^\top h \mathbf{D}) \mathbf{D}^\top \nabla h &= 2 \text{Tr}(\Delta h \Sigma) + \nabla^\top (\Sigma) \nabla h - \text{Tr}(\Delta h \Sigma) - \\ \nabla^\top (\mathbf{D}) \mathbf{D}^\top \nabla h &= \text{Tr}(\Delta h \Sigma) + \nabla^\top (\Sigma) \nabla h - \nabla^\top (\mathbf{D}) \mathbf{D}^\top \nabla h, \end{aligned}$$

q.e.d.

3.3.1 Equivalence conditions

Studying under which conditions the Stratonovich and Ito discretisation coincide is particularly interesting from both a theoretical and practical point of view. As stated before, in this thesis we will consider only the Ito formalism. Nevertheless we do include this brief and self contained subsection as a starting point for the reader interested in exploring different discretisation schemes.

The reasons for exploring such a connection are multiple. For the reader that is familiar with a literature in which one of the two formalism is used, it is helpful to understand under which circumstances the two formalisms coincide. In particular, this could unleash ideas and techniques for accurate simulation of the stochastic dynamics whose roots are discipline specific, as is for example the case of [16] (we will explore more about this in Chapter 5). Moreover, since there are many hand crafted methods for the accurate numerical simulation of SDE s that are valid only for one of the two formalism [45], it is useful to know when the techniques tailored for one of the two discretisations can be blindly applied to the problem of interest since the formalisms coincide. Formally if $\nabla^\top \Sigma = 0$ and $\nabla^\top \mathbf{D} = 0$ then

- from (3.48), $\sum_{i,q} \partial_i (D_{p,q}) D_{i,q} = (\nabla^\top (\mathbf{D}\mathbf{D}^\top))_p - (\nabla^\top (\mathbf{D})\mathbf{D}^\top)_p = 0$, the noise induced drift is zero and there is no difference between Ito and Stratonovich SDE
- since $\nabla^\top \Sigma = 0$, the FPE considerably simplify as

$$\frac{\partial p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t)}{\partial t} = \text{Tr} \left\{ \nabla \left[-\mathbf{s}(\hat{\mathbf{x}})^\top p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t) + \nabla^\top \left(p_{\mathbf{x}(t)}(\hat{\mathbf{x}}, t) \right) \Sigma(\hat{\mathbf{x}}) \right] \right\} \quad (3.51)$$

We analyse in details two important cases: the case of second order (momentum) differential equation, in Section 3.3.1, that will be extremely important for understanding the results presented in Section 5.4, and the case of time correlated noise, in Section 3.3.1, a case unexplored to the best of our knowledge in the Bayesian sampling literature.

Second order differential equation

The first considered case in which the two aforementioned conditions hold is the case in which a second order differential equation based model is considered. To avoid clutter we keep the notation simple and do not consider the most general case. It will be straightforward (see Section 5.4) to formalize an equivalence between second order systems and gradient flows with momentum. We start with the following SDE (without prescription of the discretisation), derived from a given physical model

$$\frac{d^2 \mathbf{x}(t)}{dt^2} + \gamma \frac{d\mathbf{x}(t)}{dt} - \mathbf{s}(\mathbf{x}(t)) - \sqrt{2}\mathbf{D}(\mathbf{x}(t)) \frac{d\mathbf{w}(t)}{dt} = 0. \quad (3.52)$$

Define the velocity variable as $\mathbf{v}(t) = \frac{d\mathbf{x}(t)}{dt}$. It is then straightforward to rewrite (3.52) as a system of first order differential equations

$$\begin{cases} \mathbf{v}(t) = \frac{d\mathbf{x}(t)}{dt} \\ \frac{d\mathbf{v}(t)}{dt} = -\gamma\mathbf{v}(t) + \mathbf{s}(\mathbf{x}(t)) + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\frac{d\mathbf{w}(t)}{dt}, \end{cases} \quad (3.53)$$

or, in differential form

$$\begin{cases} d\mathbf{x}(t) = \mathbf{v}(t)dt \\ d\mathbf{v}(t) = -\gamma\mathbf{v}(t)dt + \mathbf{s}(\mathbf{x}(t))dt + \sqrt{2}\mathbf{D}(\mathbf{x}(t))d\mathbf{w}(t). \end{cases} \quad (3.54)$$

Defining the super variable $\mathbf{z}(t) = [\mathbf{x}(t), \mathbf{v}(t)]^\top$ the system can be further rewritten as

$$d\mathbf{z}(t) = \begin{bmatrix} \mathbf{0} & -\mathbf{I} \\ \mathbf{0} & -\gamma\mathbf{I} \end{bmatrix} \mathbf{z}(t)dt + [\mathbf{0}, \mathbf{s}^\top(\mathbf{x}(t))]^\top dt + \sqrt{2}[\mathbf{0}, \mathbf{D}^\top(\mathbf{x}(t))]^\top d\mathbf{k}(t), \quad (3.55)$$

where $d\mathbf{k}(t)$ is a $2N$ -dimensional Wiener process. The equation can be further modified as

$$d\mathbf{z}(t) = \begin{bmatrix} \mathbf{0} & -\mathbf{I} \\ \mathbf{I} & -\gamma\mathbf{I} \end{bmatrix} \mathbf{q}(\mathbf{z}(t))dt + \sqrt{2}\mathbf{D}_E(\mathbf{z}(t))d\mathbf{k}(t), \quad (3.56)$$

where $\mathbf{q}(\mathbf{z}(t)) = [\mathbf{s}^\top(\mathbf{x}(t)), \mathbf{v}(t)]$ and $\mathbf{D}_E(\mathbf{z}(t)) = [\mathbf{0}, \mathbf{D}^\top(\mathbf{x}(t))]^\top$. Since for the new system we have $\nabla = \nabla_{\mathbf{z}} = [\nabla_{\hat{\mathbf{x}}}, \nabla_{\hat{\mathbf{v}}}]^\top$, then

$$\nabla^\top \mathbf{D}_E(\mathbf{z}(t)) = [\nabla_x^\top \mathbf{0}, \nabla_v^\top \mathbf{D}(\mathbf{x}(t))]^\top = [\mathbf{0}^\top, \mathbf{0}^\top] = \mathbf{0}^\top.$$

Similarly, since

$$\Sigma_E(\mathbf{z}(t)) = \mathbf{D}_E(\mathbf{z}(t))\mathbf{D}_E(\mathbf{z}(t))^\top = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}(\mathbf{x}(t))\mathbf{D}^\top(\mathbf{x}(t)) \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Sigma(\mathbf{x}(t)) \end{bmatrix},$$

it is shown that $\nabla^\top \Sigma_E(\mathbf{z}(t)) = \mathbf{0}$

Time colored noise

In general, when dealing with time colored noise, the Fokker-Planck theory breaks down. Alternatives, such as path integral methods, can provide cleaner and faster results.

One particular subcase, for which it is possible to state that the divergence term of the diffusion is zero is the exponentially (in time) colored noise. Formally, the noise process $\mathbf{z}(t)$, instead of being white Gaussian noise (3.3)

$$E[\mathbf{z}(t)\mathbf{z}^\top(t+\tau)] = \mathbf{I}\delta(\tau),$$

is a time correlated noise

$$E[\mathbf{z}(t)\mathbf{z}^\top(t+\tau)] = \mathbf{I}\frac{T_0}{2}\exp(-\frac{|\tau|}{T_0}),$$

notice that as $T_0 \rightarrow 0$, $\frac{T_0}{2}\exp(-\frac{|\tau|}{T_0}) \rightarrow \delta(\tau)$. It is a well known fact in the signal theory community that any time colored Gaussian noise process, i.e. a process whose autocorrelation $R_z(\tau) = E[z(t)z(t+\tau)]$ is not a Dirac delta, can be equivalently rewritten as the convolution of a white Gaussian process $n(t)$ and a filter $h(t)$ as follows

$$\begin{cases} z(t) = h(t) * n(t) \\ R_z(\tau) = h(\tau) * h(-\tau) \end{cases} \quad (3.57)$$

In the considered case, $R_z(\tau) = \frac{T_0}{2}\exp(-\frac{|\tau|}{T_0})$, it is easy to show that $h(t) = \exp(-\frac{t}{T_0})u(t)$, being $u(t)$ the Heaviside step function. If we are interested in the time derivative of the process, we can write

$$\frac{dz(t)}{dt} = \frac{dh(t)}{dt} * n(t), \quad (3.58)$$

and since $\frac{dh(t)}{dt} = -\frac{1}{T_0}\exp(-\frac{t}{T_0})u(t) + \exp(-\frac{t}{T_0})\delta(t) = -\frac{1}{T_0}h(t) + \delta(t)$, we equivalently have

$$\frac{dz(t)}{dt} = -\frac{1}{T_0}z(t) + n(t). \quad (3.59)$$

We can rearrange in differential form

$$dz(t) = -\frac{1}{T_0}z(t)dt + dw(t) \quad (3.60)$$

where $dw(t)$ is the usual Wiener process. The result of (3.60) is particularly important if we want to transform a first order Langevin equation with time colored noise

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{s}(\mathbf{x}(t)) + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\mathbf{z}(t) \quad (3.61)$$

into a system of differential equations with white noise

$$\begin{cases} \frac{d\mathbf{x}(t)}{dt} = \mathbf{s}(\mathbf{x}(t)) + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\mathbf{z}(t) \\ \frac{dz(t)}{dt} = -\frac{1}{T_0}\mathbf{z}(t) + \mathbf{n}(t) \end{cases} \rightarrow \begin{cases} d\mathbf{x}(t) = (\mathbf{s}(\mathbf{x}(t)) + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\mathbf{z}(t))dt \\ dz(t) = -\frac{1}{T_0}\mathbf{z}(t)dt + d\mathbf{w}(t) \end{cases} \quad (3.62)$$

As in the previous case, define a super variable $\mathbf{y}(t)^\top = [\mathbf{x}(t)^\top, \mathbf{z}(t)^\top]$, and rewrite the system of equations as

$$d\mathbf{y}(t) = \begin{bmatrix} \mathbf{s}(\mathbf{x}(t)) + \sqrt{2}\mathbf{D}(\mathbf{x}(t))\mathbf{z}(t) \\ -\frac{1}{T_0}\mathbf{z}(t) \end{bmatrix} dt + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} d\mathbf{h} \quad (3.63)$$

where $d\mathbf{h}$ is a Wiener process.

It is then trivial to understand that both desired conditions, $\nabla^\top \Sigma = 0$, $\nabla^\top \mathbf{D} = 0$, hold.

Chapter 4

Bayesian Machine Learning: the Montecarlo Integration approach

In this chapter we treat the motivating problem that has triggered during last years the exploration of connections between gradient algorithms and stochastic processes: the need for estimating high dimensional probability integrals without analytical tools.

In section 4.2 we present the general problem of Montecarlo integration: a sample based stochastic approximation of a certain class of integrals by means of statistical averages of functions evaluated at random points. In section 4.3 we treat the particular case (although central to the thesis) of Bayesian Machine Learning and distinguish between pointwise estimates of parametric models versus distributional one. Important quantities linked to uncertainty quantification are casted as particular problems of the integrals of interest. Section 4.4 draws the link between Montecarlo integration and gradient methods: by leveraging on the theory of Chapter 3 we show that variants of gradient descent can be used to collect samples drawn from specific probability distributions usueful for the estimation of Montecarlo integrals.

4.1 Overview

High dimensional, integrals, arise naturally in a wide range of applications in engineering, physics, biology. Unluckily, besides from simple cases no closed form solution is available.

The fundamental problem we will analyse in this chapter is the usage of Montecarlo methods for the computation of integrals of the form

$$I = \int_{\mathbb{R}^N} \mathbf{m}(\hat{\mathbf{x}})p(\hat{\mathbf{x}})d\hat{\mathbf{x}} \quad (4.1)$$

where $\mathbf{m}(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^D$ is a generic D -dimensional vector function and $p(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbb{R}_+$ is a positive, unit integral, weight function, i.e.

$$\begin{cases} \int_{\mathbb{R}^N} p(\hat{\mathbf{x}}) d\hat{\mathbf{x}} = 1 \\ p(\hat{\mathbf{x}}) \geq 0 \quad \forall \hat{\mathbf{x}} \in \mathbb{R}^N \end{cases} \quad (4.2)$$

During this dissertation we will often need to work with the logarithm of $p(\hat{\mathbf{x}})$. It is then better to rewrite equivalently the integral as

$$\begin{cases} I = \int_{\mathcal{S}} \mathbf{m}(\hat{\mathbf{x}}) p(\hat{\mathbf{x}}) d\hat{\mathbf{x}} \\ \int_{\mathcal{S}} p(\hat{\mathbf{x}}) d\hat{\mathbf{x}} = 1 \\ p(\hat{\mathbf{x}}) > 0 \quad \forall \hat{\mathbf{x}} \in \mathcal{S} \end{cases} \quad (4.3)$$

where $\mathcal{S} \subseteq \mathbb{R}^N$, and the strong positivity condition on $p(\hat{\mathbf{x}})$ is enforced (> 0 instead of ≥ 0).

It is easy to notice that we can conceptually link $p(\hat{\mathbf{x}})$ to a probability density function and interpret (4.3) as the expected value of the function \mathbf{m} of a random variable \mathbf{x} distributed according to $p_{\mathbf{x}}(\hat{\mathbf{x}})$, i.e $I = E[\mathbf{m}(\mathbf{x})]$.

In this exposition we write, when necessary and without losing generality, $p_{\mathbf{x}}(\hat{\mathbf{x}}) = \exp(-f(\hat{\mathbf{x}}))$ where, obviously, $f(\hat{\mathbf{x}})$ is implicitly defined as $f(\hat{\mathbf{x}}) = \log(p_{\mathbf{x}}(\hat{\mathbf{x}}))$. We can then rewrite (4.1) as

$$I = \int_{\mathcal{S}} \mathbf{m}(\hat{\mathbf{x}}) \exp(-f(\hat{\mathbf{x}})) d\hat{\mathbf{x}} \quad (4.4)$$

4.2 Montecarlo integration

Knowing the analytic expression for both the functions \mathbf{m} , f does not guarantee of being able to analytically compute the integral (4.4). This consideration led alone to the study and development of Montecarlo integrations techniques. Montecarlo methods have their roots in the first half of the last century, dating at least back to unpublished experiments of Enrico Fermi in the 30's [59] while studying neutron diffusion. Several references are available for the general topic, some possibilities being [63],[35],[73], although the list is far from complete.

As discussed before, $p_{\mathbf{x}}(\hat{\mathbf{x}})$ can be thought as a probability density function. If we are able to draw statistically independent samples \mathbf{x}_i from this distribution (that is, $\mathbf{x}_i \sim p_{\mathbf{x}}(\hat{\mathbf{x}})$), it is easy to show that the random variable

$$\bar{I}(N_{\text{MC}}) = \frac{1}{N_{\text{MC}}} \sum_{i=0}^{N_{\text{MC}}-1} \mathbf{m}(\mathbf{x}_i), \quad (4.5)$$

provides, in the large samples limit, where the number of samples is N_{MC} , a good approximation to (4.4).

Informally, the following holds

$$\left| \lim_{N_{\text{MC}} \rightarrow \infty} (\bar{I}(N_{\text{MC}}) - I) \right| \rightarrow 0 \quad \text{almost surely} \quad (4.6)$$

An extremely simple example is the problem of estimation of π . We start by writing π as the area of a circle with radius $r = 1$. Trivial calculations show that we can equivalently write, by integrating over $\mathcal{S} = \{\hat{x}^2 + \hat{y}^2 < 1, \quad 0 \leq \hat{x} \leq 1, \quad 0 \leq \hat{y} \leq 1\}$, the following equality

$$\pi = 4 \int_{\mathcal{S}} d\hat{x}d\hat{y} = \int_0^1 \int_0^1 4 \times \mathbb{1}(\hat{x}^2 + \hat{y}^2 < 1) d\hat{x}d\hat{y} = \int 4 \times \mathbb{1}(\hat{x}^2 + \hat{y}^2 < 1) p_{x,y}(\hat{x}, \hat{y}) d\hat{x}d\hat{y}, \quad (4.7)$$

where $\mathbb{1}(\cdot)$ is the indicator function and $p_{x,y}(\hat{x}, \hat{y})$ is the pdf of two independent random variables uniformly distributed between 0 and 1. It is then easy to understand that we can cast the problem of computing π as a Montecarlo integration

$$\bar{\pi}(N_{\text{MC}}) = \frac{4}{N_{\text{MC}}} \sum_{i=0}^{N_{\text{MC}}-1} \mathbb{1}(x_i^2 + y_i^2 < 1) \quad (4.8)$$

where x_i, y_i are sampled independently from $U[0,1]$. In figure 4.1 we depict the result of the Montecarlo integration for $N_{\text{MC}} = 1000$: for the considered experiment the estimated value of π is 3.187999963760376

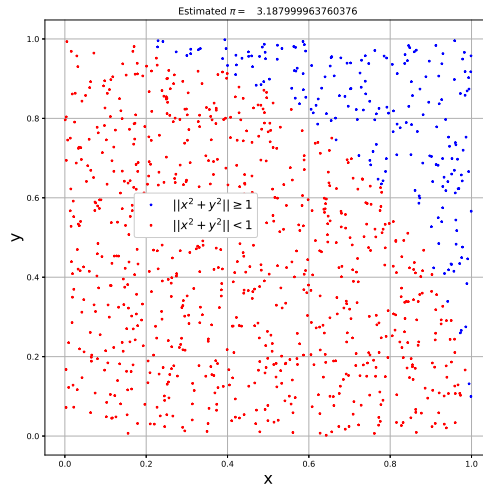


Figure 4.1: Montecarlo estimation of π using $N_{\text{MC}} = 1000$

4.2.1 Montecarlo Integrals are unbiased and consistent

The exact description of the convergence properties of the considered scheme is out of the scope of this exposition, but we nevertheless underline two important properties. The first is the following:

$$E_{\mathbf{x}_i \sim p_{\mathbf{x}}(\hat{\mathbf{x}})}[\mathbf{m}(\mathbf{x}_i)] = \int_{\mathcal{S}} \mathbf{m}(\hat{\mathbf{x}}) p_{\mathbf{x}}(\hat{\mathbf{x}}) d\mathbf{x} = I \quad (4.9)$$

that obviously translates into

$$E[\bar{I}(N_{\text{MC}})] = I \quad (4.10)$$

The second important consideration is that, under reasonable conditions, the considered estimation scheme is consistent, i.e. the estimation error variance goes to zero as N_{MC} goes to infinity. In fact

$$\begin{aligned} E[(\bar{I} - I)^2] &= E[\bar{I}^2] - I^2 = E\left[\left(\frac{1}{N_{\text{MC}}} \sum_{i=0}^{N_{\text{MC}}-1} \mathbf{m}(\mathbf{x}_i)\right)^2\right] - I^2 = \\ &= E\left[\frac{1}{N_{\text{MC}}^2} \sum_{i,j} \mathbf{m}(\mathbf{x}_i)\mathbf{m}(\mathbf{x}_j)\right] - I^2 = E\left[\frac{1}{N_{\text{MC}}^2} \sum_{i,j=i} \mathbf{m}^2(\mathbf{x}_i)\right] + E\left[\frac{1}{N_{\text{MC}}^2} \sum_{i,j \neq i} \mathbf{m}(\mathbf{x}_i)\mathbf{m}(\mathbf{x}_j)\right] - I^2 = \\ &= \frac{N_{\text{MC}}}{N_{\text{MC}}^2} E[\mathbf{m}^2(\mathbf{x}_i)] + \left(\frac{N_{\text{MC}}(N_{\text{MC}} - 1)}{N_{\text{MC}}^2} - 1\right) I^2 = \frac{1}{N_{\text{MC}}} E[(\mathbf{m}(\mathbf{x}_i) - E[\mathbf{m}(\mathbf{x}_i)])^2]. \end{aligned}$$

Under the mild assumption that $\sigma^2 = E[(\mathbf{m}(\mathbf{x}_i) - E[\mathbf{m}(\mathbf{x}_i)])^2]$ is a finite quantity, it is then evident that as $N_{\text{MC}} \rightarrow \infty$ the estimation error variance goes to zero. We study, similarly to the previous section, the same Montecarlo integral $\bar{\pi}(N_{\text{MC}}) = \frac{4}{N_{\text{MC}}} \sum_{i=0}^{N_{\text{MC}}-1} \mathbb{1}(x_i^2 + y_i^2 < 1)$. We present in figure 4.2 the empirical mean, 5 and 95 percentile values for the estimated value of π as a function of the number of samples N_{MC} , considering 100 different experiments. As expected the mean is the correct one and the variance monotonically decreases with the number of samples N_{MC} .

Notice however, that from a practical point of view, σ^2 can be extremely large, especially when the dimensionality of the problem (N) is large.

4.3 Bayesian problems in Machine learning

In this section we build the link between Montecarlo integration and Bayesian problems in machine learning. The choice of presenting Bayesian problems for machine learning after the general framework of Montecarlo integration is not casual. All the methods described in this thesis are valid for estimation of integrals of the form (4.1). The problem of Bayesian estimation in machine learning is a special case (although extremely important and worth of particular attention per se) of

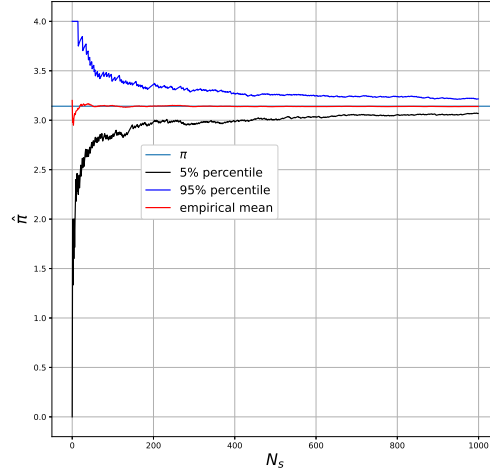


Figure 4.2: Monte Carlo estimation of π performing 1000 experiments as a function of N_{MC}

the general problem of estimation of probability density integrals. The results presented in this discussion are thus of interest for a broader audience than the one only interested in Bayesian machine learning.

More or less directly, more or less explicitly, most (if not all) modern (parametric) machine learning problems can be cast as optimization problems. Consider an observed data-set of m -dimensional observations $\hat{\mathcal{D}} = \{\hat{\mathbf{u}}_i\}_{i=1}^N$, a realization of the random variable $\mathcal{D} = \{\mathbf{u}_i\}_{i=1}^N$, and an arbitrary complex parametric probability density function $p_{\mathcal{D}|\mathbf{x}}(\hat{\mathcal{D}}|\hat{\mathbf{x}})$ (where \mathbf{x} is the set of parameters), called likelihood model. Given prior $p_{\mathbf{x}}(\hat{\mathbf{x}})$ for the d -dimensional set of parameters, the posterior is obtained by means of Bayes theorem as follows:

$$p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) = \frac{p_{\mathcal{D}|\mathbf{x}}(\hat{\mathcal{D}}|\hat{\mathbf{x}}) p_{\mathbf{x}}(\hat{\mathbf{x}})}{p_{\mathcal{D}}(\hat{\mathcal{D}})} \quad (4.11)$$

where $p_{\mathcal{D}}(\hat{\mathcal{D}})$ is also known as the model evidence, defined as the integral $p_{\mathcal{D}}(\hat{\mathcal{D}}) = \int p_{\mathcal{D}|\mathbf{x}}(\hat{\mathcal{D}}|\hat{\mathbf{x}}) p_{\mathbf{x}}(\hat{\mathbf{x}}) d\mathbf{x}$. Except when the prior and the likelihood function are conjugate, Eq. (4.11) is analytically intractable [7]. However, the joint likelihood term in the numerator is typically not hard to compute; this is a key element, since the normalisation constant $p_{\mathcal{D}}(\hat{\mathcal{D}})$ does not affect the shape of the distribution in any way other than scaling. The common assumption of most modern machine approaches is that the data-set samples are independent. Consequently, the posterior

probability of (4.11) can be written as

$$p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) = \left(\prod_{i=1}^N p_{\mathbf{u}_i|\mathbf{x}}(\hat{\mathbf{u}}_i|\hat{\mathbf{x}}) \right) \frac{p_{\mathbf{x}}(\hat{\mathbf{x}})}{p_{\mathcal{D}}(\hat{\mathcal{D}})} \quad (4.12)$$

The setting described by (4.12), is the most general. We consider in the following exposition the case of supervised classification, i.e. when the observations \mathbf{u} are composed of "inputs" datapoints \mathbf{s} and corresponding classes y . Some examples include

- The problem of **supervised classification**, in which the dataset is composed of couples of input data-points \mathbf{s}_i and corresponding classes y_i (among N_{cl} possible classes). The parametric likelihood is of the following form

$$p_{\mathbf{s}_i, y_i|\mathbf{x}}(\hat{\mathbf{s}}_i, \hat{y}_i|\hat{\mathbf{x}}) = p_{y_i|\mathbf{x}, \mathbf{s}_i}(\hat{y}_i|\hat{\mathbf{x}}, \hat{\mathbf{s}}_i) p_{\mathbf{s}_i}(\hat{\mathbf{s}}_i) = (\mathbf{h}_{\hat{\mathbf{x}}}(\hat{\mathbf{s}}_i))_{y_i} p_{\mathbf{s}_i}(\hat{\mathbf{s}}_i) \quad (4.13)$$

where $\mathbf{h}_{\hat{\mathbf{x}}}(\cdot)$ is a function that maps $\hat{\mathbf{s}}_i$ into a point in the N_{cl} -dimensional simplex, i.e. $\mathcal{C} = \{\hat{\mathbf{z}} \in \mathbb{R}^{N_{cl}} : 0 \leq \hat{z}_i \leq 1, \sum_{i=1}^{N_{cl}} \hat{z}_i = 1\}$

- The problem of **supervised regression**, in which the dataset is composed of couples of input data-points \mathbf{s}_i and corresponding target values $\mathbf{y}_i \in \mathbb{R}^P$, being P a generic positive integer. The parametric likelihood is of the following form

$$p_{\mathbf{s}_i, \mathbf{y}_i|\mathbf{x}}(\hat{\mathbf{s}}_i, \hat{\mathbf{y}}_i|\hat{\mathbf{x}}) = p_{\mathbf{y}_i|\mathbf{x}, \mathbf{s}_i}(\hat{\mathbf{y}}_i|\hat{\mathbf{x}}, \hat{\mathbf{s}}_i) p_{\mathbf{s}_i}(\hat{\mathbf{s}}_i) = (\mathbf{h}_{\hat{\mathbf{x}}}(\hat{\mathbf{s}}_i))(\hat{\mathbf{y}}_i) p_{\mathbf{s}_i}(\hat{\mathbf{s}}_i) \quad (4.14)$$

where $\mathbf{h}_{\hat{\mathbf{x}}}(\cdot)$ is a function that maps the input datapoint \mathbf{s}_i into a probability density function with domain \mathbb{R}^P . To make a concrete example

$$(\mathbf{h}_{\hat{\mathbf{x}}}(\hat{\mathbf{s}}_i))(\cdot) \propto \exp\left(-\frac{\|\cdot - \boldsymbol{\mu}_{\hat{\mathbf{x}}}\|^2}{2\sigma^2}\right) \quad (4.15)$$

The usual procedure to let the machine learn, and subsequently use it to make predictions, is the following

- Choose a prior distribution for $p_{\mathbf{x}}(\hat{\mathbf{x}})$ (possibly also the uniform one). Find, using any kind of optimization technique, the set of parameters $\hat{\mathbf{x}}^{opt}$ as

$$\hat{\mathbf{x}}^{opt} = \arg \max_{\hat{\mathbf{x}}} p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) \quad (4.16)$$

- use the optimized set of parameters to deal with a new previously unseen observation $\hat{\mathbf{s}}_*$ and compute $p_{y_*|\mathbf{s}_*, \mathbf{x}}(\hat{y}_*|\hat{\mathbf{s}}_*, \hat{\mathbf{x}}^{opt})$

The afore described technique is called pointwise estimation/optimization. It is in fact easy to notice that the considered optimization is nothing but the classical maximum a posteriori (MAP) methodology (notice that the Maximum Likelihood is trivially recovered as a particular case when the prior distribution for $p_{\mathbf{x}}(\hat{\mathbf{x}})$ is uniform). While simple, the usage of pointwise estimation is not, from a theoretical point of view, completely correct. The true posterior distribution for a new test observations $[\hat{y}, \hat{\mathbf{s}}_*]$ is in fact:

$$p_{y_*|s_*,\mathcal{D}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathcal{D}}) = \int p_{y_*|s_*,\mathbf{x}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathbf{x}})p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})d\hat{\mathbf{x}} \quad (4.17)$$

Integrals of the form (4.17) are generally intractable and will be estimated using Montecarlo techniques. The point wise estimation induces implicitly the following approximation $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) \simeq \delta(\hat{\mathbf{x}} - \hat{\mathbf{x}}^{opt})$, where $\delta(\cdot)$ is a multidimensional Dirac delta. This approximation, far from being of interest only for theoretician, has the serious drawback that we do not have access to the distribution $p_{u_*|\mathcal{D}}(\hat{u}_*|\hat{\mathcal{D}})$ but only to a single point value. This drawback has been out weighted historically to the practical benefit of being able to evaluate the integral in (4.17) since, with the considered approximation,

$$p_{y_*|s_*,\mathcal{D}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathcal{D}}) \simeq \int p_{y_*|s_*,\mathbf{x}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathbf{x}})\delta(\hat{\mathbf{x}} - \hat{\mathbf{x}}^{opt})d\hat{\mathbf{x}} = p_{y_*|s_*,\mathbf{x}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathbf{x}}^{opt}) \quad (4.18)$$

For modern problems, however, this approximation is not sufficient, as we will shortly highlight, and the full distribution must be used instead.

We focus for simplicity of exposition in the remaining of this introduction on the problem of classification. Usually, at test time, we are interested in the class probability given a new input datapoint,i.e

$$p_{y_*|s_*,\mathcal{D}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathcal{D}}) = \int p_{y_*|s_*,\mathbf{x}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathbf{x}})p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})d\hat{\mathbf{x}} \quad (4.19)$$

that can be either estimated using the naive approach of (4.18), or by computing the intractable integral of (4.19) using Montecarlo techniques as

$$p_{y_*|s_*,\mathcal{D}}(\hat{y}_*|\hat{\mathbf{s}}_*,\hat{\mathcal{D}}) = \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} p_{y_*|s_*,\mathbf{x}_i}(\hat{y}_*|\hat{\mathbf{s}}_*,\mathbf{x}_i), \quad (4.20)$$

where $\mathbf{x}_i \sim p_{\mathbf{x}_i|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$.

In particular, we focus on the role of having access to the full distribution, instead of the pointwise estimate, when dealing with large black box models that are difficult to be inspected, such as deep neural networks. Having access to the distribution of values is of fundamental importance, especially for critical problems in which confidence in the predictions needs to be quantified or out of distribution detection must be performed.

4.3.1 Confidence characterization

The problem of overconfidence of modern neural network has been intensively studied [34]. A parametric classifier is an object that given an input, associated to an unknown true class label y , produces the output tuple y_{pred}, p_{pred} where y_{pred} is the declared output class and p_{pred} is the associated confidence of the prediction. A classifier is said to be calibrated if

$$p_{y_{pred}|p_{pred}}(\hat{y}|\hat{p}) = \hat{p}. \quad (4.21)$$

If a model has good calibration, it is possible to decide to discard predictions that have a low associated confidence and request the intervention of a human expert. Consider for example the application case of automatized, personalized health care, such as [41]. It is clear that in such an application having over-confident wrong predictions is extremely dangerous. Another application domain in which calibration of confidence is crucial is the field of autonomous driving [9], where safety [58] is of paramount importance.

4.3.2 Out of distribution detection

Another important practical problem that affects complex models is the detection of out of distribution inputs in the testing phase. The underlying assumption of all machine learning models is in fact that new observations are sampled from the same distribution of the training dataset. Crucially, this is not always the case, either in a benevolent or malevolent scenario. If a model is trained on thousands of x-ray images to classify whether cancer is present or not and during testing phase a picture of a cat is presented to the network, there is not built-in, straightforward mechanism with which the network can tell us *"I don't know what this is"*.

The problem has been studied and framed extensively in the subfield of AI that deals with safety [3], and several practical examples of spectacular failures of deep models that miss out of distribution inputs are presented in several references such as [64] or [60].

4.4 The problem of sampling from $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$: Markov Chain Monte Carlo

In the previous section we hid in a simple sentence a fairly complicated operation: sampling from $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$, where, again, $\mathcal{D} = \{\mathbf{U}_i\}_{i=1}^N$ is data-set of m -dimensional observations. While for low dimensional random variable several efficient techniques exist for generating samples from a given distribution, see for example [18], when the dimensionality of the problem or the complexity of the probability density function increase, many of the existing techniques become unusable or extremely unefficient.

In general, not even a closed analytic form for the expression of $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$ is available. The usage of Markov Chain Monte Carlo (MCMC) techniques for generating samples drawn from $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$ is a general, powerful tool to solve the problem.

The idea, in its general (and simplified) form, is the usage of a generic stochastic machine/algorithm that, when receiving an input random variable \mathbf{x}_{in} , stochastically produces an output random variable \mathbf{x}_{out} according to a conditional probability density function $q_{\mathbf{x}_{out}|\mathbf{x}_{in}}(\hat{\mathbf{x}}_{out}|\hat{\mathbf{x}}_{in})$. The conditional probability is chosen such that the desired probability $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$ is an eigenfunction of the Markov transition kernel, i.e.

$$p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) = \int q_{\mathbf{x}_{out}|\mathbf{x}_{in}}(\hat{\mathbf{x}}|\tilde{\mathbf{x}})p_{\mathbf{x}|\mathcal{D}}(\tilde{\mathbf{x}}|\hat{\mathcal{D}})d\tilde{\mathbf{x}} \quad (4.22)$$

The informal idea is the following: we start with samples from a generic distribution (possibly easy to be sampled from) $r_{\mathbf{x}_0}(\hat{\mathbf{x}})$. We generate a new sample, by using the stochastic machine, whose distribution is going to be

$$r_{\mathbf{x}_1}(\hat{\mathbf{x}}) = \int q_{\mathbf{x}_{out}|\mathbf{x}_{in}}(\hat{\mathbf{x}}|\tilde{\mathbf{x}})r_{\mathbf{x}_0}(\tilde{\mathbf{x}})d\tilde{\mathbf{x}}, \quad (4.23)$$

then a new sample using as input the generated sample, whose distribution is $r_2(\hat{\mathbf{x}}) = \int q_{\mathbf{x}_{out}|\mathbf{x}_{in}}(\hat{\mathbf{x}}|\tilde{\mathbf{x}})r_{\mathbf{x}_1}(\tilde{\mathbf{x}})d\tilde{\mathbf{x}}$, and so on iteratively

$$r_{\mathbf{x}_N}(\hat{\mathbf{x}}) = \int q_{\mathbf{x}_{out}|\mathbf{x}_{in}}(\hat{\mathbf{x}}|\tilde{\mathbf{x}})r_{\mathbf{x}_{N-1}}(\tilde{\mathbf{x}})d\tilde{\mathbf{x}}. \quad (4.24)$$

If the process converges to a stationary regime ($\lim_{N \rightarrow \infty} r_{\mathbf{x}_N}(\hat{\mathbf{x}}) - r_{\mathbf{x}_{N-1}}(\hat{\mathbf{x}}) = 0$), then asymptotically

$$\lim_{N \rightarrow \infty} r_{\mathbf{x}_N}(\mathbf{x}) = p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}). \quad (4.25)$$

There are many possible theoretical ways to build Markov chains with different properties. A simple and clean mechanism to obtain samples from any given probability density function $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$ (in the limit of $\eta \rightarrow 0$) is to use the following transition probability

$$q_{\mathbf{x}_{out}|\mathbf{x}_{in}}(\hat{\mathbf{x}}[n+1]|\hat{\mathbf{x}}[n]) \propto \exp\left(-\frac{1}{4\eta} \left\| \hat{\mathbf{x}}[n+1] - (\hat{\mathbf{x}}[n] + \eta \nabla \log(p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}[n]|\hat{\mathcal{D}}))) \right\|^2\right) \quad (4.26)$$

Equation (4.26) corresponds to the transition probability of a discretised approximation of a Langevin dynamic. The reason why this transition kernel provides useful samples will be clarified in the following subsection.

4.4.1 Simulated Langevin dynamics

From a practical point of view, the stochastic machine that guarantees having transition probability (4.26), is simply obtained using

$$\mathbf{x}[n] - \mathbf{x}[n-1] = \nabla \log(p_{\mathbf{x}|\mathcal{D}}(\mathbf{x}[n-1]|\hat{\mathcal{D}}))\eta + \sqrt{2\mathbf{n}}[n], \quad (4.27)$$

where $\mathbf{n}[n] \sim \mathcal{N}(\mathbf{0}, \eta \mathbf{I})$. In the limit of small η , we can interpret (4.27) as the Euler discretisation scheme of the following SDE

$$d\mathbf{x}(t) = \nabla \log \left(p_{\mathbf{x}|\mathcal{D}}(\mathbf{x}(t)|\hat{\mathcal{D}}) \right) dt + \sqrt{2} d\mathbf{W}(t) \quad (4.28)$$

The usage of simulated Langevin dynamics to collect samples from generic probability density function stems from the exploitation of the property described by (3.14). Since the considered SDE has identity diffusion matrix $\Sigma(\hat{\mathbf{x}}) = \mathbf{I}$ and the driving force is selected as $\nabla \log \left(p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) \right)$, the stationary distribution of the stochastic process $\mathbf{x}(t)$ is equal to $\exp(\log \left(p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}) \right)) = p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$.

By simulating the SDE (4.28) for $n_0 + N_{\text{MC}}$ time steps ($n_0 \gg 1$ to ensure that the process is in the stationary regime) and storing the samples for $n \in [n_0, n_0 + N_{\text{MC}} - 1]$, it is possible to estimate the integral (4.4) using the scheme described in (4.5).

If the analogy between (4.27) and (4.28) is intuitive, we should not forget that a true continuous time simulation of the SDE is not doable on a digital computer. It can be proven that for vanishing discretisation steps η , the probability density function of the discrete time process described by (4.27) and the probability density function of the one described by (4.28), sampled with frequency $\frac{1}{\Delta t}$, will converge:

$$\lim_{\Delta t \rightarrow 0} p_{\mathbf{x}[n]}(\hat{\mathbf{x}}) - p_{\mathbf{x}(n\Delta t)}(\hat{\mathbf{x}}) = 0 \quad (4.29)$$

For arbitrary small, but finite, η we incur in the discretisation error, i.e. the discrepancy among densities induced by the fact that arbitrary small but non zero step sizes are considered. We do not consider such errors in this dissertation.

4.4.2 Gradient descent with noise to sample from posterior

We can safely assert that (4.27) is the bridge between two (seemingly) unrelated areas: Montecarlo Integration and Optimization. Its role in Montecarlo Integration has been understood under the lens of Langevin Dynamics. On the other side, (4.27) can also be understood as a maximization process using gradient ascent with the injection of noise in the updates.

Since gradient computation through automatic differentiation is already implemented in many modern software packages, it is evident why the usage of (4.27) for collecting samples has practical advantages: it is sufficient to inject noise while optimizing using gradient ascent and obtain useful samples for the computation of Montecarlo integrals. Often minimisation is preferred to maximization, thus it is practice to consider instead gradient descent on minus the logarithm of probability density

$$\mathbf{x}[n] - \mathbf{x}[n - 1] = -\nabla \left(-\log \left(p_{\mathbf{x}|\mathcal{D}}(\mathbf{x}[n]|\hat{\mathcal{D}}) \right) \right) \eta + \sqrt{2} \mathbf{n}[n], \quad (4.30)$$

In the particular case of posterior sampling for machine learning models, it is then sufficient a likelihood model $p(\cdot|\hat{\mathbf{x}})$, and a prior $p_{\mathbf{x}}(\hat{\mathbf{x}})$, to implement gradient descent with gradient equal to

$$-\nabla \log(p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})) = -\nabla \log\left(\frac{p_{\mathcal{D}|\mathbf{x}}(\hat{\mathcal{D}}|\hat{\mathbf{x}}) p_{\mathbf{x}}(\hat{\mathbf{x}})}{p_{\mathcal{D}}(\hat{\mathcal{D}})}\right) = -\nabla \log(p_{\mathcal{D}|\mathbf{x}}(\hat{\mathcal{D}}|\hat{\mathbf{x}}) p_{\mathbf{x}}(\hat{\mathbf{x}})) \quad (4.31)$$

and inject noise with identity diffusion matrix $\Sigma(\mathbf{x}) = \mathbf{I}$, to produce samples drawn from $p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}})$. Conceptually, we could then consider the problem of obtaining samples from a given distribution solved. As will be clear in the next Chapter, however, the computational cost of obtaining such samples rapidly becomes unbearable and different solutions must be adopted.

Chapter 5

SG-MCMC methods

In this chapter we treat the computational problem of full gradient based methods and their solution with the mini-batches approach (Stochastic Gradient Markov Chain MonteCarlo, SG-MCMC). In Section 5.1 we explain the unfeasibility of the full gradient approach for modern datasets, in Section 5.2 we introduce the mini-batch approach, that if on one side solves the computational problem, on the other introduces additional stochastic noise due to the random subsampling. Sections 5.3 and 5.4 carefully treat the existing landscape of sampling algorithms based on modifications of gradient descent without and with moment respectively.

5.1 The computational problem

If we look back at (4.30), we can argue that as long as we are able to compute explicitly the gradient of logarithm of probability density, we should be able to collect samples from any posterior by simply implementing the iterative scheme. If we consider the computational complexity we immediately notice that for modern large scale problems the above mentioned naive approach is unfeasible.

We start our discussion by looking back at (4.31). We develop our work by using an unnormalized version of the logarithm of the posterior density, by expressing the negative logarithm of the joint distribution of the data-set \mathcal{D} and d -dimensional parameter vector \mathbf{x} as:

$$-f(\hat{\mathbf{x}}) = \log(p_{\mathcal{D}|\mathbf{x}}(\hat{\mathcal{D}}|\hat{\mathbf{x}}) p_{\mathbf{x}}(\hat{\mathbf{x}})) = \sum_{i=1}^N \log p_{u_i|\mathbf{x}}(\hat{u}_i|\hat{\mathbf{x}}) + \log p_{\mathbf{x}}(\hat{\mathbf{x}}). \quad (5.1)$$

Notice, as already derived, that $-\nabla f(\hat{\mathbf{x}}) = \nabla \log(p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}))$ since the probability density for the dataset $p_{\mathcal{D}}(\hat{\mathcal{D}})$ is independent of $\hat{\mathbf{x}}$.

Having expressed the gradient in terms of single datapoints of the dataset, we notice that unluckily, the computational cost of a single evaluation of the gradient

$\nabla \log(p_{\mathbf{x}|\mathcal{D}}(\hat{\mathbf{x}}|\hat{\mathcal{D}}))$, scales with the number of datapoints N of the dataset. For modern applications for which N is in the order of tens of thousands (if not millions or billions) this cost rapidly becomes unaffordable.

As in the classical optimization literature the problem has been overcome using mini-batch approaches, the same principle can be applied to reduce the computational cost of the samples generation procedure. While they offer benefits from the computational point of view, the intrinsic stochasticity generated by subsampling further introduce noise in the simulation loop. The remaining parts of this thesis are focused exactly on quantifying the interaction effect between the minibatch induced noise and the injected (simulated) noise.

5.2 Stochastic Gradient

For computational efficiency, we use a mini-batch stochastic gradient $\mathbf{g}(\hat{\mathbf{x}})$, which guarantees that the estimated gradient is an unbiased estimate of the true gradient $\nabla f(\hat{\mathbf{x}})$. Starting from the gradient of the logarithm of the posterior density:

$$-\nabla f(\hat{\mathbf{x}}) = \sum_{i=1}^N \nabla \log p_{\mathbf{u}_i|\mathbf{x}}(\hat{\mathbf{u}}_i|\hat{\mathbf{x}}) + \nabla \log p_{\mathbf{x}}(\hat{\mathbf{x}}),$$

it is possible to define its *minibatch* version by computing the gradient on a random subset \mathcal{I}_{N_b} with cardinality N_b of all the indexes. The minibatch gradient $\mathbf{g}(\hat{\mathbf{x}})$ (equivalently called stochastic gradient, SG) is computed as

$$-\mathbf{g}(\hat{\mathbf{x}}) = \frac{N}{N_b} \sum_{i \in \mathcal{I}_{N_b}} \nabla \log p_{\mathbf{u}_i|\mathbf{x}}(\hat{\mathbf{u}}_i|\hat{\mathbf{x}}) + \nabla \log p_{\mathbf{x}}(\hat{\mathbf{x}}),$$

By simple calculations it is possible to show that the estimation is unbiased. Consider for simplicity the case of sampling with replacement, the case without replacement requires a slightly more technical proof. The expected value, over the indices, of the stochastic gradient is:

$$\begin{aligned} E_{\mathcal{I}_{N_b}}[-\mathbf{g}(\hat{\mathbf{x}})] &= E_{\mathcal{I}_{N_b}}\left[\frac{N}{N_b} \sum_{j \in \mathcal{I}_{N_b}} \nabla \log p_{\mathbf{u}_j|\mathbf{x}}(\mathbf{u}_j|\hat{\mathbf{x}}) + \nabla \log p_{\mathbf{x}}(\hat{\mathbf{x}})\right] = \\ &= \frac{N}{N_b} E_{\mathcal{I}_{N_b}}\left[\sum_{j \in \mathcal{I}_{N_b}} \nabla \log p_{\mathbf{u}_j|\mathbf{x}}(\mathbf{u}_j|\hat{\mathbf{x}})\right] + \nabla \log p_{\mathbf{x}}(\hat{\mathbf{x}}) = \\ &= \frac{N}{N_b} \frac{N_b}{N} \sum_{i=1}^N \nabla \log p_{\mathbf{u}_i|\mathbf{x}}(\hat{\mathbf{u}}_i|\hat{\mathbf{x}}) + \nabla \log p_{\mathbf{x}}(\hat{\mathbf{x}}) = -\nabla f(\hat{\mathbf{x}}) \end{aligned}$$

The estimation error due to the usage of mini batches instead of the full dataset, i.e. $\mathbf{g}(\hat{\mathbf{x}}) - \nabla f(\hat{\mathbf{x}})$, has covariance equal to

$$E\left[(\mathbf{g}(\hat{\mathbf{x}}) - \nabla f(\hat{\mathbf{x}}))(\mathbf{g}(\hat{\mathbf{x}}) - \nabla f(\hat{\mathbf{x}}))^{\top}\right] = 2\mathbf{B}(\hat{\mathbf{x}}). \quad (5.2)$$

An analytical expression of $\mathbf{B}(\hat{\mathbf{x}})$ is out of reach for generic parametric models. This is, both from a theoretical and practical point of view, one of the biggest problems in stochastic gradient based Montecarlo integrations.

If the minibatch size is large enough, and the variance of the single components is finite¹, invoking the central limit theorem, we can state that the minibatch gradient is normally distributed:

$$\mathbf{g}(\hat{\mathbf{x}}) \sim \mathcal{N}(\nabla f(\hat{\mathbf{x}}), 2\mathbf{B}(\hat{\mathbf{x}})). \quad (5.3)$$

where the matrix $\mathbf{B}(\hat{\mathbf{x}})$ denotes the SG noise covariance, which depends on the parametric model, the data distribution and the mini-batch size.

There are structural similarities between SG-MCMC algorithms and stochastic optimization methods, and both can be used to draw samples from posterior distributions. In what follows, we use a unified notation to compare many existing algorithms in light of the role played by their noise components [15, 57, 55].

Stochastic gradient descent (SGD), with and without momentum, can be studied through the following stochastic differential equation (SDE)[29, 48, 53], when the learning rate η is small enough:

$$d\mathbf{z}(t) = \mathbf{s}(\mathbf{z}(t))dt + \sqrt{2\eta\mathbf{D}(\mathbf{z}(t))}d\mathbf{w}(t). \quad (5.4)$$

We use a generic form of the SDE, with variable \mathbf{z} instead of $\hat{\mathbf{x}}$, that accommodates SGD variants, with and without momentum. By doing this, we will be able to easily cast the expression for the two cases in what follows.

Definition 1. A distribution $\rho(\hat{\mathbf{z}}) \propto \exp(-\phi(\hat{\mathbf{z}}))$ is said to be a **stationary** distribution for the SDE of the form (5.4), if and only if it satisfies the following Fokker-Planck equation (FPE):

$$0 = \text{Tr} \left\{ \nabla \left[-\mathbf{s}(\hat{\mathbf{z}})^\top \rho(\hat{\mathbf{z}}) + \nabla^\top (\mathbf{D}(\hat{\mathbf{z}}) \rho(\hat{\mathbf{z}})) \right] \right\}. \quad (5.5)$$

Note that in general, the stationary distribution does not converge to the desired posterior distribution, i.e. $\phi(\hat{\mathbf{z}}) \neq f(\hat{\mathbf{z}})$, as shown by Chaudhari and Soatto [13]. Additionally, given an initial condition for $\mathbf{z}(t)$, its distribution is going to converge to $\rho(\hat{\mathbf{z}})$ only for $t \rightarrow \infty$. In practice, we observe the SDE dynamics for a finite amount of time: then, we declare that the process is approximately in the stationary regime once the negative log probability $f(\hat{\mathbf{x}})$ has reached low and stable values. In this chapter we equivalently refer to $f(\hat{\mathbf{x}})$ as potential, since the gradient dynamics can be interpreted as the relaxation of a physical system described by coordinates $\hat{\mathbf{x}}$ in a potential $f(\hat{\mathbf{x}})$.

Next, we briefly overview known approaches to Bayesian posterior sampling, and interpret them as variants of an SGD process, using the FPE formalism.

¹we will actually challenge this assumption in later chapters

5.3 Gradient methods without momentum

In this section we generalize SGD (withouth momentum), by considering preconditioning matrices and noise injection from the user. The generalized updated rule of SGD, described as a discrete-time stochastic process $\hat{\mathbf{x}}[n]$, writes as:

$$d\mathbf{x}[n] = -\eta\mathbf{P}(\mathbf{x}[n-1])(\mathbf{g}(\mathbf{x}[n-1]) + \mathbf{w}[n]), \quad (5.6)$$

where $d\mathbf{x}[n] = \mathbf{x}[n+1] - \hat{\mathbf{x}}[n]$, $\mathbf{P}(\hat{\mathbf{x}})$ is a user-defined preconditioning matrix, assumed to be symmetric, and $\mathbf{w}[n]$ is a user injected noise term, distributed as $\mathbf{w}[n] = \mathbf{w}(\mathbf{x}[n]) \sim \mathcal{N}(\mathbf{0}, 2\mathbf{C}(\hat{\mathbf{x}}))$, with a user-defined covariance matrix $\mathbf{C}(\hat{\mathbf{x}})$.

The role of the two modifications is easily understood: preconditioning is a well known method to speed up and stabilize gradient based optimization algorithms, while the user injected noise is introduced to counteract the fact that SGD, in general, does not converge to the desired potential due to the non isotropic noise covariance matrix $\mathbf{B}(\hat{\mathbf{x}})$.

Since $\mathbf{g}(\hat{\mathbf{x}}) \sim \mathcal{N}(\nabla f(\hat{\mathbf{x}}), 2\mathbf{B}(\hat{\mathbf{x}}))$ we can rewrite (5.6) by collecting together the two noise sources (minibatch and user injected) as:

$$d\mathbf{x}[n] = -\eta\mathbf{P}(\mathbf{x}[n-1])(\nabla f(\mathbf{x}[n-1]) + \mathbf{w}'[n]),$$

where we defined $\mathbf{w}'[n] \sim \mathcal{N}(0, 2\Sigma(\hat{\mathbf{x}}))$, with $\Sigma(\hat{\mathbf{x}}) = \mathbf{B}(\hat{\mathbf{x}}) + \mathbf{C}(\hat{\mathbf{x}})$.

Moreover, if we separate deterministic and random components we can equivalently write:

$$\begin{aligned} d\mathbf{x}[n] &= -\eta\mathbf{P}(\mathbf{x}[n-1])\nabla f(\mathbf{x}[n-1]) + \eta\mathbf{P}(\mathbf{x}[n-1])\mathbf{w}'[n] = \\ &= -\eta\mathbf{P}(\mathbf{x}[n-1])\nabla f(\mathbf{x}[n-1]) + \sqrt{2\eta\mathbf{P}^2(\mathbf{x}[n-1])\Sigma(\mathbf{x}[n-1])}\mathbf{v}[n] \end{aligned}$$

where $\mathbf{v}[n] \sim \mathcal{N}(0, \eta\mathbf{I})$.

When η is small enough ($\eta \rightarrow dt$) we can interpret the above equation as the discrete time simulation (Euler discretisation) of the following SDE [29]:

$$d\mathbf{x}(t) = -\mathbf{P}(\mathbf{x}(t))\nabla f(\mathbf{x}(t))dt + \sqrt{2\eta\mathbf{P}(\mathbf{x}(t))^2\Sigma(\mathbf{x}(t))}d\mathbf{w}(t). \quad (5.7)$$

where $\mathbf{w}(t)$ is a d -dimensional Brownian motion.

We denote by $\mathbf{C}(\hat{\mathbf{x}})$ the covariance of the *injected noise* and $\Sigma(\hat{\mathbf{x}})$ the *composite noise* covariance. Note that $\Sigma(\hat{\mathbf{x}}) = \mathbf{B}(\hat{\mathbf{x}}) + \mathbf{C}(\hat{\mathbf{x}})$ combines the SG and the injected noise.

We define the stationary distribution of the SDE in Eq. 5.7 as $\rho(\hat{\mathbf{x}}) \propto \exp(-\phi(\hat{\mathbf{x}}))$. Note that when $\mathbf{C} = \mathbf{0}$, the potential $\phi(\hat{\mathbf{x}})$ differs from the desired posterior $f(\hat{\mathbf{x}})$ [13].

5.3.1 Conditions for convergence to desired potential

The following theorem, which is an adaptation of known results in light of our formalism, states the conditions for which the *noisy* SGD converges to the true posterior distribution.

Theorem 1. *Consider dynamics of the form (5.7) and define the stationary distribution $\rho(\hat{\mathbf{x}}) \propto \exp(-\phi(\hat{\mathbf{x}}))$. If*

$$\nabla^\top (\boldsymbol{\Sigma}(\hat{\mathbf{x}})^{-1}) = \mathbf{0}^\top \quad \text{and} \quad \eta \mathbf{P}(\hat{\mathbf{x}}) = \boldsymbol{\Sigma}(\hat{\mathbf{x}})^{-1}, \quad (5.8)$$

then $\phi(\hat{\mathbf{x}}) = f(\hat{\mathbf{x}})$.

Proof of 1. The stationary distribution of the above SDE, $\rho(\hat{\mathbf{x}}) \propto \exp(-\phi(\hat{\mathbf{x}}))$, satisfies the following FPE:

$$0 = \text{Tr} \left\{ \nabla \left[\nabla^\top (f(\hat{\mathbf{x}})) \mathbf{P}(\hat{\mathbf{x}}) \rho(\hat{\mathbf{x}}) + \eta \nabla^\top (\mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}}) \rho(\hat{\mathbf{x}})) \right] \right\},$$

that we rewrite, since $\nabla^\top (\mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}}) \rho(\hat{\mathbf{x}})) = \nabla^\top (\mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}})) \rho(\hat{\mathbf{x}}) - \nabla^\top (\phi(\hat{\mathbf{x}})) \mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}}) \rho(\hat{\mathbf{x}})$, as

$$0 = \text{Tr} \left\{ \nabla \left[\nabla^\top (f(\hat{\mathbf{x}})) \mathbf{P}(\hat{\mathbf{x}}) \rho(\hat{\mathbf{x}}) - \eta \nabla^\top (\phi(\hat{\mathbf{x}})) \mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}}) \rho(\hat{\mathbf{x}}) + \eta \nabla^\top (\mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}})) \rho(\hat{\mathbf{x}}) \right] \right\}.$$

The above equation is verified with $\nabla f(\hat{\mathbf{x}}) = \nabla \phi(\hat{\mathbf{x}})$ if

$$\begin{cases} \nabla^\top (\mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}})) = \mathbf{0} \\ \eta \mathbf{P}(\hat{\mathbf{x}})^2 \boldsymbol{\Sigma}(\hat{\mathbf{x}}) = \mathbf{P}(\hat{\mathbf{x}}) \rightarrow \eta \mathbf{P}(\hat{\mathbf{x}}) = \boldsymbol{\Sigma}(\hat{\mathbf{x}})^{-1} \end{cases}$$

that proves Theorem 1.

5.3.2 Methods based on SGD

Stochastic Gradient Langevin Dynamics (SGLD) [95] is a simple approach to satisfy Eq. (5.8); it uses no preconditioning, $\mathbf{P}(\hat{\mathbf{x}}) = \mathbf{I}$, and sets the injected noise covariance to $\mathbf{C}(\hat{\mathbf{x}}) = \eta^{-1} \mathbf{I}$. In the limit for $\eta \rightarrow 0$, it holds that $\boldsymbol{\Sigma}(\hat{\mathbf{x}}) = \mathbf{B}(\hat{\mathbf{x}}) + \eta^{-1} \mathbf{I} \simeq \eta^{-1} \mathbf{I}$. Then, $\nabla^\top (\boldsymbol{\Sigma}(\hat{\mathbf{x}})^{-1}) = \eta \nabla^\top \mathbf{I} = \mathbf{0}^\top$, and $\eta \mathbf{P}(\hat{\mathbf{x}}) = \boldsymbol{\Sigma}(\hat{\mathbf{x}})^{-1}$. While SGLD succeeds in (asymptotically) generating samples from the true posterior, its mixing rate is unnecessarily slow, due to the extremely small learning rate [1].

An extension to SGLD is Stochastic Gradient Fisher Scoring (SGFS) [1], which can be tuned to switch between sampling from an approximate posterior, using a non-vanishing learning rate, and the true posterior, by annealing the learning rate to zero. SGFS uses preconditioning, $\mathbf{P}(\hat{\mathbf{x}}) \propto \mathbf{B}(\hat{\mathbf{x}})^{-1}$. In practice, however, $\mathbf{B}(\hat{\mathbf{x}})$ is ill conditioned for complex models such as deep neural networks. Then, many of its eigenvalues are almost zero [13], and computing $\mathbf{B}(\hat{\mathbf{x}})^{-1}$ is problematic. An

in depth analysis of SGFS reveals that conditions (5.8) would be met with a non-vanishing learning rate only if, at convergence, $\nabla^\top(\mathbf{B}(\hat{\mathbf{x}})^{-1}) = \mathbf{0}^\top$, which would be trivially true if $\mathbf{B}(\hat{\mathbf{x}})$ was constant. However, recent work [19, 30] suggests that this condition is difficult to justify for deep neural networks.

The Stochastic Gradient Riemannian Langevin Dynamics (SGRLD) algorithm [66] extends SGFS to the setting in which $\nabla^\top(\mathbf{B}(\hat{\mathbf{x}})^{-1}) \neq \mathbf{0}^\top$. The process dynamic is adjusted by adding the term $\nabla^\top(\mathbf{B}(\hat{\mathbf{x}})^{-1})$. However, the term $\nabla^\top(\mathbf{B}(\hat{\mathbf{x}})^{-1})$ has not a clear estimation procedure, restricting SGRLD to cases where it can be computed analytically.

The work by [57] investigates constant-rate SGD (with no injected noise), and determines analytically the learning rate and preconditioning that minimise the Kullback–Leibler (KL) divergence between an approximation and the true posterior. Moreover, it shows that the preconditioning used in SGFS is optimal, in the sense that it converges to the true posterior, when $\mathbf{B}(\hat{\mathbf{x}})$ is constant and the true posterior has a quadratic form.

In summary, to claim convergence to the true posterior distribution, existing approaches require either vanishing learning rates or assumptions on the SG noise covariance that are difficult to verify in practice, especially when considering deep models. We instead propose a novel practical method, that induces isotropic SG noise and thus satisfies Theorem 1. We determine analytically a fixed learning rate and we require weaker assumptions on the loss shape.

5.4 Gradient methods with momentum

Momentum-corrected methods emerge as a natural extension to SGD approaches. The general set of update equations for (discrete-time) momentum-based algorithms is:

$$\begin{cases} d\mathbf{x}[n] = \eta \mathbf{P}(\mathbf{x}[n-1]) \mathbf{M}^{-1} \mathbf{r}[n-1] \\ d\mathbf{r}[n] = -\eta \mathbf{A}(\mathbf{x}[n-1]) \mathbf{M}^{-1} \mathbf{r}[n-1] - \eta \mathbf{P}(\mathbf{x}[n-1]) (\mathbf{g}(\mathbf{x}[n-1]) + \mathbf{w}[n]), \end{cases}$$

where $\mathbf{P}(\hat{\mathbf{x}})$ is a preconditioning matrix, the \mathbf{M} is called the mass matrix and $\mathbf{A}(\hat{\mathbf{x}})$ is the friction matrix, as shown by [16, 62]. The matrices \mathbf{M} , \mathbf{A} are called respectively mass and friction matrix to underline the formal equivalence between the described system of equations and the time evolution of the position and velocity of a point object in a gravitational potential when considering friction. Obviously it is only an analogy and the reader is not required to have any knowledge of such systems, the two matrices can be simply understood as further user defined degrees of freedom. Similarly to the first order counterpart, the noise term is distributed as $\mathbf{w}[n] \sim \mathcal{N}(\mathbf{0}, 2\mathbf{C}(\hat{\mathbf{x}}))$.

Similarly to the case without momentum, we rewrite the second equation of the

system as

$$\begin{aligned} \mathbf{d}\mathbf{r}[n] = & -\eta\mathbf{A}(\mathbf{x}[n-1])\mathbf{M}^{-1}\mathbf{r}[n-1] - \eta\mathbf{P}(\mathbf{x}[n-1])(\mathbf{g}(\mathbf{x}[n-1]) + \mathbf{w}[n]) = \\ & -\eta\mathbf{A}(\mathbf{x}[n-1])\mathbf{M}^{-1}\mathbf{r}[n-1] - \eta\mathbf{P}(\mathbf{x}[n-1])\nabla f(\mathbf{x}[n-1]) + \sqrt{2\eta}\mathbf{P}^2(\mathbf{x}[n-1])\boldsymbol{\Sigma}(\mathbf{x}[n-1])\boldsymbol{\nu}[n] \end{aligned}$$

where again $\boldsymbol{\nu}[n] \sim \mathcal{N}(0, \eta\mathbf{I})$. If we define the super-variable $\mathbf{z} = [\mathbf{x}, \mathbf{r}]^\top$, we can rewrite the system as:

$$\mathbf{d}\mathbf{z}[n] = -\eta \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\mathbf{x}[n-1]) \\ \mathbf{P}(\mathbf{x}[n-1]) & \mathbf{A}(\mathbf{x}[n-1]) \end{bmatrix} \mathbf{s}(\mathbf{z}[n-1]) + \sqrt{2\eta}\mathbf{D}(\mathbf{z}[n-1])\boldsymbol{\nu}[n]$$

where $\mathbf{s}(\hat{\mathbf{z}}) = \begin{bmatrix} \nabla f(\hat{\mathbf{x}}) \\ \mathbf{M}^{-1}\hat{\mathbf{r}} \end{bmatrix}$, $\mathbf{D}(\hat{\mathbf{z}}) = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}(\hat{\mathbf{x}})^2\boldsymbol{\Sigma}(\hat{\mathbf{x}}) \end{bmatrix}$ and $\boldsymbol{\nu}[n] \sim \mathcal{N}(0, \eta\mathbf{I})$.

As the learning rate goes to zero ($\eta \rightarrow dt$), similarly to the previous case, we can interpret the above difference equation as a discretisation of the following FPE

$$d\mathbf{z}(t) = - \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\mathbf{x}(t)) \\ \mathbf{P}(\mathbf{x}(t)) & \mathbf{A}(\mathbf{x}(t)) \end{bmatrix} \mathbf{s}(\mathbf{z}(t)) + \sqrt{2\eta}\mathbf{D}(\mathbf{z}(t))d\mathbf{w}(t)$$

Equivalently, the system of stochastic equations that describe the continuous-time system dynamics is:

$$\begin{cases} d\mathbf{x}(t) = \mathbf{P}(\mathbf{x}(t))\mathbf{M}^{-1}\mathbf{r}(t)dt \\ d\mathbf{r}(t) = -(\mathbf{A}(\mathbf{x}(t))\mathbf{M}^{-1}\mathbf{r}(t) + \mathbf{P}(\mathbf{x}(t))\nabla f(\mathbf{x}(t)))dt + \\ \sqrt{2\eta}\mathbf{P}(\mathbf{x}(t))^2\boldsymbol{\Sigma}(\mathbf{x}(t))d\mathbf{w}(t). \end{cases} \quad (5.9)$$

where $\mathbf{P}(\hat{\mathbf{x}})^2 = \mathbf{P}(\hat{\mathbf{x}})\mathbf{P}(\hat{\mathbf{x}})$ and we assume $\mathbf{P}(\hat{\mathbf{x}})$ to be symmetric.

5.4.1 Conditions for convergence to the desired potential (II)

The theorem hereafter describes the conditions for which noisy SGD with momentum converges to the true posterior distribution.

Theorem 2. *Consider dynamics of the form (5.9) and define the stationary distribution for $x(t)$ as $\rho(\hat{\mathbf{x}}) \propto \exp(-\phi(\hat{\mathbf{x}}))$. If*

$$\nabla^\top \mathbf{P}(\hat{\mathbf{x}}) = \mathbf{0}^\top \quad \text{and} \quad \mathbf{A}(\hat{\mathbf{x}}) = \eta\mathbf{P}(\hat{\mathbf{x}})^2\boldsymbol{\Sigma}(\hat{\mathbf{x}}), \quad (5.10)$$

then $\phi(\hat{\mathbf{x}}) = f(\hat{\mathbf{x}})$.

Proof of 2. As before we assume that the stationary distribution has form $\rho(\hat{z}) \propto \exp(-\phi(\hat{z}))$. The corresponding FPE is:

$$0 = \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) + \eta \left(\nabla^\top (\mathbf{D}(\hat{z})\rho(\hat{z})) \right) \right) \right).$$

Notice that since $\nabla^\top \mathbf{D}(z) = 0$ we can rewrite:

$$\begin{aligned} 0 &= \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) + \eta \nabla^\top (\rho(\hat{z})) \mathbf{D}(\hat{z}) \right) \right) \\ &= \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) - \eta \nabla^\top (\phi(\hat{z})) \mathbf{D}(\hat{z}) \rho(\hat{z}) \right) \right) \\ &= \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) - \eta \nabla^\top (\phi(\hat{z})) \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}(\hat{x})^2 \boldsymbol{\Sigma}(\hat{x}) \end{bmatrix} \rho(\hat{z}) \right) \right) \end{aligned}$$

that is verified with $\nabla \phi(\hat{z}) = \mathbf{s}(\hat{z})$ if:

$$\begin{cases} \nabla^\top \mathbf{P}(\hat{x}) = \mathbf{0} \\ \mathbf{A}(\hat{x}) = \eta \mathbf{P}(\hat{x})^2 \boldsymbol{\Sigma}(\hat{x}). \end{cases}$$

If $\nabla^\top \mathbf{P}(\hat{x}) = \mathbf{0}$, in fact:

$$\begin{aligned} \text{Tr} \left(\nabla \left(\nabla^\top (\phi(\hat{z})) \rho(\hat{z}) \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{0} \end{bmatrix} \right) \right) &= \nabla^\top \left(\begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{0} \end{bmatrix} \nabla (\phi(\hat{z})) \rho(\hat{z}) \right) = \\ \nabla^\top \left(\begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{0} \end{bmatrix} \right) \nabla (\phi(\hat{z})) \rho(\hat{z}) &+ \text{Tr} \left(\begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{0} \end{bmatrix} \nabla \left(\nabla^\top (\phi(\hat{z})) \rho(\hat{z}) \right) \right) = 0, \end{aligned}$$

since $\nabla^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{0} \end{bmatrix} = \mathbf{0}$ and the second term is zero due to the fact that

$\begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{0} \end{bmatrix}$ is anti-symmetric while $\nabla \left(\nabla^\top (\phi(\hat{z})) \rho(\hat{z}) \right)$ is symmetric.

Thus we can rewrite:

$$\begin{aligned} \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) - \eta \nabla^\top (\phi(\hat{z})) \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}(\hat{x})^2 \boldsymbol{\Sigma}(\hat{x}) \end{bmatrix} \rho(\hat{z}) \right) \right) &= \\ \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) - \nabla^\top (\phi(\hat{z})) \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \eta \mathbf{P}(\hat{x})^2 \boldsymbol{\Sigma}(\hat{x}) \end{bmatrix} \rho(\hat{z}) \right) \right) &= \\ \text{Tr} \left(\nabla \left(\mathbf{s}(\hat{z})^\top \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) - \nabla^\top (\phi(\hat{z})) \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) \right) \right) &= \\ \text{Tr} \left(\nabla \left(\left(\mathbf{s}(\hat{z})^\top - \nabla^\top (\phi(\hat{z})) \right) \begin{bmatrix} \mathbf{0} & -\mathbf{P}(\hat{x}) \\ \mathbf{P}(\hat{x}) & \mathbf{A}(\hat{x}) \end{bmatrix} \rho(\hat{z}) \right) \right) &= 0 \end{aligned}$$

then, $\nabla \phi(\hat{z}) = \mathbf{s}(\hat{z})$, proving Theorem 2.

5.4.2 Methods based on m-SGD

In the naive case, where $\mathbf{P}(\hat{\mathbf{x}}) = \mathbf{I}$, $\mathbf{A}(\hat{\mathbf{x}}) = \mathbf{0}$, $\mathbf{C}(\hat{\mathbf{x}}) = \mathbf{0}$, Eqs. (5.10) are not satisfied and the stationary distribution does not correspond to the true posterior [16]. To generate samples from the true posterior it is sufficient to set $\mathbf{P}(\hat{\mathbf{x}}) = \mathbf{I}$, $\mathbf{A}(\hat{\mathbf{x}}) = \eta\mathbf{B}(\hat{\mathbf{x}})$, $\mathbf{C}(\hat{\mathbf{x}}) = \mathbf{0}$ (as in Eq.(9) in [16]).

Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) [16] suggests that estimating $\mathbf{B}(\hat{\mathbf{x}})$ can be costly. Hence, the injected noise $\mathbf{C}(\hat{\mathbf{x}})$ is chosen such that $\mathbf{C}(\hat{\mathbf{x}}) = \eta^{-1}\mathbf{A}(\hat{\mathbf{x}})$, where $\mathbf{A}(\hat{\mathbf{x}})$ is user-defined. When $\eta \rightarrow 0$, the following approximation holds: $\boldsymbol{\Sigma}(\hat{\mathbf{x}}) \simeq \mathbf{C}(\hat{\mathbf{x}})$. It is then trivial to check that conditions (5.10) hold without the need for explicitly estimating $\mathbf{B}(\hat{\mathbf{x}})$. A further practical reason to avoid setting $\mathbf{A}(\hat{\mathbf{x}}) = \eta\mathbf{B}(\hat{\mathbf{x}})$ is that the computational cost for the operation $\mathbf{A}(\hat{\mathbf{x}})\mathbf{M}^{-1}\mathbf{r}$ has $\mathcal{O}(D^2)$ complexity, whereas if $\mathbf{C}(\hat{\mathbf{x}})$ is diagonal, this is reduced to $\mathcal{O}(D)$. This however, severely slows down the sampling process.

Stochastic Gradient Riemannian Hamiltonian Monte Carlo (SGRHMC) is an extension to SGHMC [55]), which considers a generic, space-varying preconditioning matrix $\mathbf{P}(\hat{\mathbf{x}})$ derived from information geometric arguments [32]. SGRHMC suggests to set $\mathbf{P}(\hat{\mathbf{x}}) = \mathbf{G}(\hat{\mathbf{x}})^{-\frac{1}{2}}$, where $\mathbf{G}(\hat{\mathbf{x}})$ is the Fisher Information matrix. To meet the requirement $\nabla^\top \mathbf{P}(\hat{\mathbf{x}}) = \mathbf{0}^\top$, it includes a correction term, $-\nabla^\top \mathbf{P}(\hat{\mathbf{x}})$. The injected noise is set to $\mathbf{C}(\hat{\mathbf{x}}) = \eta^{-1}\mathbf{I} - \mathbf{B}(\hat{\mathbf{x}})$, consequently $\boldsymbol{\Sigma} = \eta^{-1}\mathbf{I}$, and the friction matrix is set to $\mathbf{A}(\hat{\mathbf{x}}) = \mathbf{P}(\hat{\mathbf{x}})^2$. With all these choices, Theorem 2 is satisfied. While appealing, the main drawbacks of this method are the need for an analytical expression of $\nabla^\top \mathbf{P}(\hat{\mathbf{x}})$, and the assumption for $\mathbf{B}(\hat{\mathbf{x}})$ to be known.

From a practical standpoint, momentum-based methods suffer from the requirement to tune many hyper-parameters, including the learning rate, and the parameters that govern the simulation of a second-order Langevin dynamics.

The method we propose in this work can be applied to momentum-based algorithms; in this case, it could be viewed as an extension of the work in [85], albeit addressing the complex loss landscapes typical of deep neural networks. However, we leave this avenue of research for future work.

Chapter 6

Isotropic SGD: Practical Bayesian Posterior Sampling

In Chapter 5, we provided an overview of the existing methods based on stochastic gradient. As will be clearer at the end of this Chapter, when needing to implement in practice the sampling methods described, many small practical details, such as annealing of the various learning rates, or the need to fine tune any of the hyperparameters, represent the main entry barrier for a wide spread usage of the methods from practitioners whose background is not Bayesian machine learning.

The goal of the method we propose and describe in this chapter, is to come up with a simple algorithm that does not need vanishing learning rates and whose hyperparameters can be theoretically computed, Section 6.2. We first describe its properties in idealised settings, Section 6.3, for which we can guarantee convergence in distribution of the stochastic process to the true posterior, and subsequently propose a simplified, practical method built on top of experimentally validated assumptions, that we describe in Section 6.4.

Despite being the class of fixed learning rates algorithm contained in the general class of time varying learning rates, and thus being its optimal algorithm at best as good (in terms of quality of samples, not sampling efficiency and or convergence speed) as the best time varying learning rate algorithm, we prove and observe that the practical algorithm that we propose converges to the true posterior in a broader range of conditions than other competitors, as discussed in Section 6.5.

The discussion is well complemented by a large scale experimental campaign (Section 6.6) where different models are tested. The method is shown to be robust and competitive with respect to many state of the arts alternatives.

6.1 Multilayer Neural Networks

So far in the thesis, we treated abstractly parametric models, i.e. without specifying actually *how* given and input datapoint the output is computed. In this Chapter, since we perform an experimental validation, we have to specify what models we are considering. In general, we deal with multi-layer neural networks. Far from being our aim a careful introduction to such objects (the interested reader is referred to [33]), it is sufficient to settle down the basic knowledge depicted in Figure 6.1 a schematic representation of a single hidden layer neural network. The initial inputs of the network ($\mathbf{s} = [s_1, \dots, s_d]^\top$) are linearly combined with a matrix \mathbf{X}^1 , whose elements are the parameters $x_{a,b}^1$, i.e. $\mathbf{h}^1 = \mathbf{X}^1 \mathbf{s}$. The result is transformed through a given nonlinearity $r(\cdot)$ and the output $\mathbf{o}^1 = r(\mathbf{h}^1)$ is multiplied by a second matrix \mathbf{X}^2 , with $(\mathbf{X}^2)_{a,b} = x_{a,b}^2$ and the procedure is iterated up to the last layer. This is an oversimplification, but for the purpose of this chapter it is a sufficient one. The reader interested in the details of the architectures can look into the references in the experimental section (Section 6.6). The only important concept in the practical understanding of the proposed method is the one about different layers. In this simple example, the overall set of parameters \mathbf{x} is represented in its single components $x_{i,j}^l$. The layer number is represented with the superscript l .

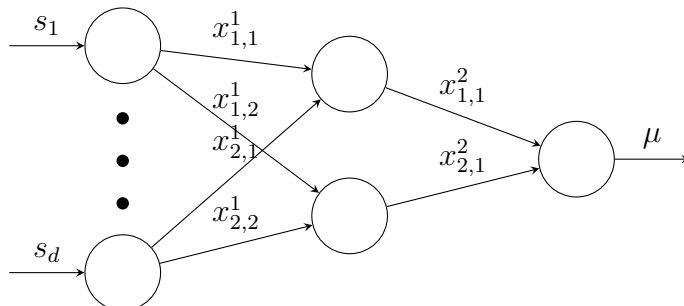


Figure 6.1: Single hidden layer Neural Network

6.2 ISGD

We present a simple and practical approach to inject noise to SGD iterates to perform Bayesian posterior sampling. Our goal is to sample from the true posterior distribution (or approximations thereof) using a *constant* learning rate, and to rely on more lenient assumptions about the geometry of the loss landscape that characterize deep models, compared to previous works.

From Theorem 1 in Chapter 5, observe that $\mathbf{P}(\hat{\mathbf{x}}), \mathbf{\Sigma}(\hat{\mathbf{x}})$ are instrumental to determine the convergence properties of SG methods to the true posterior. We consider the constructive approach of *designing* $\eta \mathbf{P}(\hat{\mathbf{x}})$ to be a constant, diagonal

matrix, constrained to be layer-wise uniform:

$$\eta \mathbf{P}(\hat{\mathbf{x}}) = \mathbf{\Lambda}^{-1} = \text{diag}(\underbrace{[\lambda^{(1)}, \dots, \lambda^{(1)}]}_{\text{layer 1}}, \dots, \underbrace{[\lambda^{(N_l)}, \dots, \lambda^{(N_l)}]}_{\text{layer } N_l})^{-1}. \quad (6.1)$$

Notice that we treat biases and weights of the layers as unique groups of parameters. By properly setting parameters $\lambda^{(l)}$, we achieve the simultaneous result of a non-vanishing learning rate and a well-conditioned preconditioning matrix. This implies a layer-wise learning rate $\eta^{(l)} = \frac{1}{\lambda^{(l)}}$ for the l -th layer, without further preconditioning. We can now state, as a corollary to Theorem 1, that our method guarantees convergence to the true posterior distribution.

Corollary 1. *Given the dynamics of Eq. (5.7) and the stationary distribution $\rho(\hat{\mathbf{x}}) \propto \exp(-\phi(\hat{\mathbf{x}}))$, if $\eta \mathbf{P}(\hat{\mathbf{x}}) = \mathbf{\Lambda}^{-1}$ as in Eq. (6.1), and $\mathbf{C}(\hat{\mathbf{x}}) = \mathbf{\Lambda} - \mathbf{B}(\hat{\mathbf{x}}) \succ \mathbf{0} \forall \hat{\mathbf{x}}$, i.e. it is positive definite, then $\phi(\hat{\mathbf{x}}) = f(\hat{\mathbf{x}})$.*

The requirement $\mathbf{C}(\hat{\mathbf{x}}) \succ \mathbf{0} \forall \hat{\mathbf{x}}$, ensures that the injected noise covariance is valid. The composite noise matrix is equal to $\mathbf{\Sigma}(\hat{\mathbf{x}}) = \mathbf{\Lambda}$. Since $\nabla^\top \mathbf{\Sigma}(\hat{\mathbf{x}}) = \nabla^\top \mathbf{\Lambda} = \mathbf{0}$ and $\eta \mathbf{P}(\hat{\mathbf{x}}) = \mathbf{\Lambda}^{-1}$ by construction, then Theorem 1 is satisfied.

If the above conditions hold, it is simple to show that matrices $\mathbf{P}(\hat{\mathbf{x}})$ and $\mathbf{C}(\hat{\mathbf{x}})$ satisfy Theorem 1. Then, we say that the composite noise covariance $\mathbf{\Sigma}(\hat{\mathbf{x}}) = \mathbf{C}(\hat{\mathbf{x}}) + \mathbf{B}(\hat{\mathbf{x}}) = \text{diag}([\lambda^{(1)}, \dots, \lambda^{(1)}, \dots, \lambda^{(N_l)}, \dots, \lambda^{(N_l)}])$ is *isotropic* within model layers. We set $\mathbf{\Lambda}$ to be, among all valid matrices satisfying $\mathbf{\Lambda} - \mathbf{B}(\hat{\mathbf{x}}) \succ \mathbf{0}$, the smallest, i.e., the one with the smallest λ 's. Indeed, larger $\mathbf{\Lambda}$ induces a small learning rate, thus unnecessarily reducing sampling speed.

6.3 An ideal method.

Now, let's consider an ideal case, in which we assume the SG noise covariance $\mathbf{B}(\hat{\mathbf{x}})$ and $\mathbf{\Lambda}$ to be known in advance. The procedure described in Algorithm 1 illustrates a naive SG method that uses the *injected noise* covariance $\mathbf{C}(\hat{\mathbf{x}})$ to sample from the true posterior.

Algorithm 1 idealised posterior sampling

Sample ($\hat{\mathbf{x}}_0, \mathbf{B}(\hat{\mathbf{x}}), \mathbf{\Lambda}$):

$\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}_0$

▷ Initialize $\hat{\mathbf{x}}$

loop

$\mathbf{g} = \frac{\nabla \tilde{f}(\hat{\mathbf{x}})}{N}$

▷ Compute SG

$\mathbf{C}(\hat{\mathbf{x}})^{\frac{1}{2}} \leftarrow (\mathbf{\Sigma} - \mathbf{B}(\hat{\mathbf{x}}))^{\frac{1}{2}}$

$\mathbf{n} \sim N(0, \mathbf{I})$

$\mathbf{w} \leftarrow \mathbf{C}(\hat{\mathbf{x}})^{\frac{1}{2}} \mathbf{n}$

$\delta \hat{\mathbf{x}} \leftarrow (N \mathbf{\Sigma})^{-1} (\mathbf{g} + \sqrt{2} \mathbf{w})$

$\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} - \delta \hat{\mathbf{x}}$

▷ Update $\hat{\mathbf{x}}$

This simple procedure is guaranteed to generate samples from the true posterior, with a non-vanishing learning rate. Note that instead of computing the gradient of $f(\hat{\mathbf{x}})$, we compute the (mini-batch) gradient of $\frac{f(\hat{\mathbf{x}})}{N}$, similarly to the notation used in [57], that we indicate with $\frac{\nabla \tilde{f}(\hat{\mathbf{x}})}{N}$. However, it cannot be used in practice as $\mathbf{B}(\hat{\mathbf{x}})$ and $\mathbf{\Lambda}$ are unknown. Also, the algorithm is costly, as it requires computing $(\mathbf{\Sigma} - \mathbf{B}(\hat{\mathbf{x}}))^{\frac{1}{2}}$, which requires $\mathcal{O}(d^3)$ operations, and $\mathbf{C}(\hat{\mathbf{x}})^{\frac{1}{2}}$, which costs $\mathcal{O}(d^2)$ multiplications. Next, we describe a practical approach, where we use approximations at the expense of generating samples from the true posterior distribution. Note that [57] suggests to explore a related preconditioning, but does not develop the idea.

6.4 A practical method: Isotropic SGD.

To make the idealised sampling method practical, we require additional assumptions which are milder than what is required by current approaches in the literature, as we explain at the end of this section.

Assumption 1. *The SG noise covariance $\mathbf{B}(\hat{\mathbf{x}})$ can be approximated with a diagonal matrix, i.e., $\mathbf{B}(\hat{\mathbf{x}}) = \text{diag}(\mathbf{b}(\hat{\mathbf{x}}))$. Thus, the noise components are independent [85, 1].*

Assumption 2. *The signal to noise ratio (SNR) of a gradient is small enough such that, in the stationary regime, the un-centered variance of the gradient is a good estimate of the true variance [85, 76]. Hence, combining with assumption 1, $\mathbf{b}(\hat{\mathbf{x}}) \simeq \frac{\mathbb{E}[\mathbf{g}(\hat{\mathbf{x}}) \odot \mathbf{g}(\hat{\mathbf{x}})]}{2}$ (being \odot the elementwise product of two vectors).*

Assumption 3. *In the stationary regime, the maximum of the variances of noise components, layer by layer, are fixed constants (similarly to [100]): $\beta^{(l)} = \max_{j \in I_l} \mathbf{b}_j(\hat{\mathbf{x}})$, where I_l is the set of indexes of parameters belonging to l_{th} layer.*

Note that Assumptions 2 and 3 must hold only in the stationary regime when the process reaches the bottom valley of the loss landscape. Concerning Assumption 3, we actually test in practice two variants: the one, described in the text, where we assume that the maximum of the variances is constant, and a variant in which we assume that the *sum* of the variances is constant layerwise. We refer later to these two variants as the *max* and *sum* variant of Assumption 3 respectively. Notice that a similar assumption is necessary for all SG-MCMC methods. Without assumptions on the boundedness of the variances, it is not possible to claim convergence to the true posterior even in the infinitesimal learning rate regime. The proofs in the text are based on the *max* variant of the assumption, but the results are trivially extended to the second case.

Given our assumptions, and our design choices, it is then possible to show that the optimal (i.e., the smallest possible) $\mathbf{\Lambda} = [\lambda^{(1)}, \dots, \lambda^{(1)}, \dots, \lambda^{(N_l)}, \dots, \lambda^{(N_l)}]$ satisfying Corollary 1 can be obtained as $\lambda^{(l)} = \beta^{(l)}$. The proof is reported hereafter.

By the assumptions the matrix $\mathbf{B}(\hat{\mathbf{x}})$ is diagonal, and consequently $\mathbf{C}(\hat{\mathbf{x}}) = \mathbf{\Lambda} - \mathbf{B}(\hat{\mathbf{x}})$ is diagonal as well. The preconditioner $\mathbf{\Lambda}$ must be chosen to satisfy the positive semi-definite constraint, i.e. $\mathbf{C}(\hat{\mathbf{x}})_{ii} \geq 0 \quad \forall i, \forall \hat{\mathbf{x}}$. Equivalently, we must satisfy $\lambda^{(l)} - \mathbf{b}_j(\hat{\mathbf{x}}) \geq 0 \quad \forall j \in I_l, \forall l, \forall \hat{\mathbf{x}}$, where I_l is the set of indexes of parameters belonging to l th layer. By assumption 3, i.e. $\beta^{(l)} = \max_{k \in I_l} \mathbf{b}_k(\hat{\mathbf{x}})$, to satisfy the positive semi-definite requirement in all cases the minimum valid set of $\lambda^{(l)}$ is determined as $\lambda^{(l)} = \beta^{(l)}$.

Since we cannot assume $\mathbf{B}(\hat{\mathbf{x}})$ to be known, in what follows we discuss two approaches to estimate its components. A simple method to estimate $\mathbf{B}(\hat{\mathbf{x}})$ is as follows (see 6.4.1): we compute $\lambda^{(l)} = \max_{j \in I_l} \mathbf{b}_j(\hat{\mathbf{x}}) = \frac{1}{2} \max_{j \in I_l} (\mathbf{g}_j(\hat{\mathbf{x}})^{(l)})^2$, where $\mathbf{g}(\hat{\mathbf{x}})^{(l)}$ is the portion of stochastic gradient corresponding to the l -th layer. Our estimates can be extended to use a moving average approach. Our empirical validation, however, indicates that this simple method does not produce stable estimates.

Indeed, a shared assumption of SG-MCMC methods is that SG noise is Gaussian. While this assumption can be justified with the C.L.T. for relatively simple models (linear models or simple feed-forward networks), its validity has been challenged in the deep learning domain [81, 80] (see 6.4.1 for a detailed discussion), suggesting that, for complex architectures, the noise distribution is heavy tailed. Then, the hypothesis is that the various components of SG noise follow an α -stable distribution: $w \sim p_w(\hat{w})$, where $\int_{-\infty}^{+\infty} \exp(j2\pi\hat{w}t) p_w(\hat{w}) d\hat{w} = \exp(-|ct|^\alpha)$, with $\alpha \in (0, 2]$, where α, c can vary across different parameters. When $\alpha = 2$, $p_w(\hat{w})$ becomes Gaussian, but for $\alpha < 2$, its variance goes to infinity, thus estimating the SG noise covariance is problematic.

Prior works [79], suggest to define a stochastic process governed by an SDE that uses Lévy Noise instead of a Brownian motion. However, this approach comes with its own challenges, such as the approximation of a fractional derivative, and the use of full gradients. Instead, we propose the following approximate method: we consider that the SG noise follows a Gaussian distribution, with parameters set to minimise the l_2 -distance between $p_w(\hat{w}) = (2\pi\tilde{\sigma}^2)^{-\frac{1}{2}} \exp\left(-\frac{\hat{w}^2}{2\tilde{\sigma}^2}\right)$ and $q_w(\hat{w}) = \int_{-\infty}^{+\infty} \exp(-j2\pi\hat{w}t) \exp(-|ct|^\alpha) dt$ (see 6.4.1 for details). We estimate the parameters of the α -stable distribution by extending [51] to space varying settings, and derive the equivalent variances $\tilde{\sigma}^2$ that minimise the l_2 distance. Then, the $\lambda^{(l)}$ are computed as $\frac{1}{2} \max_j (\tilde{\sigma}_j(\hat{\mathbf{x}})^{(l)})^2$.

Ultimately, the composite noise matrix $\mathbf{\Sigma} = \mathbf{\Lambda}$ is a layer-wise isotropic covariance matrix, which inspires the name of our proposed method as *Isotropic* SGD (I-SGD). Once all parameters $\lambda^{(l)}$ have been estimated, the layer-wise learning rate is determined: for the l -th layer, the learning rate is $\eta^{(l)} = \frac{1}{N\lambda^{(l)}}$.

The practical implementation of I-SGD is shown in Algorithm 2.

Algorithm 2 I-SGD: practical sampling

Sample ($\hat{\mathbf{x}}_0$):
 $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}_0$ ▷ Initialize $\hat{\mathbf{x}}_t$
loop
 $\mathbf{g} = \frac{\nabla \tilde{f}(\hat{\mathbf{x}})}{N}$ ▷ Compute SG
 $\mathbf{C}(\hat{\mathbf{x}})^{\frac{1}{2}(l)} \leftarrow (\lambda^{(l)} - \frac{1}{2}(\tilde{\boldsymbol{\sigma}}(\hat{\mathbf{x}})^{(l)})^2)^{\frac{1}{2}}$
 $\mathbf{n} \sim N(\mathbf{0}, \mathbf{I})$
 $\mathbf{w} \leftarrow \mathbf{C}(\hat{\mathbf{x}})^{\frac{1}{2}(l)} \mathbf{n}$
 $\delta \hat{\mathbf{x}}^{(l)} \leftarrow (N\lambda^{(l)})^{-1} (\mathbf{g}^{(l)} + \sqrt{2}\mathbf{w})$
 $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} - \delta \hat{\mathbf{x}}$ ▷ Update $\hat{\mathbf{x}}$

The computational cost of I-SGD is as follows. Similarly to [16], we define the cost of computing a gradient mini-batch as $C_g(N_b, d)$, where, as in the previous Chapter, d is the dimensionality of the parameter vector. Then (see 6.4.1), the computational cost for estimating the noise covariance scales as $\mathcal{O}(d)$ logarithm computations. The computational cost of generating random samples with the desired covariance scales as $\mathcal{O}(d)$ square roots and $\mathcal{O}(d)$ multiplications. The overall cost of our method is the sum of the above terms. Note that the cost of estimating the noise covariance does not depend on the mini-batch size. The space complexity of I-SGD is the same as SGHMC and SGFS and variants: it scales as $\mathcal{O}(N_{\text{MC}}d)$, where N_{MC} is the number of posterior samples.

6.4.1 Estimation of $\lambda^{(l)}$

As briefly introduced in the previous section, we modify the standard assumption of Gaussian noise and generalize it to the case of heavy tailed noise. We hereafter provide the details needed to build estimators of the variances for the two considered cases (the standard one and the proposed one).

The case of Gaussian noise.

We here give additional details on the estimation of $\lambda^{(l)}$. The simple and naive estimation described in the thesis is the following: $\lambda^{(l)} = \max_{j \in I_l} (\mathbf{g}_j(\hat{\mathbf{x}})^{(l)})^2$. For the Gaussian SG noise case we found however the following (safe) looser estimation of the maximum noise covariance to be more stable: $\lambda^{(l)} = \sum_{j \in I_l} b_j(\hat{\mathbf{x}}) = \frac{\|\mathbf{g}(\hat{\mathbf{x}})^{(l)}\|^2}{2}$.

From a practical point of view, we found the following filtering procedure to be useful and robust:

$$\lambda^{(l)} \leftarrow \mu \lambda^{(l)} + (1 - \mu) \frac{\|\mathbf{g}^{(l)}(\hat{\mathbf{x}})\|^2}{2} \tag{6.2}$$

where an exponential moving average is performed with estimation momentum determined by μ . Notice that during sampling, the same smoothing can be applied

to the tracking of $\mathbf{B}(\hat{\mathbf{x}})$. We refer to the variant of I-SGD implemented using this estimator as **i-sgd-G**. We also considered the case of having a unique, and not layerwise, learning rate, that we indicate by juxtaposing the **(SLR)** acronym to the right of the methods. In this case, the unique equivalent λ is computed as $\sum_l \lambda^{(l)}$.

The case of Heavy Tailed Noise.

As anticipated, we challenge the assumption that SG noise is Gaussianly distributed ([81, 80]) and consider instead heavy tailed distributions. In particular, the hypothesis is that the noise follows an α -stable distribution, i.e.

$$w \sim p_w(\hat{w}) = \mathcal{F}^{-1}(\exp(-|ct|^\alpha)) \quad (6.3)$$

where $\alpha \in [0, 2]$. Notice that except for particular cases, $p_w(\hat{w})$ cannot be expressed in closed form. In general, when $\alpha < 2$ the variance of the distribution goes to infinity and thus dealing with all methods that require the estimation or the usage of a covariance is tricky. It is interesting to underline again that for $\alpha = 2$ the distribution is the usual Gaussian one.

Having acknowledged that the noise is not Gaussian for deep models (at least) two possibilities can be considered: the first one is to study the SDE with Lèvy Noise instead of Brownian, using a formalism similar to the one considered in [79], where *fractional* FPE have been considered. Several practical difficulties are however tied to this choice, such as the necessity to numerically approximate the fractional derivative of order α or the necessity to have full batch evaluations.

The second possibility, that we name **i-sgd- α** , is to neglect the fact that the noise is non-Gaussian, treat this as an approximation error, and use for the theoretical calculations the Gaussian distribution that is closest to the real noise distribution. In particular, for the one dimensional case, we minimise the l_2 -distance between $p_x(\hat{x})$ and $q_x(\hat{x})$, where $p_x(\hat{x}) = \sqrt{2\pi\sigma^2} \exp\left(-\frac{\hat{x}^2}{2\sigma^2}\right)$ and $q_x(\hat{x}) = \mathcal{F}^{-1}(\exp(-|ct|^\alpha))$. As stated above, in general no closed form exists for $q_x(\hat{x})$. Thanks to Parseval's equality, however, we can compute the distance in the frequency domain between the two distributions, i.e.

$$C = \int_{-\infty}^{+\infty} |p_x(\hat{x}) - q_x(\hat{x})|^2 d\hat{x} = \int_{-\infty}^{+\infty} |p(t) - q(t)|^2 dt \quad (6.4)$$

where $p(t) = \exp(-\frac{\sigma^2 t^2}{2})$ and $q(t) = \exp(-|ct|^\alpha)$. Since we are optimizing w.r.t. σ ,

we can write the equivalent cost function

$$\begin{aligned}
 C_{eq} &= \int_{-\infty}^{+\infty} |p(t)|^2 dt - 2 \int_{-\infty}^{+\infty} p(t)q(t)dt = \int_{-\infty}^{+\infty} \exp(-\sigma^2 t^2) dt - 2 \int_{-\infty}^{+\infty} \exp(-\frac{\sigma^2 t^2}{2}) \exp(-|ct|^\alpha) dt \\
 &= \frac{\sqrt{\pi}}{\sigma} - \frac{2}{\sigma} \int_{-\infty}^{+\infty} \exp(-\frac{\tau^2}{2}) \exp(-|\frac{c}{\sigma}\tau|^\alpha) d\tau = \frac{\sqrt{\pi}}{\sigma} - \frac{2\sqrt{2\pi}}{\sigma} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp(-\frac{\tau^2}{2}) \exp(-|\frac{c}{\sigma}\tau|^\alpha) d\tau = \\
 &= \frac{1}{\sigma} \left(\sqrt{\pi} - \sqrt{2\pi} E_{T \sim N(0,1)}[\exp(-|\frac{c}{\sigma}T|^\alpha)] \right). \tag{6.5}
 \end{aligned}$$

Equivalently, we can maximize for $r = \frac{c}{\sigma}$, the following function

$$r \left(\sqrt{\pi} - \sqrt{2\pi} E_{T \sim N(0,1)}[\exp(-|rT|^\alpha)] \right).$$

The expected value does not have a closed form solution, but since the integral is single dimensional, it is possible to integrate numerically and derive the optimal r for a given tail index, i.e. $\hat{r} = \arg \min C(r, \alpha)$ and consequently the optimal σ as $\hat{\sigma} = \frac{c}{\hat{r}}$. Notice that even for moderately small values of α (i.e. $\alpha > 0.5$), the optimal value is roughly $\frac{1}{\sqrt{2}}$, implying that a matching of the scales is sufficient: $\hat{\sigma}^2 = 2c^2$. The parameters α, c are estimated (extending the results of [81, 51] to space varying settings) as described below. Given a sequence of $N = N_1 \times N_2$ samples $w[n]$ from an alpha-stable distribution, it is possible to estimate α, c using

$$\frac{1}{\hat{\alpha}} = \frac{1}{\log(N_1)} \left(\frac{1}{N_2} \sum_{i=0}^{N_2-1} \log \left| \sum_{j=0}^{N_1-1} w[iN_1 + j] \right| - \frac{1}{N} \sum_{i=0}^{N-1} \log |w[i]| \right) \tag{6.6}$$

$$\hat{c} = \exp\left(\frac{1}{N} \sum_{i=0}^{N-1} \log |w[i]| - \left(\frac{1}{\hat{\alpha}} - 1\right) \gamma\right) \tag{6.7}$$

where $\gamma = 0.5772156649015329\dots$ is the Euler-Macheroni constant. Notice that the computational cost for estimation of the two quantities is dominated by the calculation of logarithms, in fact for a full sequence of N independent samples the cost is for the estimation of α $N + N_2$ absolute values, $N + N_2$ logarithms, $2N$ sums, with a per sample cost roughly equal to the cost of 1 logarithm evaluation, and for the estimation of c the cost is N logarithms, sums and absolute values (and thus similarly the cost is dominated by the log evaluation). When considering vectors of independent d -dimensional samples, the computational cost scales as $\mathcal{O}(d)$ logarithms.

6.4.2 A simple toy example

We consider a simple numerical example whereby it is possible to analytically compute the true posterior distribution. We define a simple 1-D regression problem,

in which we have D trigonometric basis functions: $f(x) = \mathbf{w}^\top \cos(\boldsymbol{\omega}x - \pi/4)$, where $\mathbf{w} \in \mathbb{R}^{D \times 1}$ contains the weights of D features and $\boldsymbol{\omega} \in \mathbb{R}^{D \times 1}$ is a vector of fixed frequencies. We consider a Gaussian likelihood with variance 0.1 and prior $p_{\mathbf{w}}(\hat{\mathbf{w}}) = \mathcal{N}(0, I_D)$; the true posterior over \mathbf{w} is known to be Gaussian and it can be calculated analytically.

To assess the quality of the samples from the posterior obtained by I-SGD, in Fig. 6.2 we show the predictive posterior distribution (estimated using Eq. (4.20)) of I-SGD, in comparison to the “ground truth” posterior. Visual inspection indicates that there is a good agreement between predictive posterior distributions, especially in terms of uncertainty quantification for test points far from the input training distribution.

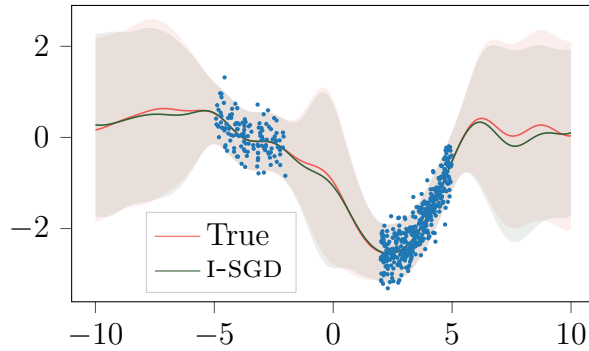


Figure 6.2: True and I-SGD predictive posterior distributions on a simple example.

6.5 Assumptions and convergence to the true posterior.

Our theory shows that the ideal version of I-SGD (1 holds, and $\mathbf{B}(\hat{\mathbf{x}})$ is known) converges to the true posterior with a constant learning rate. This is not the case for existing work. Even when $\mathbf{B}(\hat{\mathbf{x}})$ is assumed to be known, SGFS requires the correction term $\nabla^\top \mathbf{B}(\hat{\mathbf{x}})^{-1} = 0$. Also, both SGRLD and SGRHMC require computing $\nabla^\top \mathbf{B}(\hat{\mathbf{x}})^{-1}$, for which an estimation procedure is elusive. The method in [85] needs a *constant*, diagonal $\mathbf{B}(\hat{\mathbf{x}})$, a condition that does not necessarily hold for deep models. Among all the considered variants, I-SGD is the one that can claim convergence to the true posterior in the broader range of conditions with a constant learning rate determined according to the theory. Since $\mathbf{B}(\hat{\mathbf{x}})$ is estimated, the practical I-SGD can only approximate the true posterior, but having a consistent estimator the error can be made arbitrarily small by increasing the amount of data used for the estimation. All other methods requires either restrictive assumptions about the loss landscape or learning rate annealing to guarantee convergence to the

true posterior. Excluding the first case for preserving generality of the discussion, there is not a clear way of determining the annealing procedure and the tradeoffs are often chosen based on the designer experience more than on sound theoretical basis. Often (if not always), in practice, for all the considered methods practitioners stop the annealing procedure and fix the learning rate. This allows to gather samples in a finite amount of time but introduces a bias in the approximation of the posterior that is not reducible by increasing the number of collected samples. As will be clear from the content of Section 6.6.1, it is not possible to validate the goodness of the various methods and choices in absolute terms since the true posterior distribution is not available and only proxy metrics, with limited significance, are available.

6.6 Experimental Results

We first study I-SGD using standard UCI data-sets [20] and a fully connected Multi Layer Perceptron. Then, we focus on classification and use a CNN (Convolutional Neural Network) on MNIST [49]. We compare I-SGD (with the Gaussian approximation to the estimated α -stable distribution) to SGHMC [16], SGLD [95] and to alternative approaches to approximate Bayesian inference, including MCD [28], and SWAG [56], the variational SGD approach (v-SGD [57]). The code has been written in PYTHON.

6.6.1 A disclaimer on performance characterization

Before diving into the numerical results, the author find of paramount importance to stress a detail on the analysis of the experimental campaign. The concept is seldom discussed in the literature, but is important to underline the limitations of the current scientific status on the subject.

Up to this point, the discussion has been focused on the goodness of the various methods for representing the true posterior distribution. Different methods can or cannot claim convergence to the true posterior according to certain assumptions and the nature of the hyperparameters (annealing learning rate, etc...). When it comes down to the experimental validation of the results the following fundamental problem arises: in general we do not have access to the form of the true posterior, the whole point of SG-MCMC sampling methods is exactly that we are not able to compute posteriors in general, and thus there is no explicit way of measuring the goodness of the collected samples. Only in a limited, uninteresting set of cases, such as the Bayesian linear regression, we can analytically compute the posterior and measure consequently the performance of the various methods.

In absence of closed form solutions, the usual methodology adopted in the literature is to adopt SGHMC as the golden standard and the various methods are compared against it. This choice comes however with at least two shortcomings:

the first one is that in every practical situation we are incurring in a logical loop. As long as the learning rate is not strictly infinitesimal SGHMC does not converge to the true posterior. In the literature various theoretical rates of convergence have been presented for these non ideal cases, but usually such rates contains unknown constants that are either impossible or extremely difficult to be estimated. Any comparison with SGHMC is therefore, at least in line of principle, vacuous.

Even accepting SGHMC as the golden standard, with a leap of faith, it still remains an other important practical problem: having samples from SGHMC and the competitor method that we want to characterize, how to compute the goodness (distance in probability space)? Choosing any valid probability distance metric, such as the KL-divergence, there is not an explicit (and unbiased) way of computing such distance starting from samples. Different techniques are available in literature but they either rely on parametric assumptions about the generating distributions, or are not performing when the cardinality of the problem increases.

The final solution adopted is to compare the different methods in terms of *proxy* metrics evaluated on the test sets, such as the accuracy, the uncertainty quantification performance, the capability of detecting out of distribution samples and many more. It has to be stressed however that being better in terms of these performance metrics does not imply that the sampling method is better at approximating the posterior distribution. The shape of the true posterior distribution is in fact determined by the choice of the likelihood model and the prior for the parameters. If the likelihood is badly chosen (known as model misspecification) or the prior is not representative of our initial knowledge about the problem, then a distribution that is not the true posterior could have better test performance metrics. It is then possible that a method is better than an other in terms of test metrics while being poorer in terms of goodness of the posterior representation.

The two problems should be orthogonal and independent one on the other. In practice, for the reasons above described they are not. Moreover, the considered metrics are useful for any realistic implementation in which we care only about the performance in terms of uncertainty quantification and similar on a test set, albeit they do not provide information about the intrinsic quality of the sampling scheme.

6.6.2 UCI regression tasks, with a multi layer perceptron.

We use a multi layer perceptron (MLP) with two layers and a ReLU activation function; the hidden layer has 50 units. We use the root mean square error (RMSE) for the predictive performance and the mean negative log-likelihood (MNLL) for uncertainty quantification. At test time we use 100 samples to estimate the predictive posterior distribution, using Eq. (4.20). All our experiments use 10-splits. In this set of experiments we use for the class of I-SGD variants an estimation momentum $\mu = 0.5$. We perform a complete analysis of the I-SGD methodology and consider 6 different variants, described in the following.

- **i-sgd-G**. The noise is estimated with the Gaussian hypothesis. We use the *max* version of Assumption 3. The starting point of the sampling method is the same as the beginning of SGHMC sampling.
- **i-sgd- α** . It is equal to the previous variant, but the noise is estimated using the α stable distribution.
- **i-sgd-G sum**. This variant is again equal to the first variant but we use instead the *sum* version of Assumption 3.
- **i-sgd- α sum**. This variant is again equal to the **i-sgd- α** variant while using the *sum* version of Assumption 3.
- **i-sgd-G Ad**. As the **i-sgd-G** variant, but the starting point is a model trained for 20000 iterations using Adam optimizer [43] and learning rate 0.01.
- **i-sgd- α Ad**. As **i-sgd- α** but using as starting point a model trained with Adam optimizer as the previous case.

We compare ourselves against the following methods:

- **sghmc**. We use learning rate equal to 0.01 and *mdecay* 0.01 (check the original paper [85] for more details). Notice that the SGHMC version we are using is not the standard one, but an already improved one with respect to the original [16].
- **sgld**. We use the annealing procedure described in the original paper [95], with initial learning rate 10^{-6} and final learning rate 10^{-8} . The warmup period corresponds to 40000 iterations. After the warmup we keep the learning rate constant, as suggested in both [95] and [1].
- **v-sgd C**. It is the implementation of the variational scheme described in [57]. The starting point of the sampling method is again a model trained for 20000 iterations using Adam and learning rate 0.01.
- **v-sgd**. As **v-sgd C**, with the difference that the variance of the noise is estimated, according to Assumption 2 without removing the mean (i.e. assuming variance equal to second order moment). This experiment is included to perform an ablation study on whether the superior performance of I-SGD with respect to V-SGD is due to the goodness of the sampling method or simply due to the stability introduced by Assumption 2.
- **Baseline**. The single sample result obtained by training the model 20000 iterations using Adam and learning rate 0.01.

Table 6.1: RMSE results for regression on UCI data-sets.

Method	wine	protein	kin8nm	energy	power	boston	concrete
i-sgd-G	0.640 ± 0.04	4.758 ± 0.03	nan ± nan	0.492 ± 0.06	4.521 ± 0.16	3.651 ± 1.17	5.901 ± 0.16
i-sgd-α	0.640 ± 0.04	4.758 ± 0.03	0.078 ± 0.00	0.496 ± 0.06	4.521 ± 0.16	3.651 ± 1.17	5.895 ± 0.17
i-sgd-G sum	0.637 ± 0.04	4.723 ± 0.03	nan ± nan	0.490 ± 0.06	4.410 ± 0.15	3.615 ± 1.12	5.871 ± 0.29
i-sgd-α sum	0.637 ± 0.05	4.723 ± 0.03	0.078 ± 0.00	0.491 ± 0.06	4.410 ± 0.15	3.615 ± 1.12	5.871 ± 0.29
i-sgd-G Ad	0.637 ± 0.04	4.723 ± 0.03	nan ± nan	0.490 ± 0.06	4.410 ± 0.15	3.615 ± 1.12	5.871 ± 0.29
i-sgd-α Ad	0.637 ± 0.04	4.723 ± 0.03	0.078 ± 0.00	0.490 ± 0.05	4.410 ± 0.15	3.615 ± 1.12	5.871 ± 0.29
sghmc	0.629 ± 0.04	4.721 ± 0.03	0.077 ± 0.00	0.487 ± 0.05	4.309 ± 0.14	3.624 ± 1.22	5.791 ± 0.23
sgld	0.733 ± 0.05	5.602 ± 0.08	nan ± nan	2.862 ± 0.32	10.520 ± 2.24	9.454 ± 1.99	14.084 ± 0.94
v-sgd C	0.633 ± 0.04	4.684 ± 0.02	0.078 ± 0.00	nan ± nan	nan ± nan	nan ± nan	5.675 ± 0.54
v-sgd	0.636 ± 0.05	4.723 ± 0.03	0.078 ± 0.00	0.504 ± 0.08	6.085 ± 5.23	nan ± nan	5.669 ± 0.26
Baseline	0.635 ± 0.05	4.736 ± 0.03	0.080 ± 0.00	0.497 ± 0.06	4.353 ± 0.12	3.678 ± 1.21	5.544 ± 0.22

If not stated differently for the single methods, we consider a warmup period of 10000 and we do store a sample every 1000 iterations (the technical name in the SG-MCMC community is *keepevery*, whose value in this case is 1000) for all methods. In this set of experiments we omit results for SWAG and MCD, which we keep for more involved scenarios.

Tabs. 6.1,6.2 present an overview of our results. The two metrics considered are the root mean squared error (RMSE) and the mean negative log likelihood (MNLL) (in both cases the lower, the better). As anticipated, all experiments are performed over 10 random splits. If *any* of the runs over the splits for a particular algorithm did diverged we indicate it with a *nan* (this is useful to assess robustness of the various methods). The sheer observation of the best performing algorithm would indicate that, not differently from the literature expectations, SGHMC is the best performing algorithm. We would like to underline however that in most cases I-SGD variants are very close, especially if considering the standard deviations of the performance and the improvements with respect to the baseline. All I-SGD methods perform essentially on par, outperforming alternatives in some cases. For this set of experiments we had no luck in obtaining good results with SGLD. The two considered variants of V-SGD are generally slightly worse than the I-SGD family. However in most cases they still improve the results with respect to the naive baseline. As a general comment, the difference between the baseline and the different sampling schemes is more evident in terms of MNLL, a metric more linked to uncertainty quantification than the basic RMSE.

We moreover include an analysis of the speed of convergence for **i-sgd- α** , **i-sgd-G** compared with SGHMC for the WINE dataset by continuing the sampling collecting 1000 samples (Figure 6.3). For this first set of experiments it seems that the presence of momentum do help SGHMC converging faster.

6.6.3 Classification tasks, with deeper models.

In this set of experiments we use a LENET-5 (Convolutional Neural Network) on the MNIST dataset [49] to perform classification.

Table 6.2: MNLL results for regression on UCI data-sets

Method	wine	protein	kin8nm	energy	power	boston	concrete
i-sgd-G	1.095 ± 0.12	4.396 ± 0.03	nan ± nan	0.711 ± 0.14	3.095 ± 0.06	3.141 ± 0.77	6.627 ± 0.54
i-sgd- α	1.095 ± 0.12	4.396 ± 0.03	-0.499 ± 0.65	0.737 ± 0.14	3.095 ± 0.06	3.140 ± 0.77	6.621 ± 0.53
i-sgd-G sum	1.085 ± 0.12	4.349 ± 0.03	nan ± nan	0.710 ± 0.13	3.083 ± 0.07	3.227 ± 0.88	6.384 ± 0.55
i-sgd- α sum	1.094 ± 0.13	4.349 ± 0.03	-0.508 ± 0.64	0.811 ± 0.21	3.083 ± 0.07	3.227 ± 0.88	6.385 ± 0.56
i-sgd-G Ad	1.085 ± 0.12	4.349 ± 0.03	nan ± nan	0.710 ± 0.13	3.083 ± 0.07	3.227 ± 0.88	6.384 ± 0.55
i-sgd- α Ad	1.085 ± 0.12	4.349 ± 0.03	-0.511 ± 0.65	0.716 ± 0.13	3.083 ± 0.07	3.227 ± 0.88	6.385 ± 0.56
sghmc	1.044 ± 0.12	4.150 ± 0.02	-0.785 ± 0.38	0.927 ± 0.27	2.932 ± 0.04	3.074 ± 0.84	5.601 ± 0.43
sgld	1.452 ± 0.18	5.496 ± 0.10	nan ± nan	99.089 ± 41.12	7.648 ± 2.24	35.004 ± 16.58	114.546 ± 38.16
v-sgd C	1.122 ± 0.14	4.347 ± 0.03	-0.506 ± 0.64	nan ± nan	nan ± nan	nan ± nan	39.396 ± 23.68
v-sgd	1.130 ± 0.14	4.393 ± 0.04	-0.497 ± 0.65	5.885 ± 2.69	3.685 ± 1.98	nan ± nan	41.911 ± 8.30
Baseline	1.179 ± 0.02	3.967 ± 0.03	0.924 ± 0.00	1.048 ± 0.03	3.071 ± 0.06	5.376 ± 2.79	13.838 ± 1.03

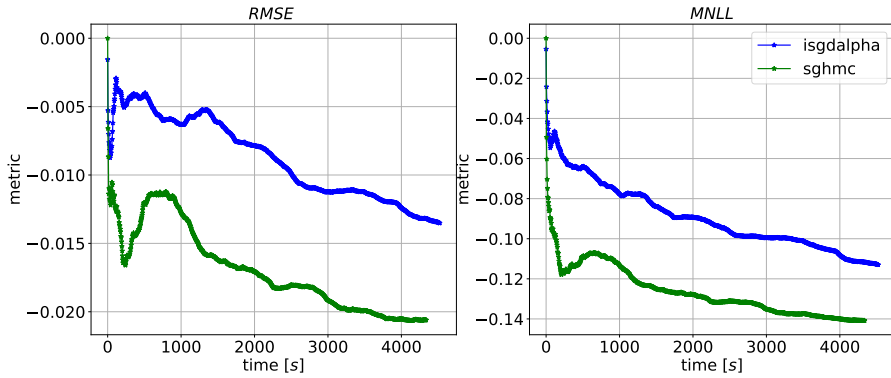


Figure 6.3: Convergence speed of SGHMC and I-SGD for the WINE dataset. To improve readability the metrics are shifted vertically with respect to the value of the SGHMC method at time instant 0.

We compare different methods in terms of

- Accuracy (ACC). The percentage of correctly classified samples (the higher, the better)
- Mean negative log likelihood MNLL (the lower, the better).
- Mean entropy for the test dataset. More details on the concept of entropy are also discussed in Chapter 9 (the lower, the better). The entropy of the output of an N_{cl} classes classifier, represented by the vector \mathbf{p} , is defined as

$$-\sum_{i=1}^{N_{cl}} p_i \log p_i.$$
- Expected calibration error (ECE). It is the average, across all values of $p \in [0,1]$, of the difference between the confidence of the model and its actual precision. Referring to Equation (4.21), the difference is calculated as $P(\hat{y} = y | \hat{p} = p) - p$. Zero is a perfect value. A value greater than zero indicates an overconfident model and viceversa for a value smaller than zero.

- Mean entropy for the NOTMNIST dataset. It is the entropy at the output of the classifier when the input is taken from a different dataset. It is a test of detection capability of out of distribution samples (the higher, the better).

Also in this set of experiments we consider different variants of I-SGD:

- **i-sgd-G**. Noise is estimated under the Gaussian hypothesis. The version of Assumption 3 is the *max* one. The starting point of the sampling scheme is a model pretrained for 20000 iterations using Adam with learning rate 0.01.
- **i-sgd- α** . As the previous version but using alpha stable noise estimation procedure.
- **i-sgd-G sum**. As the first version but using the *sum* version of Assumption 3.
- **i-sgd- α sum**. As the second version but using the *sum* version of Assumption 3.

We compare our method against the following competitors:

- **sghmc**. The learning rate is 0.01 and the *mdecay* is 0.01. The warmup period is 10000 iterations.
- **sgld**. The initial learning rate is set to 10^{-5} while the final one is 10^{-3} . The warmup period is 40000 iterations.
- **v-sgd**. and **v-sgd C**. The two variants of the sampling scheme are initialized with a model pretrained using the same procedure as the I-SGD family.
- **swag**. The initial pretraining is again the same as the I-SGD cases. The measurements are collected every epoch for 100 epochs.
- **mcd**. The pretraining is done for 20000 iterations using SGD with learning rate 0.001 and momentum 0.5. The model is modified to include learnable dropout rates for all the parameters. Differently from the other schemes, at test time we use 1000 samples.

We stress the fact that neither SWAG or MCD are SG-MCMC methods. For all the SG-MCMC methods, including v-SGD and variant, we collect 100 samples every 10000 iterations. The considered batch size for all methods is 128. The estimation momentum for I-SGD is again 0.9 for all variants.

Results, obtained by averaging over three random seeds, are presented in Table 6.3. For this set of experiments, v-SGD C is the best performing algorithm for most of the metrics, besides from the out of distribution entropy where it fails. The I-SGD

Table 6.3: Results for classification on MNIST data-set.

Method	ACC	MNLL	mean H_0	ECE	mean H_1
i-sgd-α	9926.3333 \pm 1.8856	244.7462 \pm 2.3174	0.0413 \pm 0.0007	0.0512 \pm 0.0002	0.8268 \pm 0.0420
i-sgd-G	9924.6667 \pm 2.6247	239.2841 \pm 1.8637	0.0414 \pm 0.0007	0.0509 \pm 0.0004	0.7937 \pm 0.0268
i-sgd-α sum	9922.0000 \pm 1.4142	245.4821 \pm 2.2720	0.0415 \pm 0.0004	0.0506 \pm 0.0001	0.8237 \pm 0.0085
i-sgd-G sum	9924.3333 \pm 0.9428	244.1239 \pm 1.7034	0.0415 \pm 0.0006	0.0508 \pm 0.0002	0.8367 \pm 0.0448
sghmc	<i>9932.3333 \pm 3.6818</i>	259.7695 \pm 11.2759	0.0584 \pm 0.0016	0.0532 \pm 0.0005	1.1733 \pm 0.0596
sgld	9931.0000 \pm 2.9439	240.1740 \pm 5.5880	0.0485 \pm 0.0008	0.0523 \pm 0.0003	<i>1.1940 \pm 0.0664</i>
v-sgd	9922.3333 \pm 5.2493	229.5869 \pm 5.9108	0.0288 \pm 0.0003	0.0486 \pm 0.0002	0.0619 \pm 0.0066
v-sgd C	9922.6667 \pm 4.7842	<i>223.7863 \pm 4.1745</i>	<i>0.0287 \pm 0.0003</i>	<i>0.0485 \pm 0.0001</i>	0.0792 \pm 0.0231
mcd	9923.0000 \pm 4.9666	331.6276 \pm 11.3830	0.0910 \pm 0.0014	0.0543 \pm 0.0003	0.6179 \pm 0.0831
swag	9916.3333 \pm 2.4944	282.0489 \pm 6.1477	0.0524 \pm 0.0018	0.0517 \pm 0.0003	0.4667 \pm 0.0562

variants perform similarly to SGLD and in general they outperform SGHMC. However, in general, the sampling methods perform similarly. It is interesting to notice the difference between the two non SG-MCMC methods (MCD and SWAG) for which the performances are generally poorer than the cluster represented by SG-MCMC methods. We stress that also this set of results confirms the desired behaviour: to be able to obtain competitive results with a possibly simpler algorithm.

Chapter 7

Conclusions of Part I

High performing hardware and growing capabilities of large scale collection of data has pushed the development of complex machine learning models. Concurrent to such expansion has grown the request, both from academic and industrial community, to deploy efficient and robust mechanisms for quantification of uncertainty of the considered models. In this first part of the thesis we explored a topic, Stochastic Gradient Montecarlo Methods SG-MCMC, that find its roots in the intersection of Statistical Physics, Stochastic Optimization and Montecarlo Integration. We provided a clean theoretical basis for all the considered topics in a tutorial fashion. The goal was to provide a solid reference for anyone interested in the topic that is formally correct but still easy to understand. As highlighted in the text, many different references are available. To the best of our knowledge, there does not exist a book or thesis whose goal is to present with a unified notation all these topics.

As mentioned in Chapter 5, SG methods allowed Bayesian posterior sampling algorithms, such as MCMC, to regain relevance in an age when data-sets have reached extremely large sizes, drastically reducing the computational costs. However, despite mathematical elegance and promising results, standard approaches from the literature are restricted to simple models. As explained, in fact, the noise induced by the random subsampling destroys, in general, the convergence guarantees of the methods.

The sampling properties of these algorithms are determined by simplifying assumptions on the loss landscape, which do not hold in general.

By leveraging the presented theoretical basis, the sampling properties of these algorithms are determined. To obtain useful results simplifying assumptions on the loss landscape, which do not hold for very large deep models, must be taken. SG-MCMC algorithms require vanishing learning rates, which force practitioners to develop creative annealing schedules that are often model specific and difficult to justify. The current state is thus that besides from idealistic scenarios or when considering extremely large sampling times (to collect samples with vanishing learning

rates) there is not the possibility of guaranteeing convergence to the desired posterior. In fact SG-MCMC algorithms produce samples from the desired posterior only in the limit of learning rates going to zero. This impractical requirement force practitioners to create learning rate schedules that are model specific and, more importantly, difficult to justify (the optimal parameters are often found by brute force grid search).

We have attempted to target these weaknesses by suggesting a simpler algorithm that relies on fewer parameters and mild assumptions compared to the literature. We introduced a unified mathematical notation to deepen our understanding of the role of the SG noise and learning rate on the behaviour of SG-MCMC algorithms. We presented a practical variant of the SGD algorithm, which uses a constant learning rate, and an additional noise term to perform Bayesian posterior sampling. Our proposal is derived from an ideal method, which guarantees that samples are generated from the true posterior. In the presence of an Oracle, in fact, it is possible to inject noise with a covariance such that, when combined with the subsampling induced noise, the resulting covariance is an identity matrix. The condition is referred to as noise covariance isotropy, we call the method Isotropic Stochastic Gradient Descent (I-SGD). When the noise terms are empirically estimated, the theoretical properties cannot be guaranteed anymore. In accordance with the recent literature we consider also the case in which the SG noise is not assumed to be Gaussian, but, more generally, α -stable. Our method determines the learning rate, and it offers a very good approximation to the posterior, as demonstrated by our experimental campaign in which we show that the proposed algorithm is competitive with the others, extensively analysing different variants of the proposed method. Future research will be focused on a deeper understanding of the interplay between statistical properties of the SG noise and SDE simulation goodness.

Part II

Deep Information Networks

Chapter 8

Introduction to Part II

The first part of the thesis has focused on sampling methods to be applied on parametric models with the implicit assumption of considering scenarios where the algorithms have continuous parameters. Continuous parameters algorithms have been specifically designed for continuous input datasets. While it could be possible to apply discrete parameters algorithms to inputs such as images, here assumed to belong to some high dimensional continuous space, it is natural to understand why continuous parameters algorithms are preferred.

However, in the real world, many datasets are discrete (categorical) in nature. The simplest example is the case of natural language, where the dataset is composed by the set of words. Another example could be a dataset that provides information about whether a given device is susceptible to malicious attacks or not. Some of the features, such as the type of operating system (*OS*), will be non ordinal and categorical in nature.

The distinction between the two types of data, continuous or categorical, has even deeper roots in the field of machine learning and artificial intelligence. It traces back to the old dichotomy [83] that has emerged between the symbolic approaches to artificial intelligence, and the computational connectionist approaches. To determine whether the first or second approach is the correct one is outside the scope of this dissertation, and we simply accept that in real world it could be useful as well to be proficient at designing and tuning algorithms that work well with categorical datasets.

In this second part of the thesis we propose a novel machine learning algorithm specifically designed for classification on categorical datasets. We name this algorithm Deep Information Networks (DIN), [25, 26]. The scheme we propose is based on information theoretic concepts, specifically the information bottleneck criterion. Differently from the first part of the thesis we do not aim at presenting the family of classification algorithms for categorical variables in its entirety, whereas we focus mainly on the proposed algorithm and on its closest relative (decision trees). We put an accent in the discussion on possible dedicated hardware implementations of

the considered method. Again we investigated a new algorithm having the target of practicality in mind: in this case the practicality metric we consider is the hardware implementation easiness. Despite being the discussion on implementations at an extremely high level we stress important concepts like modularity, locality of the computations and parallelizability of the pipelines. In summary, the algorithm we propose, is specifically designed to perform classification on categorical datasets and it is easily deployable in hardware.

8.1 Overview

This second part of the thesis will follow a natural path in the presentation of the topics: it will be easy to directly understand why the theoretical foundations, in this case Information Theory, are needed for the development of the implementation part of the discussion.

This second part is composed as follows:

- Chapter 9 is focused on the introduction of the Information Theory concepts that are needed for the understanding of the proposed method (including the Information Bottleneck and Statistical Sufficiency) and gives a general introduction to decision trees.
- Chapter 10 presents the first version of the proposed DIN algorithm, based on the Information Bottleneck principle.
- Chapter 11 treats an extension and modification of the proposed algorithm, that we name Probabilistic DIN and extend the DIN framework to ensemble methods.
- Chapter 12 draws conclusions about the proposed method.

We hereafter expand briefly the highlight of the thesis above mentioned. Chapter 9 builds the theoretical foundations for the rest of the thesis. In particular, in Section 9.1 the main concepts of Information Theory are presented. The concept of Entropy and conditional entropy is introduced, explaining them under the perspective of average ignorance that an omniscient daemon (Maxwell's daemon) has to fill for us. Built on top of this theoretical foundation we find in Section 9.2 the concept of Information Bottleneck, the most important element of this second part of the thesis. A careful comparison with the concept of (minimal) sufficient statistic is then proposed. Then in Section 9.3 we concentrate on the the problem of designing and studying classification algorithms when dealing with categorical datasets, as mentioned in the Introduction (Chapter 8). It would be a mistake to interpret our proposed algorithm, the DIN, as a particular instance of a decision tree, since the differences, both structural and conceptual, are not negligible. To

fully understand such differences, however, it is evident that it is necessary to have at least the basic knowledge of what decision trees are and their basic working mechanisms (respectively Section 9.3.1, Section 9.3.2). Finally, since the DIN algorithm will be extended to the ensemble case, the concept of *random forests*, is presented in Section 9.3.3.

We describe in Chapter 10 the first version of the proposed DIN algorithm, built using the Information Bottleneck principle. The overall architecture and its building components are carefully described in Section 10.2, where the concept of Information node and Mixers are presented. In this section in particular it can be understood the structural difference between the DIN and a decision tree. By making use of well known information theoretic inequalities, section 10.3 gives some insight on the theoretical properties of the proposed algorithm and gives an hint about its working principle (the concept of *sifting* information). A first set of experimental results is presented in Section 10.4.

As mentioned before, the DIN concept is extended by considering ensemble methods with the so called Probabilistic DIN. Chapter 11 is exactly focused on this topic. The different assumption about statistical independence of the features is discussed in Section 11.1.3, where a new version of the combiner is described. Many different, independent networks are trained and combined, as described in Section 11.2.1. Sections 11.1 and 11.2 investigate the architecture and the working mechanism of the new Probabilistic DIN (with a special focus on differences w.r.t. the previous version, Chapter 11). In Section 11.3 the theoretical properties of the method are discussed while a large experimental campaign is reported in Section 11.4.

8.2 Notations

Similarly to the first part of the thesis, we here briefly introduce the necessary notations. Some differences with respect to the first part are necessary to be consistent with the related literature. As for the first part, when dealing with system of equations we use matricial notation. We indicate scalar with lowercase symbols, e.g. a , vectors (always column) with bold lowercase symbols, \mathbf{a} . When considering the single j_{th} element of the vector \mathbf{a} , we use the following notation $(\mathbf{a})_j = a_j$. The transposition operation is indicated with the symbol \top . Matrices are indicated with bold, uppercase letters, \mathbf{A} . Differently from the first part, we slightly modify the notation to help the reader and indicate scalar random variables with uppercase (non bold) symbols, e.g. A . This will be hopefully helpful in distinguishing between the random variables and their realization. This notation is consistent with many standard Information Theory textbooks [23].

In this second part of the thesis we deal only with categorical random variables. A discrete (categorical) random variable X is an element of the finite set \mathcal{X} with

cardinality $|\mathcal{X}|$. We indicate again the *probability distribution* using the symbol p . The probability distribution must be valid. This requirement corresponds to having a function $p : \mathcal{X} \rightarrow [0,1]$ that jointly satisfies $p(X = x) = p_X(x) \geq 0 \quad \forall x \in \mathcal{X}$ and $\sum_{x \in \mathcal{X}} p_X(x) = 1$. When useful in solving ambiguities we add a pedix corresponding to the considered random variable(s) to the probability distribution symbol p , i.e. $p_{X,Y}(x,y)$ corresponds to the joint distribution of two random variables X, Y whereas $p_{X|Y}(x|y)$ to the conditional. Simply indicating $p(x,y)$ could introduce ambiguities. Notice, that differently from the first part of the thesis, and consistently with many Information Theory texts, the letter X will usually indicate the input datapoints and not the parameters of the considered algorithms.

The indicator function $\mathbb{1}(\cdot)$, already defined, is easily extendible to discrete (categorical) domains. The behaviour of the function is the same as for the one defined with continuous domains: it takes value 1 if the considered event is verified and 0 otherwise.

The Dirac delta finds its counterpart in the discrete domain with the Kronecker delta. It is usually considered a function of two integers p, q , with codomain $\{0,1\}$. Classical notations include: $\delta[p - q], \delta[p, q], \delta_{p,q}$. In this thesis we use the latter. Specifically we define

$$\delta_{p,q} = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$$

The last useful quantity of interest is the Kronecker product between two matrices \mathbf{A}, \mathbf{B} , indicated with the symbol \otimes and defined as:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}$$

The same can be written in explicit form as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ & & & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ & & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ & & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ & & & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}$$

Chapter 9

Information Theory and Decision Trees

In this chapter we provide a generic introduction to Information Theory and Decision Trees. The first Section in this introduction, 9.1, presents briefly the main concepts of Information Theory. Almost single handedly founded by Claude Shannon, with a seminal paper in 1948 (a recent reprint of the original paper is available as [78]), it is the theory that deals with the amount of *information* that multiple random variables contains one about the other. As to what *exactly* is information, the answer is extremely difficult from a practical, and even philosophical, point of view. Being outside the aim of this thesis to give a through introduction to Information Theory, we will give only the highlights under a lens that is particularly useful for the understanding of the new classification algorithm proposed in this thesis.

In Section 9.2 we present the Information Bottleneck principle [92, 91]. It is possible the single most important concept for this thesis. It is the framework in which random variables are stochastically transformed according to a well defined principle of compression and information preservation. The reader more acquainted with the Signal Processing literature, will find interesting the presentation of the link of the IB principle with the concept of Minimal Sufficient Statistic. The connection is presented in Section 9.2.1 where we give a proof slightly different than the one presented in [77]. The technical proof concerning the IB framework is presented in Section 9.2.2.

In Section 9.3 we explain the problem of building classifiers when the underlying dataset is composed by discrete (categorical) random variables. The general, black box methods that have been described in the first part of the thesis, are in fact better suited for dataset composed by continuous attributes, and an overview of the possibilities that we have when dealing with categorical random variables is presented. Among the many possibilities, two different schools of thought are the basis for the biggest branches: transform the categorical variables into continuous

ones according to some procedure, and use the methods that are known to work with continuous variables, or devise new algorithms that are specifically tailored to categorical datasets. In this second part of the thesis we consider the latter option.

A synthetic presentation of decision trees is then provided in Section 9.3.1 where the basic topological features of trees are discussed. Of particular interest is then Section 9.3.2 in which the general form algorithm for learning (a.k.a. growing) a decision tree is presented, and the basic information theoretic concepts introduced in Section 9.1 are used in the construction of the tree. Finally, in Section 9.3.3 the concept of *random forests*, the union of many different (randomly generated) decision trees is presented.

9.1 Information Theory

A discrete (categorical) random variable X is an element of the finite set \mathcal{X} with cardinality $|\mathcal{X}|$. It has probability distribution fully determined by a function $p : \mathcal{X} \rightarrow [0,1]$, where $p(X = x) = p_X(x) \geq 0 \quad \forall x \in \mathcal{X}$ and $\sum_{x \in \mathcal{X}} p(x) = 1$.

The entropy of a random variable X , measured in *bits*, is defined as

$$\mathbb{H}(X) = - \sum_{x \in \mathcal{X}} p_X(x) \log_2 p_X(x). \quad (9.1)$$

Entire books cover in detail how to interpret the concept of entropy, a notable example being [23], but being a full exposition outside the scope of this thesis, we simply state that entropy is a measure of the uncertainty linked to a given random variable. In particular it is possible to show [23], that the concept of entropy can be understood through the following mental experiment:

Suppose that a random variable X is extracted according to a given probability distribution $p_X(x)$. The entropy of the random variable is the average (over the distribution $p_X(x)$) number of questions with a binary (yes/no) answer that we have to ask an omniscient demon, a.k.a. Maxwell's daemon, to be able to determine what is the extracted random variable. It is assumed that we do have knowledge about the distribution $p_X(x)$ and that we are able to construct, based on this knowledge, an optimal policy for question posing.

The statement is not completely correct. To be true we must have that for all elements x of the set \mathcal{X} , the probability of x can be expressed exactly as a negative power of 2, i.e. $-\log_2(p_X(x)) = n$, $n \in \mathbb{N}$. If this is not the case however, it can be shown [23], that if we group together $N \rightarrow \infty$ symbols, the average number of questions we have to ask the omniscient daemon converges to $N\mathbb{H}(X)$, and thus the average *per symbol* number of questions is again $\mathbb{H}(X)$. The entropy is always non negative, i.e. $\mathbb{H}(X) \geq 0$, for any valid probability distribution $p_X(x)$. The

proof is simple from (9.1) by noticing that being $p_X(x) \leq 1$ for any distribution, $\log_2(p_X(x)) \leq 0$. In according to the relationship between entropy and number of questions it makes sense that a negative number of questions is an unphysical concept. We stress that this line of reasoning and all the following properties we will list are only valid for discrete random variables, since for continuous random variables completely different bounds and lines of reasoning are necessary.

For a couple of random variables $X \in \mathcal{X}, Y \in \mathcal{Y}$, with probability defined by $p(X = x, Y = y) = p_{X,Y}(x, y)$, the joint entropy is defined as

$$\mathbb{H}(X, Y) = - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p_{X,Y}(x, y) \log_2 p_{X,Y}(x, y). \quad (9.2)$$

An easy way to interpret the joint entropy is to simply promote the couple of random variables X, Y to a vector $Z = (X, Y) \in \mathcal{X} \times \mathcal{Y}$ and consider the entropy of the vector Z .

$$\mathbb{H}(X, Y) = \mathbb{H}(Z) = - \sum_{z \in \mathcal{X} \times \mathcal{Y}} p_Z(z) \log_2 p_Z(z). \quad (9.3)$$

Similarly the conditional entropy of X given Y is

$$\mathbb{H}(X|Y) = - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p_{X,Y}(x, y) \log_2 p_{X|Y}(x, y), \quad (9.4)$$

and can be understood as the uncertainty about the random variable X if we are able to observe Y , the average number of questions we have to ask to guess X given that we observed Y . The same holds for the inverse direction,

$$\mathbb{H}(Y|X) = - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p_{X,Y}(x, y) \log_2 p_{Y|X}(x, y). \quad (9.5)$$

Fundamental to the understanding of the rest of the thesis is the concept of mutual information between the two variables X and Y . Suppose that we are only observing the variable Y but not the variable X . Mutual information is defined as the average number of questions that we are saving to determine the variable X by asking the daemon with respect to the case in which Y is not observed. Formally

$$\mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y). \quad (9.6)$$

It is easy to prove that an equivalent definition is $\mathbb{I}(X; Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$. We want to stop for a moment and explain why the concept of mutual information is of paramount importance when dealing with classification problems on categorical datasets. We propose an example tailored to the classification settings, although with little effort the same example can be adapted to a classical telecommunication coding scenario. Consider the case in which a random variable of interest, X generate a second random variable Y , in graphic language

$$X \rightarrow Y, \quad (9.7)$$

and while our interest lies in the variable X we are only able to observe Y . It is obvious that we do not have access to a Maxwell daemon and thus all information that is lost is not recoverable. However, and this is particularly clear by rewriting

$$\mathbb{H}(X) = \mathbb{I}(X; Y) + \mathbb{H}(X|Y), \quad (9.8)$$

out of the $\mathbb{H}(X)$ questions we would have to ask to the daemon, $\mathbb{I}(X; Y)$ are saved by observing Y , while $\mathbb{H}(X|Y)$ will remain unanswered and their information content forever lost and contribute to the uncertainty that we have about X for the classification problem. It is then trivial to understand why the higher the mutual information, the better.

The mutual information can be expressed as the KL -divergence between the joint and the marginal probabilities

$$\mathbb{I}(X; Y) = \mathbb{KL}(p_{X,Y}(x, y) || p_X(x)p_Y(y)) = \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{X,Y}(x, y) \log_2 \left(\frac{p_{X,Y}(x, y)}{p_X(x)p_Y(y)} \right). \quad (9.9)$$

Usually the KL -divergence is expressed using the natural logarithm base, but the conversion between the two definitions is simply achieved by a multiplicative constant. From (9.9) we can observe important properties, the first one being that mutual information, being expressible as a KL -divergence, is always positive. Since we are interpreting mutual information as the average number of questions we are saving, it is reasonable to observe that the quantity is positive. Having a negative number of average saved questions would imply that knowledge of Y would *confuse* us more about the variable X . Non negativity of mutual information implies the important property that *knowledge cannot hurt*. Equivalently, we can look at the derived inequality

$$-\mathbb{I}(X; Y) = -\mathbb{H}(X) + \mathbb{H}(X|Y) \leq 0 \rightarrow \mathbb{H}(X|Y) \leq \mathbb{H}(X) \quad (9.10)$$

that once again tells us: knowing Y cannot increase the average number of questions we have to make to determine X with respect to the case in which Y is not observed. It can be trivially proved moreover that

$$\mathbb{I}(X; Y) \leq \min(\mathbb{H}(X), \mathbb{H}(Y)) \quad (9.11)$$

It is useful to introduce the so called Information Processing Inequality and its relationship with Markov chains. Suppose that X, Y, Z form a Markov chain

$$X \rightarrow Y \rightarrow Z, \quad p_{Z|X,Y} = p_{Z|Y} \quad (9.12)$$

The Data Processing Inequality states that the mutual information between the two ends of a Markov chain can only decrease as the length of the chain is increased

$$\mathbb{I}(X, Z) = \mathbb{H}(X) - \mathbb{H}(X|Z) \leq \mathbb{H}(X) - \mathbb{H}(X|Y, Z) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{I}(X, Y) \quad (9.13)$$

where the first inequality is due to the fact that the entropy of X knowing Z, Y is smaller or equal to the entropy of X when just Z is known, and the second equality ($\mathbb{H}(X|Y, Z) = \mathbb{H}(X|Y)$) is easily shown thanks to the Markovian nature of the chain. For the interested reader, it is possible to draw an analogy between this inequality and the thermodynamic principle that states that for every physical system, every time something is done, the total entropy can only increase (at best remain the same).

9.2 Information Bottleneck (IB)

In this thesis, we make heavy use of the so-called “information bottleneck” principle [92, 91]. The formal setting is the following: suppose that a random variable (a source) X_{in} , is probabilistically linked to another target random variable Y according to some probability $p_{Y|X_{in}}$. Our goal is, given the observation of the random variable X_{in} , to produce another random variable X_{out} according to some conditional probability $p(X_{out} = j|X_{in} = i)$. The conditional probability is chosen such that minimises the mutual information $\mathbb{I}(X_{in}; X_{out})$ between its input X_{in} and output X_{out} for a given mutual information between its output and the target $\mathbb{I}(X_{out}; Y)$. In other terms, compresses as much as possible and throws away irrelevant information, while saving the information about the desired random variable Y .

The principle can be cast in the form of an optimization problem, as the minimisation of the following Lagrangian

$$\mathcal{L} \{p(X_{out} = j|X_{in} = i)\} = \mathbb{I}(X_{in}; X_{out}) - \beta \mathbb{I}(Y; X_{out}), \quad (9.14)$$

where β is a Lagrangian coefficient that implicitly determines the target mutual information between Y and X_{out} . The idea is that in the following chain

$$Y \rightarrow X_{in} \rightarrow X_{out} \quad (9.15)$$

we have from the data processing inequality that $\mathbb{I}(X_{out}; Y) \leq \mathbb{I}(X_{in}; Y)$. Transforming X_{in} then reduces the amount of information that we have about Y . The concept is however to build the conditional probability $p(X_{out} = j|X_{in} = i)$ in such a way that the information about Y is preserved (totally or not) while the irrelevant information contained in X_{in} , that is useless from a classification point of view and increase the probability of overfitting, is removed.

It can be proved ([92], also Section 9.2.2) that the probabilistic mapping between X_{in} and X_{out} that maximizes the Lagrangian is given by

$$p(X_{out} = j|X_{in} = i) = \frac{1}{Z(i; \beta)} p(X_{out} = j) e^{-\beta d(i, j)}, \quad i = 0, \dots, N_{in}-1, j = 0, \dots, N_{out}-1 \quad (9.16)$$

where $d(i, j)$ is the Kullback-Leibler divergence

$$\begin{aligned} d(i, j) &= \sum_{m=0}^{N_{class}-1} p(Y = m|X_{in} = i) \log_2 \frac{p(Y = m|X_{in} = i)}{p(Y = m|X_{out} = j)} \\ &= \mathbb{KL}(p(Y|X_{in} = i) || p(Y|X_{out} = j)) \end{aligned} \quad (9.17)$$

and $Z(i; \beta)$ is a normalizing coefficient that allows to get

$$\sum_{j=1}^{N_{out}-1} p(X_{out} = j|X_{in} = i) = 1 \quad (9.18)$$

The probabilities $p(X_{out} = j|X_{in} = i)$ can be iteratively found using the Blahut-Arimoto algorithm [92].

9.2.1 Sufficient Statistic

This section is intended to be useful as guide for the understanding of the main idea behind the algorithm we will describe in Chapters 10,11.

Consider again the case in which the probabilistic link between two random variables X_{in}, Y is

$$Y \rightarrow X_{in}.$$

If we observe X_{in} but are only interested in Y it is not necessary to *store* X_{in} in its entirety. This concept is formalized (usually with a frequentist and not Bayesian approach) by means of **sufficient statistic** [42]. There are many possible, equivalent ways of defining such a concept [42],[23]. We here present the one based on information theory. Since we are not interested in X_{in} in its entirety, it is possible to consider a transformed (possibly stochastic) version of it. Formally, we say that we generate $X_{out} = T(X_{in})$, where the mapping $T(\cdot)$ can be either stochastic or deterministic. In graphical terms we extend the chain as

$$Y \rightarrow X_{in} \rightarrow X_{out} = T(X_{in})$$

The data processing inequality (9.13) immediately tells us that $\mathbb{I}(X_{out}; Y) \leq \mathbb{I}(X_{in}; Y)$.

Definition 2. *The variable X_{out} is said to be sufficient statistic about Y iff $\mathbb{I}(X_{out}; Y) = \mathbb{I}(X_{in}; Y)$*

Equivalently, all the relevant information that X_{in} contains about Y is conserved and the data processing inequality is attained with equality. It seems reasonable, at least in an ideal world, that no matter what we choose for transforming the variable X_{in} we should always aim at conserving all the relevant information about Y and thus build a sufficient statistic. Among all sufficient statistics, of particular interest is the subclass of **minimal** sufficient statistics:

Definition 3. X_{out} is said to be a minimal sufficient statistic about Y **iff** it is a sufficient statistic and for every other sufficient statistic P there exists a deterministic function $f(\cdot)$ such that $X_{out} = f(P)$.

Among all sufficient statistics, the "smallest" ones are of particular interest both for theoretical and practical reasons.

Assuming to have knowledge of the quantity $\mathbb{I}(X_{in}; Y)$, the problem of finding the minimal sufficient statistic can be solved by means of the Information Bottleneck criterion [77]. Remember that the value of β of the Lagrangian implicitly defines a target mutual information $I(Y; X_{out})$. Given the target $I(X_{out}; Y) = I(X_{in}; Y) \leftrightarrow \beta = \bar{\beta}$, finding a minimal sufficient statistic is equivalent to solving the Lagrangian

$$\mathcal{L} = \mathbb{I}(X_{in}; X_{out}) - \bar{\beta} \mathbb{I}(Y; X_{out}), \quad (9.19)$$

or, equivalently

$$\min \mathbb{I}(X_{in}; X_{out}) \quad \text{s.t.} \mathbb{I}(Y; X_{out}) = \mathbb{I}(Y; X_{in}), \quad (9.20)$$

We hereafter present a proof (slightly different from the one in [77]) to show that (9.20) provides us with a minimal sufficient statistic. We start by rewriting (9.20) as

$$\min \mathbb{I}(X_{in}; X_{out}) \quad \text{s.t. } X_{out} \text{ is a sufficient statistic for } Y \quad (9.21)$$

or

$$\min_{X_{out} \in \mathcal{S}(Y)} \mathbb{I}(X_{in}; X_{out}) \quad (9.22)$$

where the set $\mathcal{S}(Y)$ is the set of sufficient statistics for Y . We here show that $X_{out} \in \mathcal{MS}(Y)$ (the set of minimal sufficient statistics) **iff**

$$\mathbb{I}(X_{in}; X_{out}) = \min_{P \in \mathcal{S}(Y)} \mathbb{I}(X_{in}; P). \quad (9.23)$$

in fact, from Definition 3, if $X_{out} \in \mathcal{MS}(Y)$ then $\forall P \in \mathcal{S}(Y), \exists f(\cdot) : X_{out} = f(P)$. By data processing inequality $\mathbb{I}(X_{in}; X_{out}) = \mathbb{I}(X_{in}; f(P)) \leq \mathbb{I}(X_{in}; P)$, that is thus equivalent to say that $\mathbb{I}(X_{in}; X_{out}) = \min_{P \in \mathcal{S}(Y)} \mathbb{I}(X_{in}; P)$. It remains to be shown that also the converse holds: if $\mathbb{I}(X_{in}; X_{out}) = \min_{P \in \mathcal{S}(Y)} \mathbb{I}(X_{in}; P)$, then $X_{out} \in \mathcal{MS}(Y)$. We proceed by absurd by assuming that $\mathbb{I}(X_{in}; X_{out}) = \min_{P \in \mathcal{S}(Y)} \mathbb{I}(X_{in}; P)$ and $X_{out} \notin \mathcal{MS}(Y)$. Since $X_{out} \notin \mathcal{MS}(Y)$ it exists a transformation $f(\cdot)$ that produces a minimal sufficient statistic: $Z = f(X_{out})$, $f \in \mathcal{MS}(Y)$. By data processing inequality we immediately see that $\mathbb{I}(Z; X_{in}) \leq \mathbb{I}(X_{in}; X_{out})$. We split the \leq into its $<$, $=$ components. If $\mathbb{I}(Z; X_{in}) < \mathbb{I}(X_{in}; X_{out})$ then the assumption $\mathbb{I}(X_{in}; X_{out}) = \min_{P \in \mathcal{S}(Y)} \mathbb{I}(X_{in}; P)$ is violated since $\mathcal{MS}(Y) \subset \mathcal{S}(Y)$. Considering instead $\mathbb{I}(Z; X_{in}) = \mathbb{I}(X_{in}; X_{out})$ to be true, this implies that also $X_{in} \rightarrow Z \rightarrow X_{out}$ forms a Markov Chain. Since moreover, by construction, from X_{out} we can re-obtain deterministically Z through the function $f(\cdot)$ the following holds $X_{in} \rightarrow Z \rightarrow X_{out} \rightarrow Z$. We can then say that it exists a deterministic

function $g(\cdot)$ that maps Z into X_{out} . This implies however that $X_{out} \in \mathcal{MS}(Y)$. In fact: X_{out} is a sufficient statistic and for every sufficient statistic P it exists a deterministic function h , we can choose $h = g \circ f$, such that $X_{out} = h(P)$. We then proved by absurd also the converse of (9.23).

Summarising, the following concept is the key of this discussion: provided knowledge of $\mathbb{I}(Y; X_{in})$ and of the corresponding $\bar{\beta}$, solving the Information bottleneck with $\beta = \bar{\beta}$, or finding a minimal sufficient statistic are equivalent problems. Finding a minimal sufficient statistic allows to compress as much as possible an observed random variable X_{in} while saving all the information about Y . Information theoretic compression can be easily translated into *physically required memory* to store a given random variable. The fundamental idea of this work is to hierarchically combine compressed versions of random variables while saving all the relevant information. On one side, combining different random variables increases the dimensionality of the problem, on the other, the performed compression reduces it. By balancing these opposing forces we will be able, Chapter 10,11, to design a new classification algorithm.

Two important details must be highlighted: first, the mutual information $\mathbb{I}(Y; X_{in})$ is unknown (and consequently $\bar{\beta}$), second, even having an estimate of such mutual information does not solve the problem. In an ideal scenario, where the estimation is very accurate, there is in fact no reason to choose any β different from the $\bar{\beta}$ corresponding to the mutual information (why throw away information about Y or compress less than necessary?). In practice, the estimation will be noisy and, under certain pathological conditions, much higher than the true value. This implies that, while in principle the value of β could be fixed a priori, in practice it remains an extremely important degree of freedom that must be tuned.

When working with a finite dataset, it is not possible to have access to the true mutual information but just an empirical estimate of it. Consider two random variables $X \in \mathcal{X}, Y \in \mathcal{Y}$ distributed according to $p_{X,Y}$ and a dataset of joint observations $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$. The simplest way to estimate the joint probability distribution is as follows:

$$\hat{p}_{X,Y}(x, y) = \frac{\sum_{i=1}^N \mathbb{1}(x_i = x, y_i = y)}{N}. \quad (9.24)$$

The empirical marginals are then easily derived as well. To estimate the mutual information is then sufficient to compute

$$\hat{\mathbb{I}}(X; Y) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \hat{p}_{X,Y}(x, y) \log \left(\frac{\hat{p}_{X,Y}(x, y)}{\hat{p}_X(x) \hat{p}_Y(y)} \right). \quad (9.25)$$

A practical example of spurious correlation that could induce a fake higher *empirical* mutual information is described hereafter. The following is not a formal proof. Consider two independent random variables X, Y uniformly distributed in

their sample space with cardinalities $|\mathcal{X}| = |\mathcal{Y}| = N_s$. When the number of observations is much smaller than the cardinalities $N \ll N_s$, the empirical joint probability distribution will be sparsely populated and provide the statistical illusion of false correlations. In such a scenario, it will be likely that the conditional probabilities $\hat{p}_{X|Y}(x, y)$ will be extremely concentrated and the estimated mutual information inflated. Using directly $\hat{\mathbb{I}}(X; Y)$ as a target mutual information in the Information Bottleneck algorithm could thus makes us incur into learning spurious correlations, also colloquially known as overfitting. By choosing a different target mutual information (and thus a different β) we will in practice be able to move across the underfitting/overfitting landscape (look also at Figure 11.6, Chapter 11).

9.2.2 Proof of IB solution

The following section is purely technical and serves the purpose of proving the fact that (11.10) describes the optimal conditional probability in terms of Information Bottleneck maximization.

We want to minimise

$$\mathcal{L} = \mathbb{I}(X_{in}; X_{out}) - \beta \mathbb{I}(Y; X_{out})$$

among the functionals $p(X_{out}|X_{in})$. To simplify the derivation, the mutual informations will be measured in *nats* (i.e. natural logarithm instead of \log_2). The mutual information $\mathbb{I}(X_{in}; X_{out})$ can be written as

$$\mathbb{I}(X_{in}; X_{out}) = \mathbb{H}(X_{out}) - \mathbb{H}(X_{out}|X_{in}) \quad (9.26)$$

Let us then find the derivative of $\mathbb{H}(X_{out})$ with respect to $p(X_{out} = p|X_{in} = q)$. For sake of brevity, in the following we will write $p(X_{out} = p|X_{in} = q) = p_{qp}$:

$$\frac{\partial \mathbb{H}(X_{out})}{\partial p_{qp}} = -\frac{\partial}{\partial p_{qp}} \sum_{j=1}^{N_{out}} p(X_{out} = j) \ln p(X_{out} = j)$$

Note that

$$p(X_{out} = j) = \sum_{i=1}^{N_{in}} p(X_{out} = j|X_{in} = i)p(X_{in} = i) \quad (9.27)$$

and the term p_{qp} appears only when $j = p$ and $i = q$, so that

$$\frac{p(X_{out} = j)}{dp_{qp}} = \begin{cases} 0 & j \neq p \\ p(X_{in} = q) & j = p \end{cases} \quad (9.28)$$

Remember also that

$$\frac{d}{dx}(x \ln x) = \ln x + 1$$

Therefore

$$\begin{aligned}\frac{\partial \mathbb{H}(X_{out})}{\partial p_{qp}} &= -\frac{\partial}{\partial p_{qp}} [p(X_{out} = p) \ln p(X_{out} = p)] \\ &= -[1 + \ln p(X_{out} = p)]p(X_{in} = q)\end{aligned}$$

The derivative of $\mathbb{H}(X_{out}|X_{in})$ is

$$\begin{aligned}\frac{\partial \mathbb{H}(X_{out}|X_{in})}{\partial p_{qp}} &= -\frac{\partial}{\partial p_{qp}} \sum_{j=1}^{N_{out}} \sum_{i=1}^{N_{in}} p(X_{out} = j|X_{in} = i)p(X_{in} = i) \ln p(X_{out} = j|X_{in} = i) \\ &= -\frac{\partial}{\partial p_{qp}} [p(X_{out} = p|X_{in} = q)p(X_{in} = q) \ln p(X_{out} = p|X_{in} = q)] \\ &= -p(X_{in} = q)[1 + \ln p(X_{out} = p|X_{in} = q)]\end{aligned}$$

The derivative of $\mathbb{I}(X_{in}; X_{out})$ is therefore

$$\begin{aligned}\frac{\partial \mathbb{I}(X_{in}; X_{out})}{\partial p_{qp}} &= -[1 + \ln p(X_{out} = p)]p(X_{in} = q) + p(X_{in} = q)[1 + \ln p(X_{out} = p|X_{in} = q)] \\ &= p(X_{in} = q) \ln \frac{p(X_{out} = p|X_{in} = q)}{p(X_{out} = p)}\end{aligned}$$

The mutual information $\mathbb{I}(Y; X_{out})$ can be written as

$$\mathbb{I}(Y; X_{out}) = \mathbb{H}(X_{out}) - \mathbb{H}(X_{out}|Y) \quad (9.29)$$

The derivative of the entropy of X_{out} was already found:

$$\begin{aligned}\frac{\partial \mathbb{H}(X_{out})}{\partial p_{qp}} &= -\frac{\partial}{\partial p_{qp}} [p(X_{out} = p) \ln p(X_{out} = p)] \\ &= -[1 + \ln p(X_{out} = p)]p(X_{in} = q)\end{aligned}$$

As for the conditional entropy $\mathbb{H}(X_{out}|Y)$, we can write it as

$$\mathbb{H}(X_{out}|Y) = -\sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} p(X_{out} = j, Y = m) \ln p(X_{out} = j|Y = m)$$

Then

$$\begin{aligned}
 \frac{\partial \mathbb{H}(X_{out}|Y)}{\partial p_{qp}} &= - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} \left\{ \frac{\partial}{\partial p_{qp}} p(X_{out} = j, Y = m) \right\} \ln p(X_{out} = j|Y = m) \\
 &\quad - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} p(X_{out} = j, Y = m) \left\{ \frac{\partial}{\partial p_{qp}} \ln p(X_{out} = j|Y = m) \right\} \\
 &= - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} \left\{ \frac{\partial}{\partial p_{qp}} p(X_{out} = j|Y = m) p(Y = m) \right\} \ln p(X_{out} = j|Y = m) \\
 &\quad - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} p(X_{out} = j, Y = m) \frac{1}{p(X_{out} = j|Y = m)} \left\{ \frac{\partial}{\partial p_{qp}} p(X_{out} = j|Y = m) \right\} \\
 &= - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} \left\{ \frac{\partial}{\partial p_{qp}} p(X_{out} = j|Y = m) \right\} p(Y = m) \ln p(X_{out} = j|Y = m) \\
 &\quad - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} p(Y = m) \left\{ \frac{\partial}{\partial p_{qp}} p(X_{out} = j|Y = m) \right\}
 \end{aligned}$$

We have

$$\begin{aligned}
 \frac{\partial}{\partial p_{qp}} p(X_{out} = j|Y = m) &= \frac{\partial}{\partial p_{qp}} \sum_{i=1}^{N_{in}} p(X_{out} = j, X_{in} = i|Y = m) \\
 &= \frac{\partial}{\partial p_{qp}} \sum_{i=1}^{N_{in}} p(X_{out} = j|X_{in} = i) p(X_{in} = i|Y = m) \\
 &= \delta_{iq} \delta_{jp} p(X_{in} = i|Y = m) = \delta_{jp} p(X_{in} = q|Y = m)
 \end{aligned}$$

and therefore

$$\begin{aligned}
 \frac{\partial \mathbb{H}(X_{out}|Y)}{\partial p_{qp}} &= - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} \delta_{jp} p(X_{in} = q|Y = m) p(Y = m) \ln p(X_{out} = j|Y = m) \\
 &\quad - \sum_{j=1}^{N_{out}} \sum_{m=1}^{N_{class}} p(Y = m) \delta_{jp} p(X_{in} = q|Y = m) \\
 &= - \sum_{m=1}^{N_{class}} p(X_{in} = q|Y = m) p(Y = m) \ln p(X_{out} = p|Y = m) \\
 &\quad - \sum_{m=1}^{N_{class}} p(Y = m) p(X_{in} = q|Y = m) \\
 &= - \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln p(X_{out} = p|Y = m) - p(X_{in} = q)
 \end{aligned}$$

In the overall,

$$\begin{aligned}
 \frac{\partial \mathbb{I}(Y; X_{out})}{\partial p_{qp}} &= \frac{\partial \mathbb{H}(X_{out})}{\partial p_{qp}} - \frac{\partial \mathbb{H}(X_{out}|Y)}{\partial p_{qp}} \\
 &= -p(X_{in} = q)[1 + \ln p(X_{out} = p)] \\
 &\quad + \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln p(X_{out} = p|Y = m) + p(X_{in} = q) \\
 &= -p(X_{in} = q) \ln p(X_{out} = p) + \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln p(X_{out} = p|Y = m) \\
 &= -\sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln p(X_{out} = p) \\
 &\quad + \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln p(X_{out} = p|Y = m) \\
 &= \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln \frac{p(X_{out} = p|Y = m)}{p(X_{out} = p)} \\
 &= \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln \frac{p(X_{out} = p, Y = m)}{p(Y = m)p(X_{out} = p)} \\
 &= \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln \frac{p(Y = m|X_{out} = p)}{p(Y = m)}
 \end{aligned}$$

Let us now set the derivative of \mathcal{L} with respect to p_{qp} equal to 0. We get:

$$\begin{aligned}
 \frac{\partial \mathcal{L}}{\partial p_{qp}} &= p(X_{in} = q) \ln \frac{p(X_{out} = p|X_{in} = q)}{p(X_{out} = p)} \\
 &\quad - \beta \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln \frac{p(Y = m|X_{out} = p)}{p(Y = m)} = 0
 \end{aligned}$$

which leads to:

$$\begin{aligned}
 p(X_{in} = q) \ln \frac{p(X_{out} = p|X_{in} = q)}{p(X_{out} = p)} &= \beta \sum_{m=1}^{N_{class}} p(X_{in} = q, Y = m) \ln \frac{p(Y = m|X_{out} = p)}{p(Y = m)} \\
 &= \beta p(X_{in} = q) \sum_{m=1}^{N_{class}} p(Y = m|X_{in} = q) \ln \frac{p(Y = m|X_{out} = p)}{p(Y = m)}
 \end{aligned}$$

and consequently

$$\begin{aligned} p(X_{out} = p|X_{in} = q) &= p(X_{out} = p) \exp \left\{ \beta \sum_{m=1}^{N_{class}} p(Y = m|X_{in} = q) \ln \frac{p(Y = m|X_{out} = p)}{p(Y = m)} \right\} \\ &= p(X_{out} = p) e^{-\beta d(q,p)} \end{aligned}$$

where

$$d(q, p) = \sum_{m=1}^{N_{class}} p(Y = m|X_{in} = q) \ln \frac{p(Y = m)}{p(Y = m|X_{out} = p)} \quad (9.30)$$

Actually, we did not introduce the constraints, namely:

$$\sum_{p=1}^{N_{out}} \sum_{q=1}^{N_{in}} p(X_{out} = p|X_{in} = q) p(X_{in} = q) = 1 \quad (9.31)$$

$$\sum_{p=1}^{N_{out}} p(X_{out} = p|X_{in} = q) = 1 \quad (9.32)$$

In the presence of the constraints, we get

$$p(X_{out} = p|X_{in} = q) = \frac{p(X_{out} = p)}{Z(q)} e^{-\beta d(q,p)}$$

where $Z(q)$ is such that $\sum_{p=1}^{N_{out}} p(X_{out} = p|X_{in} = q) = 1$ holds and (11.10) is proved.

9.3 Inference for categorical datasets

In the first part of the thesis, we treated the problem of Bayesian machine learning linked to differentiable parametric models. Usually, such kind of models, are intended to be applied to datasets whose domain is some subset (proper or improper) of the D -dimensional Euclidean space \mathbb{R}^D . The simplest example is the linear regression model in which the output, when the input is the vector $\mathbf{u} \in \mathbb{R}^D$, is simply computed as $y = \hat{\mathbf{x}}^\top \mathbf{u}$. The parameter vector $\hat{\mathbf{x}}$ is usually assumed to belong to \mathbb{R}^D and any form of gradient based estimation method can be applied as described in the previous part of the thesis.

On the contrary, a **categorical** random variable C is an element of a finite set of objects $\mathcal{C} = \{o_1, \dots, o_{N_c}\}$, $C \in \mathcal{C}$, in which an algebra does not make generally sense. The concept is best explained informally with an example: suppose $\mathcal{C} = \{\text{dog}, \text{cat}, \text{yellow}\}$. It is evident that it does not make sense, at least in a principled way, to consider algebraic manipulations of elements of the set such as $\text{dog} - \frac{\text{yellow}}{13}$. To overcome the problem, at least two broad possibilities are available: the first one is to follow the route of *embedding* the objects in some euclidean

space \mathbb{R}^D and use the classical parametric tools described in the first part of the thesis. The second one is to consider models in which no algebraic manipulation is needed and treat the datapoints as element of a finite discrete set on which it is not possible to perform algebraic manipulations. Member of the former class are notably the natural language (NLP, an example of implementation can be found in [94]) or the DNA (as in [2]). These cases are somehow an exception since usually building embeddings is an extremely difficult and data hungry task, both considered cases succeeded in such an approach due to the enormous available amount of data and the temporal structure of the datasets that helps in extracting structured information.

In this part of the thesis, instead, we follow the latter approach, that is more feasible for smaller, less regular datasets. Currently, in fact, in the applied machine learning domain the unwritten rule is that when the considered dataset is composed of categorical (or mixture of real and categorical) features, inherently categorical based methods outperform other real parametric models such as deep neural networks (with the notable exception of NLP and DNA processing).

9.3.1 Decision trees

The most important class of categorical methods is based on decision trees. A decision tree is a classifier, i.e. an object that takes as input a given D -dimensional datapoint \mathbf{x} (an element of the space $\mathcal{C}^{(1)} \times \mathcal{C}^{(2)} \dots \times \mathcal{C}^{(D)}$) and produces as output $y(\mathbf{x})$, one out of N_{cl} classes.

A decision tree is a graph (tree) where the root and intermediate nodes represent features, the routing algorithm from a parent to one of the child is determined by the value taken by the parent feature node, and the leaves contain the predicted class (or a probability for a predicted class). The construction of the tree topology is assumed for the moment to be known thanks to external sources. The concept is best explained with a toy example. Suppose that each datapoint is composed by three features: $\text{sex} \in \{\text{male}, \text{female}\}$, $\text{age} \in \{\text{young}, \text{adult}, \text{old}\}$ and $\text{green blood} \in \{\text{yes}, \text{no}\}$. The objective is to determine whether the considered subject (datapoint) is or is not affected by the XYZ-illness (healty, ill). Figure 9.1(left) depicts the decision tree corresponding to the problem. The classification rule (either externally fixed or somehow learnt) is the following: if you are a male (young or old) with green blood, then you have XYZ-illness, otherwise you are healthy. To classify a datapoint (a patient) it is then sufficient to follow the route in the tree according to the features value, starting from the root feature, in this case the green blood feature, and descend the tree until a leaf (classification value) is reached. Starting from the root, for example, if a subject has green blood the right path must be chosen. Then, similarly, according to the sex the left or right path is chosen, and so on.

Actually, the example just described it is somehow unrealistic in that the rules

are too rigid, fixed, and a leaf node contains a strict classification rule. It would be nicer, to have probabilities of being ill or healthy instead of an hard label , something like *the patient is healthy at 75%*. From a technical point of view this is obviously possible, and the result is depicted in Figure 9.1(right). Leaves where the probability of being healthy is higher than 0.5 are colored in blue, while leaves for which the probability of being ill is the dominant one are colored in red.

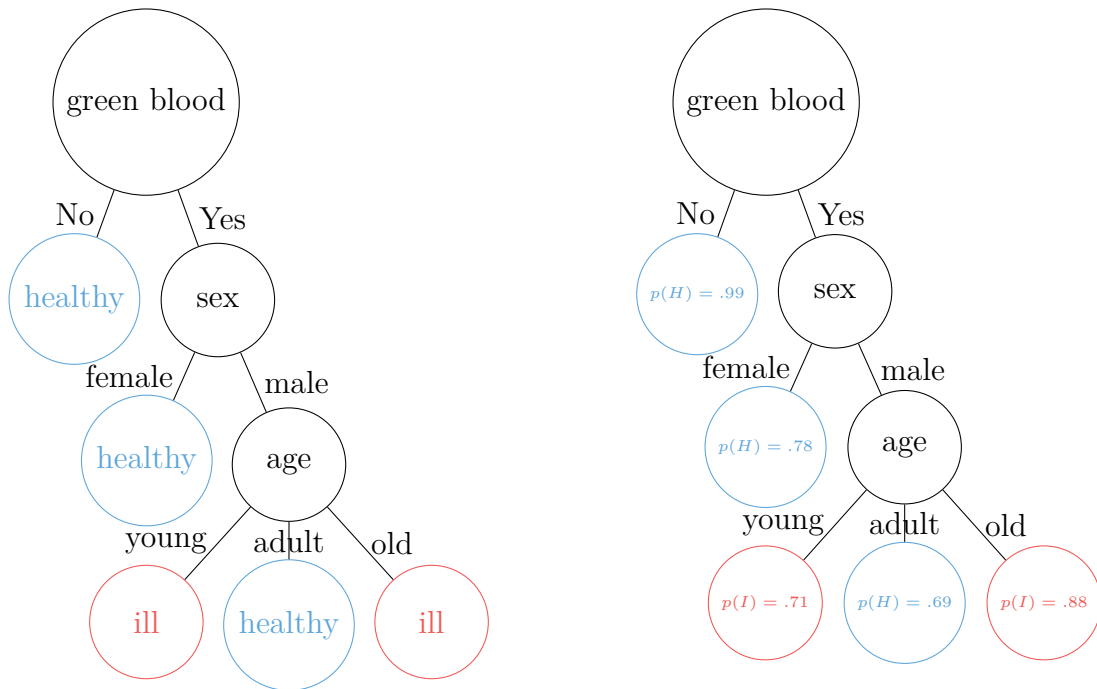


Figure 9.1: A decision tree (left). A soft decision tree (right), the events H, I are short hand for healthy and ill respectively.

9.3.2 Simple algorithm for growing (learning) a tree

It is not, by any means, the objective of this introduction to fully explain how decision trees can be built. We simply underline the general mechanism behind such construction algorithms, having in mind that we will explain the new proposed method in Chapters 10,11 and that to fully appreciate similarities and differences between the methods an introduction to decision trees is necessary.

The pseudocode is described in Algorithm 3. To avoid clutter only exploration is described, obviously one must keep track of the various paths while executing the algorithm to be able to use the stored tree subsequently. Our goal is to build a decision tree starting from a given dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, composed of features \mathbf{x}_i and targets y_i . We would like to build the tree according to information theoretic criteria.

Before proceeding with the exposition we notice again that the main element of such an approach, the true mutual information between any feature $X^{(a)} \in \mathcal{C}^{(a)}$ and the target class:

$$\mathbb{I}(X^{(a)}; Y) = \sum_{x \in \mathcal{C}^{(a)}, y \in \mathcal{Y}} p_{X^{(a)}, Y}(x, y) \log \left(\frac{p_{X^{(a)}, Y}(x, y)}{p_{X^{(a)}}(x)p_Y(y)} \right), \quad (9.33)$$

to be computed requires the knowledge of the true distributions $p_{X^{(a)}, Y}(x, y)$. These probabilities are in general unknown. To overcome the problem, and this is the hearth of the learning process, the simplest solution is to compute instead the *empirical* mutual information based on the *empirical* probabilities. The estimated empirical joint probabilities are computed as

$$\hat{p}_{X^{(a)}, Y}(x, y) = \frac{\sum_{i=1}^N \mathbb{1}(x_i^{(a)} = x, y_i = y)}{N}, \quad (9.34)$$

and similarly for the marginals $\hat{p}_{X^{(a)}}(x), \hat{p}_Y(y)$. A straightforward mechanism to estimate the mutual information is, starting from (9.33),

$$\hat{\mathbb{I}}(X^{(a)}; Y) = \sum_{x \in \mathcal{C}^{(a)}, y \in \mathcal{Y}} \hat{p}_{X^{(a)}, Y}(x, y) \log \left(\frac{\hat{p}_{X^{(a)}, Y}(x, y)}{\hat{p}_{X^{(a)}}(x)\hat{p}_Y(y)} \right). \quad (9.35)$$

Having clarified this technicality, we can proceed in the exposition of the tree learning procedure. Let us slightly rearrange the notation for the dataset \mathcal{D} , and

indicate with $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix}$, an $N \times D$ matrix containing the D features of the dataset,

and with $\mathbf{y} = [y_1, \dots, y_N]^\top$ the vector containing the target variables. We use the short-hand $i(\mathbf{y}, \mathbf{X}(:, s))$ to indicate the empirical mutual information between the target variable vector \mathbf{y} and the s -th column of the dataset, $\mathbf{X}(:, s)$.

The tree is built by calling the recursive function `BUILD TREE`(\mathbf{y}, \mathbf{X}) described in Algorithm 3. The summary is the following: starting from the complete dataset pick the feature with the highest empirical mutual information. Divide the dataset according to the possible values taken by such feature and create children of this root node. For each children repeat recursively the procedure until a termination condition is reached (there are no left features or the sub-dataset targets belong to a unique class).

Algorithm 3 Recursive decision tree construction

```

BUILD TREE ( $\mathbf{y}, \mathbf{X}$ ):
Select  $p = \arg \max_s i(\mathbf{y}, \mathbf{X}(:, s))$ 
loop For all  $a \in \mathcal{C}^{(s)}$ :
    Select the rows ( $\{j\}$ ) of  $\mathbf{X}$  and  $\mathbf{y}$  for which  $\mathbf{X}(j, p) = a$ 
    Remove  $\{j\}$  rows from  $\mathbf{y}, \mathbf{X}$ 
    Remove the  $p$  – th column from  $\mathbf{X}$ 
    Call the reduced dataset ( $\mathbf{y}\{p\}, \mathbf{X}\{p\}$ )
    Compute number of columns of  $\mathbf{X}\{p\}$  ( $N_f$ )
    Compute number of unique values of vector  $\mathbf{y}\{p\}$  ( $N_u$ )
    if  $N_f = 0$  or  $N_u = 1$  then
        Build a leaf node
        Store empirical distribution of  $\mathbf{y}\{p\}$ 
    else BUILD TREE ( $\mathbf{y}\{p\}, \mathbf{X}\{p\}$ )

return

```

Many different variants can be derived, as described in [70, 71], based on different criteria. Instead of choosing the feature with the maximum mutual information, a common alternative is to use the Gini impurity coefficient. The Gini impurity of a dataset is defined as *the probability that a randomly chosen datapoint is misclassified if the classification label is randomly chosen according to the label probability distribution*:

$$g(\mathcal{D}) = \sum_{y \in \mathcal{Y}} \hat{p}_Y(y) \sum_{p \in \mathcal{Y}, p \neq y} \hat{p}_Y(p) \quad (9.36)$$

The Gini impurity of a feature is simply derived as the weighted sum of Gini impurities when the dataset is split according to the feature value. In practice, if the s – *th* feature is chosen, the dataset is split according to the values taken by the feature, $a \in \mathcal{C}^{(s)}$, and the Gini impurity for the feature is computed as

$$g_f(s) = \sum_{a \in \mathcal{C}^{(s)}} |\mathcal{D}_a| g(\mathcal{D}_a) \quad (9.37)$$

where \mathcal{D}_a is the portion of the dataset \mathcal{D} in which the s – *th* features has the value a , and $|\mathcal{D}_a|$ is its cardinality. It is simple to rewrite (9.37) in terms of empirical probability and draw a parallel with the splitting method based on Mutual Information

9.3.3 Ensemble of trees: random forests

Conceptually, what has been described in the previous Section is sufficient to understand the functionality and the learning mechanisms behind trees. Many

important details are left out, such as the possibility of having pruning procedures, or the discussion about hardware implementation and computational and memory complexity. While extremely important, we do consider these aspects outside the scope of this dissertation, and refer instead the interested reader to one of the many good resources on the topic [69, 70].

One important aspect that we consider worthy of discussion is the concept of *ensemble* of trees. To build an ensemble of trees, usually, the dataset is randomly splitted into overlapping sub-datasets, a decision tree is built and their results are combined, giving rise to the name *random forest*. The first time the name random forest appeared in the literature was in [90].

The author in [90], based on previous experience on aggregating multiple independent classifiers [39], proposes to combine multiple trees built on the possible 2^D subdatasets, i.e. every possible combination of taking or not a given feature out of the D available.

Construction of random forests is based on the combination of two stochastic operations:

1. Random selection of features, that as just described corresponds to building multiple copies of the dataset and throw away some of the features
2. Bagging, a short-hand for **bootstrap aggregating**, that corresponds to the operation of building multiple datasets from a single one by sampling with replacement the datapoints

In practice, the combination of the two operations just described, corresponds to a random sub-sampling with replacement of the dataset $\mathcal{D} = \mathbf{X}, \mathbf{y}$ obtained by deletion of some of the N rows and D columns of the matrix \mathbf{X} . The procedure for the learning of random forest is described in Algorithm 4. It is simply composed by three elements: the generation of many random subsets of the original dataset, the construction of trees using the already described Algorithm 3, and the construction of an object that aggregates all the single classification probabilities from the single trees. Concerning this last operation, a simple and common choice, that is also the one described in [90], is to simply average the result of the single trees

$$\hat{p}_{forest}(y|\mathbf{x}) = N_{tr}^{-1} \sum_{i=1}^{N_{tr}} \hat{p}_{tree}^i(y|\mathbf{x}), \quad (9.38)$$

where N_{tr} is the number of trees composing the forest.

Algorithm 4 Random forest construction

RANDOM FOREST (\mathbf{y}, \mathbf{X}):**loop** For number of decision trees N_{tr} : Randomly select rows of \mathbf{X} and \mathbf{y} Randomly remove columns of \mathbf{X} Call the reduced dataset as $(\tilde{\mathbf{y}}, \tilde{\mathbf{X}})$ BUILD TREE $(\tilde{\mathbf{y}}, \tilde{\mathbf{X}})$ (Algorithm 3)Build aggregator of classification probabilities for the single trees

This brief introduction to random forest will be useful in Chapter 11, where a form of ensembling of multiple copies of the same basic network will be used to build stronger classifiers. Important differences with respect to the basic methodology just described will be underlined.

Chapter 10

Deep Information Networks

This chapter is entirely focused on the presentation of the proposed algorithm (Deep Information Networks, DIN [25]) and on its properties. Differently from decision trees, where the decision is routed by considering features one at the time, and already considered features are discarded (actually this step in certain variants of trees is avoided), in the DIN framework the "tree" is inverted in that the root is the target variable and the leaves are the features. Starting from the joint observation of all features for a single datapoints, equivalent features are obtained by merging together the initial features in a recursive, hierarchical manner. The algorithm stops when a unique equivalent feature is remaining, at the root, and a decision is taken.

10.1 Introduction

The so-called "information bottleneck" was described in [92, 91], where it was employed to analyse how the information flows inside a classification neural network. We here propose to exploit the same idea to build a supervised learning machine that compresses the input data and generates the estimated class, using a modular structure with a tree topology.

The key element of the proposed Deep Information Network (DIN) is the information node, that compresses the input while keeping its information content: during the training phase the node evaluates the conditional probabilities $P(X_{out} = j | X_{in} = i)$ that minimise the mutual information $\mathbb{I}(X_{in}; X_{out})$ between its input X_{in} and output X_{out} for a given mutual information between its output and the target $\mathbb{I}(X_{out}; Y)$. The input of each information node at the first layer is one of the features/attributes of the data; once the conditional probabilities are found, the node randomly generates its output according to them.

The second element of the DIN is the combiner that merge the outputs of two or

more information nodes to generate the input of the subsequent one. The machine is therefore made of several layers, and the number of information nodes per layer decreases from layer to layer, thanks to the presence of combiners, until a unique information node is obtained in the last layer, whose task is to provide the estimated class of the input.

Thus a DIN has a layered architecture, similarly to a neural network, but the working principle is completely different. In particular, there is no global objective function and each information node is separately trained, using only its input and the target class. The layers are trained in order: the nodes of the first layer are trained using the available raw data (one node for each of the available features) and stochastically generate the data for the combiners that feed the second layer; the nodes at the second layer are trained using their input (ideally the information contained in two or more features in the raw data) and the target, etc. Each information node has no knowledge about the input and output of other nodes. From layer to layer, the information nodes and the combiners allow to extract the information content of the original data as if they were sifting it. Moreover, differently from neural networks where end to end backpropagation is necessary, when training a DIN the flow of data is in the forward direction only. Whereas algorithm C4.5 by Quinlan [70] builds a hierarchical decision tree starting from the root, we here propose a way to build a similar tree, but starting from the leaves.

The advantages of DINs are clear: extreme flexibility, high modularity, complexity that linearly depends on the number of nodes, overall number of nodes that linearly depends on the number of features of the data. Since the machine works with information theory, categorical and missing data are easily managed; on the other hand, continuous data should be first quantized, but the true quantization can be left to the first layer information nodes.

What has to be assessed for the proposed machine is the performance: does it provide good results in terms of misclassification probability? are there constraints on its parameters?

Section 10.2 more precisely describes the structure of the DIN, section 10.3 gives some insight on the theoretical properties and section 10.4 comments the results obtained with a simple dataset. Conclusions are finally drawn in section 10.5.

10.2 The information network

As pointed out in 10.1, the machine is made of information nodes, described in section 10.2.1, and combiners joined together through a network described in section 10.2.2.

10.2.1 The information node

Each information node, described in Fig. 10.1, has an input vector \mathbf{x}_{in} , whose elements take values in a set of cardinality N_{in} , and an output vector \mathbf{x}_{out} , whose elements take values in a set of cardinality N_{out} ; moreover the target class vector \mathbf{y}_{train} is available as input, with elements in the set $[0, N_{class} - 1]$; all the vectors have the same size N_{train} . By setting $N_{out} < N_{in}$, the information node performs a compression (source encoding). Let X_{in} (X_{out} , Y) identify the random variable whose samples are stored in vector \mathbf{x}_{in} (\mathbf{x}_{out} , \mathbf{y}).

In the training phase, the information node randomly generates the output vector \mathbf{x}_{out} , according to the conditional probability distribution that satisfies equation [92]

$$P(X_{out} = j | X_{in} = i) = \frac{1}{Z(i; \beta)} P(X_{out} = j) e^{-\beta d(i, j)}, \quad (10.1)$$

with $i = 0, \dots, N_{in} - 1, j = 0, \dots, N_{out} - 1$, where

- $P(X_{out} = j)$ is the probability mass function of the output random variable X_{out}

$$P(X_{out} = j) = \sum_{i=0}^{N_{in}-1} P(X_{in} = i) P(X_{out} = j | X_{in} = i), \quad j = 0, \dots, N_{out} - 1 \quad (10.2)$$

- $d(i, j)$ is the Kullback-Leibler divergence

$$\begin{aligned} d(i, j) &= \sum_{m=0}^{N_{class}-1} P(Y = m | X_{in} = i) \log_2 \frac{P(Y = m | X_{in} = i)}{P(Y = m | X_{out} = j)} \\ &= \mathbb{KL}(P(Y | X_{in} = i) || P(Y | X_{out} = j)) \end{aligned} \quad (10.3)$$

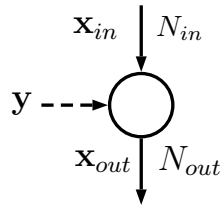


Figure 10.1: Schematic representation of an information node, showing the input and output vectors, with alphabets of cardinality N_{in} and N_{out} , respectively, and the target vector \mathbf{y} which is available during the training phase.

and

$$P(Y = m|X_{out} = j) = \sum_{i=0}^{N_{in}-1} P(Y = m|X_{in} = i)P(X_{in} = i|X_{out} = j),$$

$$m = 0, \dots, N_{class} - 1, j = 0, \dots, N_{out} - 1 \quad (10.4)$$

- β is a real positive parameter
- $Z(i; \beta)$ is a normalizing coefficient that allows to get

$$\sum_{j=1}^{N_{out}-1} P(X_{out} = j|X_{in} = i) = 1 \quad (10.5)$$

The probabilities $P(X_{out} = j|X_{in} = i)$ can be iteratively found using the Blahut-Arimoto algorithm [92].

According to [92], Eqn. (11.10) solves the information bottleneck: it minimises the mutual information $\mathbb{I}(X_{in}; X_{out})$ under the constraint of a given mutual information $\mathbb{I}(Y; X_{out})$. In particular, eqn. (11.10) is the solution of the minimisation of the Lagrangian

$$\mathcal{L} = \mathbb{I}(X_{in}; X_{out}) - \beta \mathbb{I}(Y; X_{out}) \quad (10.6)$$

If the Lagrangian multiplier β is increased, then the constraint is privileged and the information node tends to maximize the mutual information between its output X_{out} and the class Y ; if β is reduced, then minimisation of $\mathbb{I}(X_{in}; X_{out})$ is obtained, thus a compression is obtained. The information node must actually balance compression from X_{in} to X_{out} and propagation of the information on Y , in order to correctly decide its value. In our implementation, the compression is also imposed by the fact that the cardinality of the output alphabet N_{out} is smaller than that of the input alphabet N_{in} .

10.2.2 The network

Fig. 11.4 shows an example of a DIN, where we assume that the dataset is made of a matrix \mathbf{X} with N rows and $D = 8$ columns (features) and the corresponding class vector \mathbf{y} ; we use N_{train} rows for the training phase and $N_{test} = N - N_{train}$ rows for the testing phase, getting matrices \mathbf{X}_{train} and \mathbf{X}_{test} (see Fig. 10.3).

Now refer to Fig. 11.4. The k -th column $\mathbf{x}(k)$ of matrix \mathbf{X}_{train} should be, together with vector \mathbf{y}_{train} , the input of the information node $(k,0)$ at layer 0 of the structure. However the algorithm needs a finite number $N_{in}^{(0)}$ of values in the input vectors, and a pre-processing quantization phase is needed. Quantization (uniform, Max-Lloyd, etc.) is not discussed here, in the example described in Sect 10.4 we used linear quantization. Thus the true input of node $(k,0)$ is $\mathbf{x}_q(k)$, the quantized

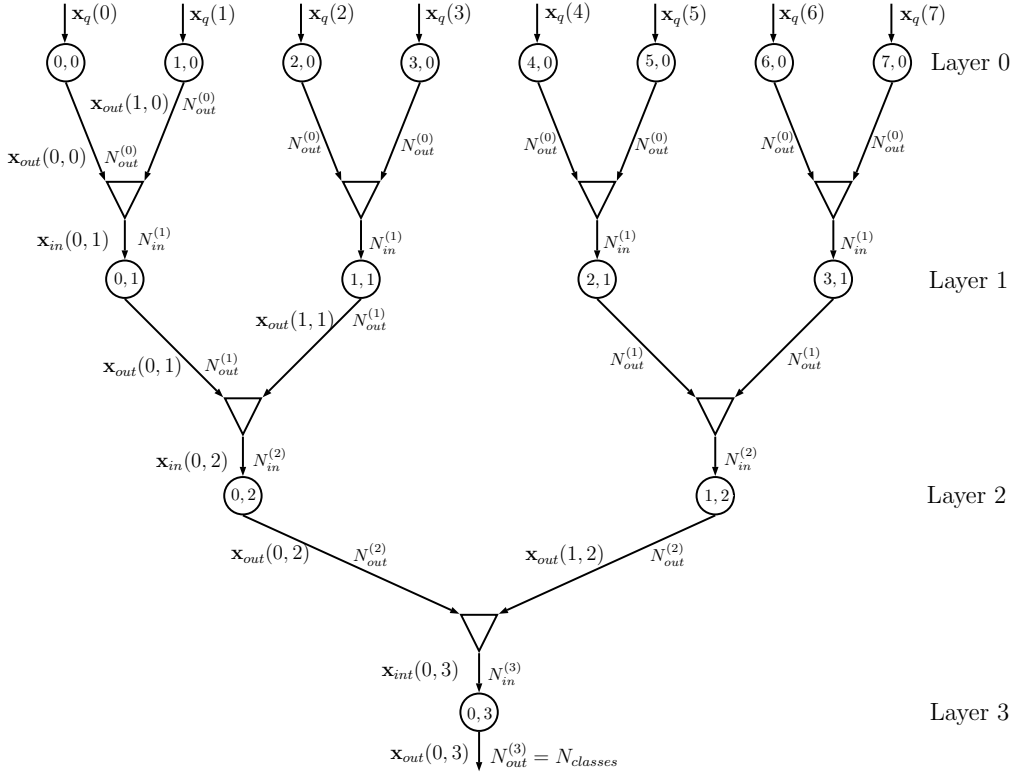


Figure 10.2: Architecture of a simple information node network for $D = 8$: each info node is represented by a circle, the numbers inside the circle identify the node, the triangles identify the combiners, $N_{in}^{(k)}$ is the number of values taken by the input of the info node at layer k , $N_{out}^{(k)}$ is the number of values taken by the output of the info node at layer k .

version of $\mathbf{x}(k)$. Of course quantization is not necessary for categorical data. The number of nodes at layer 0 of the structure is D , equal to the number of columns of \mathbf{X} .

Information node $(k,0)$ at layer 0 thus processes $\mathbf{x}_q(k)$ and \mathbf{y}_{train} , and generates the output vector $\mathbf{x}_{out}(k,0)$ with alphabet of cardinality $N_{out}^{(0)}$. Note that this output vector is randomly generated by the information node, according to its probability matrix $\mathbf{P}_{k,0}$, whose element i, j is $P(X_{out} = j | X_{in} = i)$ as found by the algorithm described in section 10.2.1.

The output vectors $\mathbf{x}_{out}(2k,0)$ and $\mathbf{x}_{out}(2k+1,0)$ are combined together by a combiner (shown as a triangle in Fig. 11.4) that outputs

$$\mathbf{x}_{in}(k,1) = \mathbf{x}_{out}(2k,0) + N_{out}^{(0)} \mathbf{x}_{out}(2k+1,0) \quad (10.7)$$

Vector $\mathbf{x}_{in}(k,1)$ is now the input of the information node $(k,1)$ of layer 1, and has

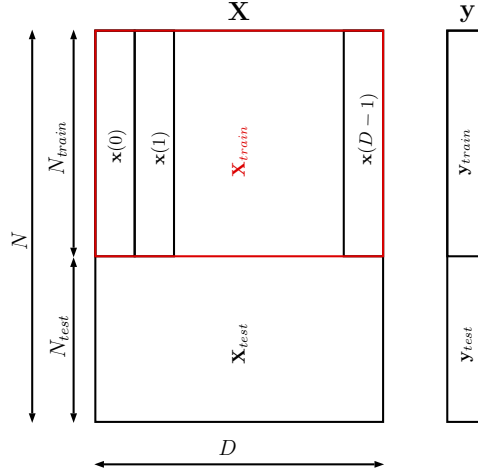


Figure 10.3: Definition of matrices, vectors and their dimensions

an alphabet with cardinality

$$N_{in}^{(1)} = N_{out}^{(0)} \times N_{out}^{(0)} \quad (10.8)$$

The topology is iterated halving the number of information nodes from layer to layer, until an isolated information node is obtained at layer $d = \log_2(D)$, that has input vector $\mathbf{x}_{in}(0, d)$ with alphabet cardinality $N_{in}^{(d)}$ and output vector $\mathbf{x}_{out}(0, d)$, whose alphabet cardinality must be $N_{out}^{(d)} = N_{class}$, the number of classes in the target vector \mathbf{y}_{train} . If the parameters are correctly chosen, the output vector $\mathbf{x}_{out}(0, d)$ is equal to \mathbf{y}_{train} , and the probability matrix $\mathbf{P}_{d,0}$ has just one value equal to 1 in each of its rows. A tree topology is thus obtained.

In the overall, the topology proposed in Fig. 11.4 requires a number of information nodes equal to

$$N_{nodes} = D + \frac{D}{2} + \frac{D}{4} + \cdots + 2 + 1 = 2D - 1 \quad (10.9)$$

and a number of combiners equal to

$$N_{combiners} = \frac{D}{2} + \frac{D}{4} + \cdots + 2 + 1 = D - 1 \quad (10.10)$$

All the nodes run exactly the same algorithm and all the combiners are equal, apart from the input/output vector alphabet cardinalities. If the cardinalities of the alphabets are all equal, i.e. $N_{in}^{(i)}$ and $N_{out}^{(i)}$ do not depend on the layer i , then all the nodes and all the combiners are exactly equal, which might help in a possible hardware implementation.

10.2.3 The testing/running phase

During the testing or running phase, the quantized columns of matrix \mathbf{X}_{test} are used as inputs and each node simply generates the output vector \mathbf{x}_{out} according to the input vector \mathbf{x}_{in} , using the probability matrix \mathbf{P} that has been optimized during the training phase. For example, if the n -th value stored in the vector $\mathbf{x}_{in}(k, \ell)$ of node k, ℓ is i , then the n -th value in vector $\mathbf{x}_{out}(k, \ell)$ is j with probability $\mathbf{P}_{k, \ell}(i, j)$. Each node then generates a stochastic output, according to its probability matrix.

10.3 Analysis

In the following, the DIN is studied considering first the overall conditional probabilities between the input features and the output, and second the information flow inside it.

10.3.1 The probabilistic point of view

Consider the case of the sub-network made of nodes a, b, c as shown in Fig. 11.3; the alphabet cardinality at the input of nodes a and b is N_0 , the cardinality at the output of nodes a and b is N_1 , the alphabet cardinality at the input of node c is $N_1 \times N_1$, the cardinality at its output is N_2 . Node a is characterized by matrix \mathbf{P}_a , whose element $P_a(i, j)$ is $P(X_{out,a} = j | X_{in,a} = i)$; similar definitions hold for \mathbf{P}_b and \mathbf{P}_c . Note that \mathbf{P}_a and \mathbf{P}_b have N_0 rows and N_1 columns, whereas \mathbf{P}_c has $N_1 \times N_1$ rows and N_2 columns; the overall probability matrix between the inputs $X_{in,a}, X_{in,b}$ and the output $X_{out,c}$ is \mathbf{P} with $N_0 \times N_0$ rows and N_2 columns. Then

$$\begin{aligned}
 & P(X_{out,c} = i | X_{in,a} = j, X_{in,b} = k) \\
 &= \sum_{r=0}^{N_1-1} \sum_{s=0}^{N_1-1} P(X_{out,c} = i, X_{out,a} = r, X_{out,b} = s | X_{in,a} = j, X_{in,b} = k) \\
 &= \sum_{r=0}^{N_1-1} \sum_{s=0}^{N_1-1} P(X_{out,c} = i | X_{out,b} = r, X_{out,c} = s) P(X_{out,a} = r | X_{in,a} = j) P(X_{out,b} = s | X_{in,b} = k) \\
 &= \sum_{r=0}^{N_1-1} \sum_{s=0}^{N_1-1} P(X_{out,c} = i | X_{out,b} = r, X_{out,c} = s) \mathbf{P}_a(j, r) \mathbf{P}_b(k, s) \tag{10.11}
 \end{aligned}$$

It can be shown that

$$\mathbf{P} = (\mathbf{P}_a \otimes \mathbf{P}_b) \mathbf{P}_c \tag{10.12}$$

where \otimes identifies the Kronecker matrix multiplication; note that $\mathbf{P}_a \otimes \mathbf{P}_b$ has $N_0 \times N_0$ rows and $N_1 \times N_1$ columns. By iteratively applying the above rule, we can get the expression of the overall matrix \mathbf{P} for the exact topology of Fig. 11.4, with

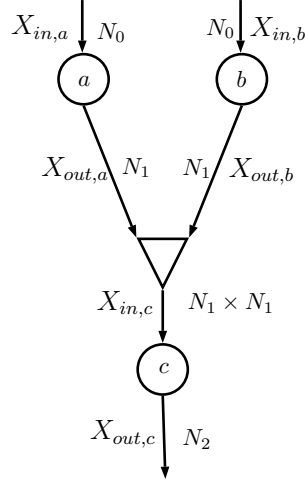


Figure 10.4: Sub-network used for the evaluation of the probability matrix; $X_{in,a}$, $X_{out,a}$, $X_{in,b}$, $X_{out,b}$, $X_{in,c}$, $X_{out,c}$ are all random variables; N_0 is the number of values taken by $X_{in,a}$ and $X_{in,b}$, N_1 is the number of values taken by $X_{out,a}$ and $X_{out,b}$, N_2 is the number of values taken by $X_{out,c}$.

8 input nodes and four layers:

$$\mathbf{P} = \left[\left\{ \left[(\mathbf{P}_{0,0} \otimes \mathbf{P}_{1,0}) \mathbf{P}_{0,1} \right] \otimes \left[(\mathbf{P}_{2,0} \otimes \mathbf{P}_{3,0}) \mathbf{P}_{1,1} \right] \right\} \mathbf{P}_{0,2} \right. \\ \left. \otimes \left\{ \left[(\mathbf{P}_{4,0} \otimes \mathbf{P}_{5,0}) \mathbf{P}_{2,1} \right] \otimes \left[(\mathbf{P}_{6,0} \otimes \mathbf{P}_{7,0}) \mathbf{P}_{3,1} \right] \right\} \mathbf{P}_{1,2} \right] \mathbf{P}_{0,3} \quad (10.13)$$

The DIN then behaves like a one-layer system that generates the output according to matrix \mathbf{P} , whose size might be impractically large (with D features all quantized with N_0 levels, matrix \mathbf{P} has size $N_0^D \times 2$). The proposed layered structure needs smaller probability matrices, which makes the system computationally efficient.

10.3.2 The information theory point of view

Each information node compresses the data and therefore, by a basic application of data processing inequality, we have

$$\mathbb{I}(X_{out}; Y) \leq \mathbb{I}(X_{in}; Y) \quad (10.14)$$

Consider now the outputs $X_{out}(2k, i)$ and $X_{out}(2k + 1, i)$ of two nodes at layer i that are combined by the combiner to generate $X_{in}(k, i + 1)$. It is interesting for analysis and integrity purposes to derive an upper and a lower bound of the total

amount of information obtained at the output of the combiner. First of all notice that, from an information theoretic point of view, we can write

$$\mathbb{I}(X_{in}(k, i + 1); Y) = \mathbb{I}([X_{out}(2k, i), X_{out}(2k + 1, i)]; Y) \quad (10.15)$$

We can expand (10.15) as

$$\begin{aligned} \mathbb{I}([X_{out}(2k, i), X_{out}(2k + 1, i)]; Y) &= \mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]) - \mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]|Y) \\ &= \mathbb{H}(X_{out}(2k + 1, i)|X_{out}(2k, i)) + \mathbb{H}(X_{out}(2k, i)) \\ &\quad - (\mathbb{H}(X_{out}(2k + 1, i)|X_{out}(2k, i), Y) - \mathbb{H}(X_{out}(2k, i)|Y)) \\ &= \mathbb{I}(X_{out}(2k, i); Y) + \mathbb{I}(X_{out}(2k + 1, i)|X_{out}(2k, i); Y) \quad (10.16) \\ &= \mathbb{I}(X_{out}(2k + 1, i); Y) + \mathbb{I}(X_{out}(2k, i)|X_{out}(2k + 1, i); Y) \quad (10.17) \end{aligned}$$

where the last equality is due to evident symmetries in the equations. Combining (10.16) and (10.17), and due to non-negativity of mutual information, we obtain

$$\mathbb{I}(X_{in}(k, i + 1); Y) \geq \max\{\mathbb{I}(X_{out}(2k, i); Y), \mathbb{I}(X_{out}(2k + 1, i); Y)\} \quad (10.18)$$

where equality is attained when $\mathbb{H}(X_{out}(2k + 1, i)|X_{out}(2k, i)) = 0$ (i.e. when $X_{out}(2k + 1, i)$ can be exactly predicted if $X_{out}(2k, i)$ is known).

We can also derive an upper bound for the total mutual information. First we observe that

$$\mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]) \leq \mathbb{H}(X_{out}(2k, i)) + \mathbb{H}(X_{out}(2k + 1, i)) \quad (10.19)$$

$$\mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]|Y) \geq \max\{\mathbb{H}(X_{out}(2k, i)|Y), \mathbb{H}(X_{out}(2k + 1, i)|Y)\} \quad (10.20)$$

and thus we can derive, using (10.19) and (10.20), that

$$\begin{aligned} \mathbb{I}(X_{in}(k, i + 1); Y) &= \mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]) - \mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]|Y) \\ &\leq \mathbb{H}([X_{out}(2k, i), X_{out}(2k + 1, i)]) - \mathbb{H}(X_{out}(2k, i)|Y) \\ &\leq \mathbb{H}(X_{out}(2k, i)) + \mathbb{H}(X_{out}(2k + 1, i)) - \mathbb{H}(X_{out}(2k, i)|Y) \\ &= \mathbb{I}(X_{out}(2k, i); Y) + \mathbb{H}(X_{out}(2k + 1, i)) \quad (10.21) \end{aligned}$$

$$= \mathbb{I}(X_{out}(2k + 1, i); Y) + \mathbb{H}(X_{out}(2k, i)) \quad (10.22)$$

where again the last equality is due to symmetry. We can then combine (10.21) and (10.22) in

$$\begin{aligned} \mathbb{I}(X_{in}(k, i + 1); Y) &\leq \\ &\min\{\mathbb{I}(X_{out}(2k, i); Y) + \mathbb{H}(X_{out}(2k + 1, i)), \mathbb{I}(X_{out}(2k + 1, i); Y) + \mathbb{H}(X_{out}(2k, i))\} \quad (10.23) \end{aligned}$$

Finally, combining (10.18) and (10.23)

$$\begin{aligned} & \max\{\mathbb{I}(X_{out}(2k, i); Y), \mathbb{I}(X_{out}(2k + 1, i); Y)\} \leq \\ & \mathbb{I}(X_{in}(k, i + 1); Y) \leq \\ & \min\{\mathbb{I}(X_{out}(2k, i); Y) + \mathbb{H}(X_{out}(2k + 1, i)), \mathbb{I}(X_{out}(2k + 1, i); Y) + \mathbb{H}(X_{out}(2k, i))\} \end{aligned} \quad (10.24)$$

obtaining an upper and lower bound for the mutual information.

10.4 The Kidney Disease experiment

In this section we present the results of an experiment performed on the UCI Kidney Disease dataset [93],[75]. The dataset has a total of 24 medical features, consisting of mixed categorical, integer and real values, with missing values present. The aim of the experiment is to correctly classify patients affected by chronic kidney disease. Remember that the machine works also with missing values, since they are seen as a special categorical value, whose probability is the only required measure.

In the proposed information network, layer zero has as many information nodes as the number of features (i.e. 24), and the input of each node is one of the quantized features; these nodes are trained in parallel. Then the outputs of layer zero are mixed two at a time and they become the input of layer one with 12 information nodes. These 12 nodes are merged into 6 nodes into layer two and the 6 nodes are mixed into 3 nodes in layer three. Finally the three nodes are combined into the unique final node whose output cardinality is equal to 2 (corresponding to ill or healthy). The 24 input features are uniformly quantized with different values of $N_{in}^{(0)}$, depending on the feature (from 2 to 470); on the contrary the value of N_{out} is 3 for all the nodes apart from the last one. The combiners generate vectors taking values in the range $[0,8]$, and the information nodes compress these values in the range $[0,2]$. The value of β is equal to 5; the number of training points is equal to $N_{train} = 200$ (half of which positive), the number of testing points is again $N_{test} = 200$.

Figure 10.5 shows how the mutual information $\mathbb{I}(X; Y)$ evolves in the DIN, and it is possible to appreciate that it increases from the first to the last layer. In particular $\mathbb{I}(X_{out}(2k, i); Y)$ and $\mathbb{I}(X_{out}(2k + 1, i); Y)$, represented as circles, are linked through segments to $\mathbb{I}(X_{in}(k, i + 1); Y)$, represented as a triangle, to show that the lower bound in (10.24) holds from $i = 0$ (blue markers), to $i = 3$ (black markers). Figure 10.5 also shows that $\mathbb{I}(X_{in}(k, 1); Y)$ (blue triangle) is higher than $\mathbb{I}(X_{out}(k, 1); Y)$ (green circle at the same x-value), which is due to the compression performed by the information node, that reduces the cardinality of $X_{in}(k, 1)$ to just three values; this is true in general when $\mathbb{I}(X_{in}(k, i); Y)$ is compared to $\mathbb{I}(X_{out}(k, i); Y)$ for all values of k and i .

In terms of performance, the results listed in table 10.1 were obtained:

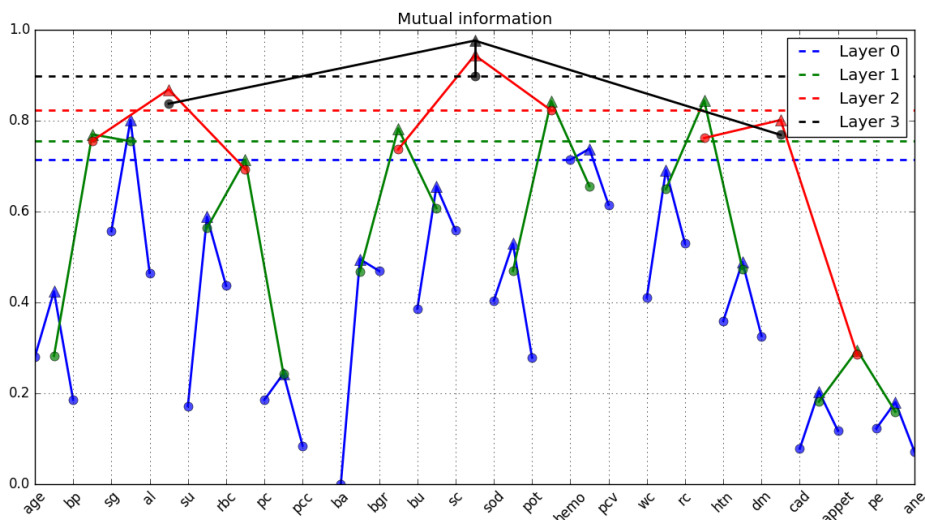


Figure 10.5: Mutual information in the network: circles correspond to inputs of the combiners, triangles to their outputs, each dotted line represents the maximum value of the mutual information at the input of the combiners of a given layer. Results obtained during one run of the training phase with $N_{train} = 200$ (half ill, half healthy), $N_{out}^{(i)} = 3$.

	$N_{out}^{(i)}$	N_{train}	N_{test}	Accuracy	Sensitivity	Specificity	F1 score
training	3	200	200	0.9970	0.9948	0.9993	0.9970
testing	3	200	200	0.9303	0.9188	0.9343	0.9260
training	2	200	200	0.9924	0.9985	0.9963	0.9974
testing	2	200	200	0.9648	0.9400	0.9734	0.9560
training	3	320	80	0.9947	0.9868	0.9996	0.9932
testing	3	320	80	0.9705	0.9435	0.9875	0.9647
training	2	320	80	0.9924	0.9820	0.9986	0.9902
testing	2	320	80	0.9762	0.9511	0.9918	0.9709

Table 10.1: Results obtained with DINs on the Kidney Disease dataset; averages over 1000 runs; $\beta = 5$, the specified value of $N_{out}^{(i)}$ is valid for $i = 0, 1, 2, 3$, the last layer has $N_{out}^{(4)} = N_{classes} = 2$.

- some form of overfitting is present when $N_{out}^{(i)} = 3$ and better results are obtained when $N_{out}^{(i)} = 2$, $i = 1, \dots, 3$
- accuracy, specificity and sensitivity are high in all the considered cases

- when $N_{out}^{(i)} = 2$, the accuracy obtained with $N_{train} = 200$ is similar to that obtained with $N_{test} = 320$, which shows that 200 patients are enough to train the DIN
- results obtained with $N_{train} = 320$ and $N_{test} = 80$ can be comparable with those obtained in [75] using other algorithms (k-nearest neighbour, random forest, neural networks)

Notice that the DINs were run one thousand times, each time randomly changing the subsets of patients used for the training and the testing phases; table 10.1 shows the average results.

10.5 Conclusions

The proposed Deep Information Network (DIN) shows good results in terms of accuracy and represents a new alternative to decision trees. We stress that, as far as a hardware implementation is concerned, the DIN has the following good properties:

- Simple modular structure. Only two types of elements are necessary to build a full network (infonode, combiner).
- Flexible. By changing the number of information nodes and their connection topology a wide range of possibilities are available.
- Computations are performed locally. Not only each infonode does not care of computations that are performed at other nodes, but the flow of data is single directional (forward) only.

Further optimization might be obtained by using a different value of N_{out} for each of the information nodes (not necessarily all equal at each layer) and appropriately selecting the value of the Lagrangian multiplier β . In the next Chapter 11 we will explore more carefully the role of β and its impact on the performance of the classification method.

Chapter 11

Probabilistic Deep Information Networks

In this Chapter, we modify and extend the architecture proposed in Chapter 10, [25], by changing the structure of the information node and extending the simple DIN to the case where multiple independent DINs are trained and combined. The content of this chapter is an editorial adaptation of the content of [26].

Decision trees are particularly interesting because they can be easily interpreted. Various types of tree classifiers can be discriminated according to the metric for the iterative construction and selection of features [74]: popular tree classifiers are based on information theoretic metrics, such as ID3, C4.5 [70, 71]. However, it is known that the greedy splitting procedure at each node can be sub-optimal [65], and that decision trees are prone to overfitting when dealing with small datasets. When a classifier is not strong enough, there are, roughly speaking, two possibilities: choosing a more sophisticated classifier or ensembling multiple "weak" classifiers [10, 11]. This second approach is usually called *ensemble* method. In the performance tradeoff by using multiple classifiers simultaneously we improve classification performance, paying with the loss of interpretability.

The so-called "information bottleneck", described in [92], [91], was proposed in Chapter 10 ([25]) to build a classifier (Deep Information Network, DIN) with a tree topology that compresses the input data and generates the estimated class. DINs, Chapter 10 ([25]), are based on the so-called information node that, using the input samples of a feature X_{in} , generates samples of a new feature X_{out} , according to the conditional probabilities $P(X_{out} = j | X_{in} = i)$ obtained by minimising the mutual information $\mathbb{I}(X_{in}; X_{out})$, with the constraint of a given mutual information $\mathbb{I}(X_{out}; Y)$ between X_{out} and the target/class Y (information bottleneck [92]). The outputs of two or more nodes are combined, without information loss, to generate samples of a new feature passed to a subsequent information node. The final node (root) directly outputs the class of each input datapoint. The tree structure of the

network is thus built from the leaves, whereas C4.5 and ID3 build it from the root.

We here propose an improved implementation of the DIN scheme in Chapter 10 ([25]) that only requires the propagation through the tree of small matrices containing conditional probabilities. Notice that the previous version of the DIN was stochastic, while the one we propose here is deterministic. Moreover we use an ensemble (like [10, 11]) of trees with randomly permuted features and weigh their outputs to improve classification accuracy.

The proposed architecture has several advantages in terms of:

- Extreme flexibility and high modularity: all the nodes are functionally equivalent and with a reduced number of inputs and outputs, which gives good opportunities for a possible hardware implementation.
- High parallelizability: each tree can be trained in parallel with the others.
- Memory usage: we need to feed the network with data only at the first layer and simple incremental counters can be used to estimate the initial probability mass distribution.
- Training time and training complexity: the locality of the computed cost function allows a nodewise training that does not require any kind of information from other points of the tree apart from its feeding nodes (that are usually a very small number, e.g. 2-3).

With respect to the DINs in Chapter 10 ([25]), the main difference is that samples of the random variables in the inner layers of the tree are never generated, which is an advantage in the case of large datasets. However an assumption of statistical independence (see Sect. 11.1.3) is necessary to build the probability matrices and this might be seen as a limitation of the newly proposed method. However, experimental results (see Sect. 11.4) show that this approximation does not compromise the performance.

We underline similarities and differences of the proposed classifier with respect to the methods described in [70, 71] since they are among the best performing ones. When using decision trees, as well as DINs, categorical and missing data are easily managed, but continuous random variables are not: quantization of these input features is necessary in a pre-processing phase, and it can be performed as in C4.5 [70], using other heuristics, or manually. Concerning differences, instead, the first one is that normally a hierarchical decision tree is built starting from the root and splitting at each node, whereas we here propose a way to build a tree starting from the leaves. The topology of our network implies that, once the initial ordering of the features has been set, there is no need, after each node is trained, to perform a search of the best possible next node. The second important difference is that we do not use directly mutual information as a metric for building the tree but we base our algorithm on the Information Bottleneck principle [92, 91, 82, 86, 87, 14, 31].

This allows us to extract all the relevant information (the *sufficient statistic*) while removing the redundant one, which is helpful in avoiding overfitting. As in [10, 11] we use an ensemble method. We choose the simplest possible form of ensemble combination: we train independently many structurally equivalent networks, using the same single dataset but permuting the order of the features, and produce a weighted average of the outputs based on a simple rule described in section 11.2.1. Notice that we use a one shot procedure, i.e. we do not iterate more than once over the entire dataset and exploit techniques similarly to [27, 17]. We leave the study of more sophisticated techniques to future works.

Sections 11.1 and 11.2 more precisely describe the structure of the DIN and how it works, section 11.3 gives some insight on the theoretical properties and section 11.4 comments the results obtained with standard datasets. Conclusions are finally drawn in section 11.5.

11.1 The DIN architecture and its training

The information network is made of input nodes (section 11.1.1), information nodes (section 11.1.2) and combiners joined together through a tree network described in section 11.1.3. Moreover, an ensemble of N_{mach} trees is built, based on which the final estimated class is produced (section 11.2.1). In Chapter 10 ([25]), the input nodes are not present, the information node has a slightly different role, the combiners are much simpler than those described here, and just one tree was considered. As already stated, the new version of the DIN is more efficient when a large dataset with relatively few features is analysed.

In the following, it is assumed that all the features take a finite number of discrete values; a case of continuous random variables is discussed in section 11.4.2.

It is also assumed that N_{train} points are used in the training phase, N_{test} points in the testing phase, and that D features are present. The n -th training point corresponds to one of N_{class} possible classes.

11.1.1 The input node

Each input node (see Fig. 11.1) has two input vectors:

1. \mathbf{x}_{in} of size N_{train} , whose elements take values in a set of cardinality N_{in} ; \mathbf{x}_{in} corresponds to one of the D features of the dataset (typically one column).
2. \mathbf{y} of size N_{train} , whose elements take values in a set of cardinality N_{class} ; \mathbf{y} corresponds to the known classes of the N_{train} points.

The notation we use in the equations below is the following: Y, X_{in} represent random variables, $\mathbf{y}(n)$ and $\mathbf{x}_{in}(n)$ are the n -th elements of vectors \mathbf{y} and \mathbf{x}_{in} ,

respectively, and $\mathbf{1}(c)$ is equal to 1 if c is true otherwise it is equal to 0. Using Laplace smoothing [61], the input node estimates the following probabilities¹:

$$\hat{P}(Y = m) \simeq \frac{1 + \sum_{n=0}^{N_{train}-1} \mathbf{1}(\mathbf{y}(n) = m)}{N_{train} + N_{class}} \quad m = 0, \dots, N_{class} - 1 \quad (11.1)$$

$$\hat{P}(X_{in} = i) \simeq \frac{1 + \sum_{n=0}^{N_{train}-1} \mathbf{1}(\mathbf{x}_{in}(n) = i)}{N_{train} + N_{in}}, \quad i = 0, \dots, N_{in} - 1 \quad (11.2)$$

$$\hat{P}(Y = m, X_{in} = i) \simeq \frac{1 + \sum_{n=0}^{N_{train}-1} \mathbf{1}(\mathbf{y}(n) = m) \mathbf{1}(\mathbf{x}_{in}(n) = i)}{N_{train} + N_{class} N_{in}} \quad (11.3)$$

From basic application of probability rules, $\hat{P}(Y = m|X_{in} = i)$ and $\hat{P}(X_{in} = i|Y = m)$ are then computed. From now on, for simplicity, we will denote all the estimated probabilities \hat{P} simply as P .

All the above probabilities can be organized in matrices defined as follows:

$$\mathbf{P}_Y \in \mathbb{R}^{1 \times N_{class}}, \quad \mathbf{P}_Y(m) = P(Y = m) \quad (11.4)$$

$$\mathbf{P}_{X_{in}} \in \mathbb{R}^{1 \times N_{in}}, \quad \mathbf{P}_{X_{in}}(i) = P(X_{in} = i) \quad (11.5)$$

$$\mathbf{P}_{X_{in}|Y} \in \mathbb{R}^{N_{class} \times N_{in}}, \quad \mathbf{P}_{X_{in}|Y}(m, i) = P(X_{in} = i|Y = m) \quad (11.6)$$

$$\mathbf{P}_{Y|X_{in}} \in \mathbb{R}^{N_{in} \times N_{class}}, \quad \mathbf{P}_{Y|X_{in}}(i, m) = P(Y = m|X_{in} = i) \quad (11.7)$$

Note that vectors \mathbf{x}_{in} and \mathbf{y} are not needed by the subsequent elements in the tree, only the input nodes have access to them.

Notice also that the following equalities hold:

$$\mathbf{P}_{X_{in}} = \mathbf{P}_Y \mathbf{P}_{X_{in}|Y} \quad (11.8)$$

$$\mathbf{P}_Y = \mathbf{P}_{X_{in}} \mathbf{P}_{Y|X_{in}} \quad (11.9)$$

¹The probability mass function of Y in (11.1) is common to all the input nodes: it can be evaluated only by the first one and passed to the others.

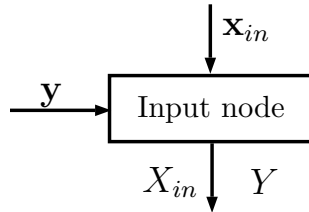


Figure 11.1: Schematic representation of an input node: the inputs are two vectors, the outputs are matrices that statistically describe the random variables X_{in} and Y .

11.1.2 The information node

The information node is schematically shown in Fig. 11.2: the input discrete random variable X_{in} is stochastically mapped into another discrete random variable X_{out} (see Chapter 10 ([25]) for further details) through probability matrices:

- The input probability matrices $\mathbf{P}_{X_{in}}$, $\mathbf{P}_{X_{in}|Y}$, $\mathbf{P}_{Y|X_{in}}$, \mathbf{P}_Y describe the input random variable X_{in} , with N_{in} possible values, and its relationship with class Y .
- The output matrices $\mathbf{P}_{X_{out}}$, $\mathbf{P}_{X_{out}|Y}$, $\mathbf{P}_{Y|X_{out}}$, \mathbf{P}_Y describe the output random variable X_{out} , with N_{out} possible values, and its relationship with Y .

Compression (source encoding) is obtained by setting $N_{out} < N_{in}$.

In the training phase, the information node generates the conditional probability mass function that satisfies equation (see [92])

$$P(X_{out} = j | X_{in} = i) = \frac{1}{Z(i; \beta)} P(X_{out} = j) e^{-\beta d(i,j)}, \quad i = 0, \dots, N_{in}-1, j = 0, \dots, N_{out}-1 \quad (11.10)$$

where

- $P(X_{out} = j)$ is the probability mass function of the output random variable X_{out}

$$P(X_{out} = j) = \sum_{i=0}^{N_{in}-1} P(X_{in} = i) P(X_{out} = j | X_{in} = i), \quad j = 0, \dots, N_{out} - 1 \quad (11.11)$$

- $d(i, j)$ is the Kullback-Leibler divergence

$$\begin{aligned} d(i, j) &= \sum_{m=0}^{N_{class}-1} P(Y = m | X_{in} = i) \log_2 \frac{P(Y = m | X_{in} = i)}{P(Y = m | X_{out} = j)} \\ &= \mathbb{KL}(P(Y | X_{in} = i) || P(Y | X_{out} = j)) \end{aligned} \quad (11.12)$$

and

$$\begin{aligned} P(Y = m | X_{out} = j) &= \sum_{i=0}^{N_{in}-1} P(Y = m | X_{in} = i) P(X_{in} = i | X_{out} = j), \\ & \quad m = 0, \dots, N_{class} - 1, j = 0, \dots, N_{out} - 1 \end{aligned} \quad (11.13)$$

- β is a real positive parameter

- $Z(i; \beta)$ is a normalizing coefficient to get

$$\sum_{j=1}^{N_{out}-1} P(X_{out} = j | X_{in} = i) = 1. \quad (11.14)$$

The probabilities $P(X_{out} = j | X_{in} = i)$ can be iteratively found using the Blahut-Arimoto algorithm [92, 4, 8].

Eqn. (11.10) solves the information bottleneck: it minimises the mutual information $\mathbb{I}(X_{in}; X_{out})$ under the constraint of a given mutual information $\mathbb{I}(Y; X_{out})$. In particular, eqn. (11.10) is the solution of the minimisation of the Lagrangian

$$\mathcal{L} = \mathbb{I}(X_{in}; X_{out}) - \beta \mathbb{I}(Y; X_{out}). \quad (11.15)$$

If the Lagrangian multiplier β is increased, then the constraint is privileged and the information node tends to maximize the mutual information between its output X_{out} and the class Y ; if β is reduced, then minimisation of $\mathbb{I}(X_{in}; X_{out})$ is obtained (compression). The information node must actually balance compression from X_{in} to X_{out} and propagation of the information about Y . In our implementation, the compression is also imposed by the fact that the cardinality of the output alphabet N_{out} is smaller than that of the input alphabet N_{in} .

The role of the information node is thus that of finding the conditional probability matrices

$$\mathbf{P}_{X_{out}|X_{in}} \in \mathbb{R}^{N_{in} \times N_{out}}, \quad \mathbf{P}_{X_{out}|X_{in}}(i, j) = P(X_{out} = j | X_{in} = i) \quad (11.16)$$

$$\mathbf{P}_{Y|X_{out}} \in \mathbb{R}^{N_{out} \times N_{class}}, \quad \mathbf{P}_{Y|X_{out}}(j, m) = P(Y = m | X_{out} = j) \quad (11.17)$$

$$\mathbf{P}_{X_{out}} \in \mathbb{R}^{1 \times N_{out}}, \quad \mathbf{P}_{X_{out}}(j) = P(X_{out} = j). \quad (11.18)$$

11.1.3 The combiner

Consider the case depicted in Fig. 11.3, where the two information nodes a and b feed a combiner (shown as a triangle) that generates the input of the information node c . The random variables $X_{out,a}$ and $X_{out,b}$, both having alphabet with cardinality N_1 , are combined together as

$$X_{in,c} = X_{out,a} + N_1 X_{out,b} \quad (11.19)$$

that has an alphabet with cardinality $N_1 \times N_1$.

The combiner actually does not generate $X_{in,c}$, it simply evaluates the probability matrices that describe $X_{in,c}$ and Y . In particular, the information node c needs

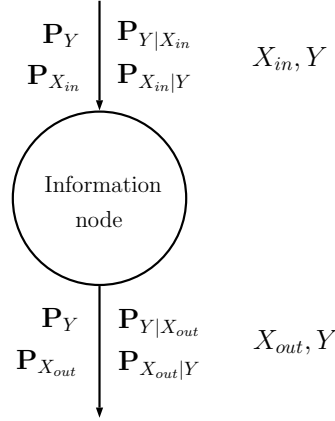


Figure 11.2: Schematic representation of an information node, showing the input and output matrices.

$\mathbf{P}_{X_{in,c}|Y}$, which can be evaluated **assuming** that $X_{out,a}$ and $X_{out,b}$ are conditionally independent given Y ²:

$$\begin{aligned} P(X_{in,c} = k|Y = m) &= P(X_{out,a} = k_a, X_{out,b} = k_b|Y = m) \\ &= P(X_{out,a} = k_a|Y = m)P(X_{out,b} = k_b|Y = m) \end{aligned} \quad (11.20)$$

where $k = k_a + N_1 k_b$. In particular, the m -th row of $\mathbf{P}_{X_{in,c}|Y}$ is the Kronecker product of the m -th rows of $\mathbf{P}_{X_{out,a}|Y}$ and $\mathbf{P}_{X_{out,b}|Y}$

$$\mathbf{P}_{X_{in,c}|Y}(m, :) = \mathbf{P}_{X_{out,a}|Y}(m, :) \otimes \mathbf{P}_{X_{out,b}|Y}(m, :), \quad m = 0, \dots, N_{class} - 1 \quad (11.21)$$

(here $\mathbf{A}(m, :)$ identifies the m -th row of matrix \mathbf{A}). The probability vector $\mathbf{P}_{X_{in,c}}$ can be evaluated considering that

$$P(X_{in,c} = k) = \sum_{m=0}^{N_{class}-1} P(X_{in,c} = k, Y = m) = \sum_{m=0}^{N_{class}-1} P(X_{in,c} = k|Y = m)P(Y = m) \quad (11.22)$$

so that

$$\mathbf{P}_{X_{in,c}} = \mathbf{P}_Y \mathbf{P}_{X_{in,c}|Y} \quad (11.23)$$

At this point, matrix $\mathbf{P}_{Y|X_{in,c}}$ can be evaluated element by element since

$$\begin{aligned} P(Y = m|X_{in,c} = k) &= \frac{P(X_{in,c} = k|Y = m)P(Y = m)}{P(X_{in,c} = k)}, \\ m &= 1, \dots, N_{class} - 1, k = 0, \dots, N_1 \times N_1 - 1 \end{aligned} \quad (11.24)$$

It is straightforward to extend the equations to the case in which $X_{in,a}$ and $X_{in,b}$ have different cardinalities.

²Notice that in implementation Chapter 10 ([25]) this assumption was not necessary.

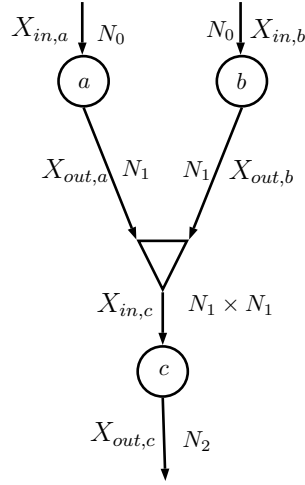


Figure 11.3: Sub-network: $X_{in,a}$, $X_{out,a}$, $X_{in,b}$, $X_{out,b}$, $X_{in,c}$, $X_{out,c}$ are all random variables; N_0 is the number of values taken by $X_{in,a}$ and $X_{in,b}$, N_1 is the number of values taken by $X_{out,a}$ and $X_{out,b}$, N_2 is the number of values taken by $X_{out,c}$.

11.1.4 The tree architecture

Fig. 11.4 shows an example of a probabilistic DIN, where we assume that the dataset has $D = 8$ features and that training is thus obtained using a matrix \mathbf{X}_{train} with N_{train} rows and $D = 8$ columns, with a corresponding class vector \mathbf{y} . The k -th column $\mathbf{x}(k)$ of matrix \mathbf{X}_{train} feeds, together with vector \mathbf{y} , the input node $I(k)$, $k = 0, \dots, D - 1$.

Information node $(k,0)$ at layer 0 processes the probability matrices generated by the input node $I(k)$, with $N_{in}^{(0)}$ possible values of $X_{in}(k,0)$, and evaluates the conditional probability matrices with $N_{out}^{(0)}$ possible values of $X_{out}(k,0)$, using the algorithm described in Section 11.1.2. The outputs of info nodes $(2k,0)$ and $(2k+1,0)$ are given to a combiner that outputs the probability matrices for $X_{in}(k,1)$, having alphabet of cardinality $N_{in}^{(1)} = N_{out}^{(0)} \times N_{out}^{(0)}$, using the equations described in Section 11.1.3. The sequence of combiners and information nodes is iterated, decreasing the number of information nodes from layer to layer, until the final root node is obtained. In the previous implementation of the DINs in Chapter 10 ([25]), the root information node outputs the estimated class of the input and it is therefore necessary that the output cardinality of the root info node is equal to N_{class} . In the current implementation, this cardinality can be larger than N_{class} , since classification is based on the output probability matrix $\mathbf{P}_{Y|X_{out}}$.

For a number of features $D = 2^d$, the number of layers is d . If D is not a power of 2, then it is possible to use combiners with 3 or more inputs (the changes in the equations in Sect. 11.1.3 are straightforward, since a combiner with three inputs

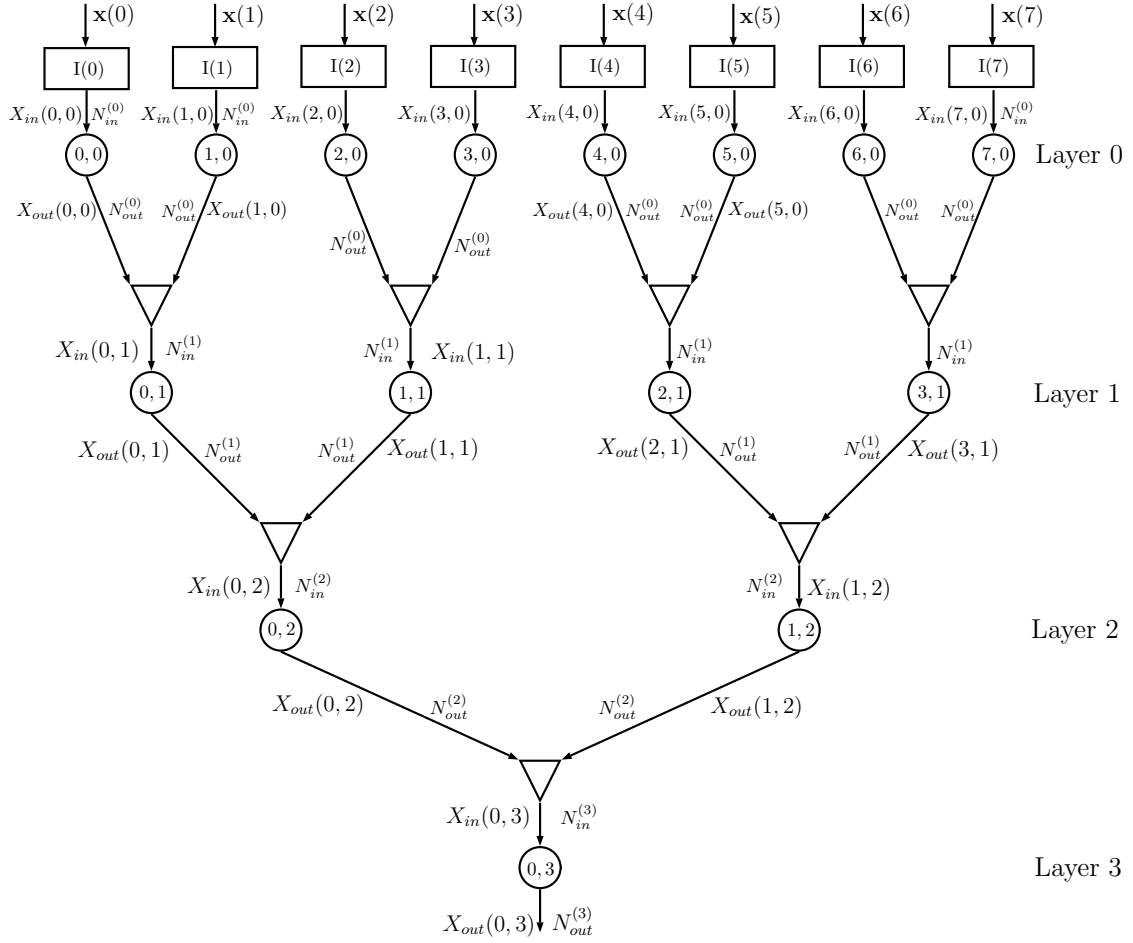


Figure 11.4: Example of a DIN for $D = 8$: the input nodes are represented as rectangles, the info nodes as circles, the combiners as triangles. The numbers inside each circle identify the node (position inside the layer and layer number), $N_{in}^{(k)}$ is the number of values taken by the input of the info node at layer k , $N_{out}^{(k)}$ is the number of values taken by the output of the info node at layer k . In this example the info nodes at a given layer all have the same input and output cardinalities.

can be seen as two cascaded combiners with two inputs each).

In the overall, the binary topology proposed in Fig. 11.4 requires a number of information nodes equal to

$$N_{nodes} = D + \frac{D}{2} + \frac{D}{4} + \cdots + 2 + 1 = 2D - 1 \quad (11.25)$$

and a number of combiners equal to

$$N_{comb} = \frac{D}{2} + \frac{D}{4} + \cdots + 2 + 1 = D - 1 \quad (11.26)$$

All the info nodes run exactly the same algorithm and all the combiners are equal, apart from the input/output alphabet cardinalities. If the cardinalities of the alphabets are all equal, i.e. $N_{in}^{(i)}$ and $N_{out}^{(i)}$ do not depend on the layer i , then all the nodes and all the combiners are exactly equal, which might help in a possible hardware implementation; in this case the number of parameters of the network is $(N_{out} - 1) \times N_{in} \times N_{nodes}$.

Actually the network performance depends on how the features are coupled in subsequent layers and a random shuffling of the columns of matrix \mathbf{X}_{train} provides results that might be significantly different. This property will be used in Section 11.2.1 for building the ensemble of networks.

11.1.5 A note on computational complexity and memory requirements

The modular structure of the proposed method has several advantages both in terms of memory footprint and computational cost. The considered topology in this explanation is binary, similarly to what depicted in Fig. 11.4. We will furthermore consider for simplicity cardinalities of the D input features all equal to N_{in} and input/output cardinalities of subsequent layers information node to be also fixed constants N_{in}^* and $N_{out}^* = \frac{N_{in}^*}{2}$ respectively. As we will show in the experimental section 11.4 small values for N_{in}^* and N_{out}^* such as 2,3 or 4 are sufficient in the considered cases. Straightforward generalizations are possible when considering dishomogeneous cases.

At the first layer (the input node layer) each of the D input nodes stores the joint probabilities of the target variable Y and its input feature. Each node thus includes a simple counter that fills the probability matrix of dimension $N_{in} \times N_{class}$. Both the computational cost and the memory requirements for this first stage are the same as the Naive Bayes algorithm. Notice that from memory requirements point of view, is not necessary to store all the training data but just counters with number of joint occurrences of features/ classes. If after training new datapoints are observed it is in fact sufficient to update the counters and properly renormalize the values to obtain the updated probability matrices. In this thesis we do not cover the topic of online learning as well as possible strategies to reduce the computational complexity in such a scenario.

At the second layer (the first information node layer) each node receives as input the joint probability matrix of feature and target variable and performs the Blahut-Arimoto algorithm. The internal memory requirement of this node is the space needed to store two probability matrices of dimensions $N_{in}^* \times N_{class}$ and $N_{in}^* \times N_{out}^*$ respectively. The cost per iteration of Blahut-Arimoto depends on matrices multiplication of sizes $N_{in}^* \times N_{out}^*$ matrix and $N_{in}^* \times N_{class}$, and thus obviously the complexity scales with the number of classes of the considered classification

problem. To the best of our knowledge, the convergence rate for the Blahut-Arimoto algorithm applied to Information Bottleneck problems is unknown. In this work we found however empirically that for the considered datasets 5-6 iterations per node are sufficient, as will be discussed in Section 11.4.5.

Each combiner process the matrices generated by two information nodes: the memory requirement is zero and the computational cost is roughly N_{class} Kronecker products between rows of probability matrices. Since for ease of explanation we chosed $N_{out}^* = \frac{N_{in}^*}{2}$ the output probability matrix have again dimensions $N_{in}^* \times N_{class}$.

The overall memory requirement and computational complexity (for a single DIN) is thus going to scale as D times the requirements for an input node, $2D - 1$ times the requirements for an information node and $D - 1$ times the requirements for a combiner. To complete the discussion we have to remember that a further multiplication factor of N_{mach} is required to take into account that we are considering an ensemble of networks³.

11.2 The running phase

During the running phase, the columns of matrix \mathbf{X} with N rows and D columns are used as inputs. Assume again that the network architecture is that depicted in Fig. 11.4 with $D = 8$, and consider the n -th input row $\mathbf{X}(n, :)$.

In particular, assume that $\mathbf{X}(n, 2k) = i$ and $\mathbf{X}(n, 2k + 1) = j$. Then

1. (a) input node $I(2k)$ passes value i to info node $(2k, 0)$;
 (b) input node $I(2k + 1)$ passes value j to info node $(2k + 1, 0)$;
2. (a) info node $(2k, 0)$ passes the probability vector $\mathbf{p}_a = \mathbf{P}_{X_{out}(2k,0)|X_{in}(2k,0)}(i, :)$ (i -th row) to the combiner; \mathbf{p}_a stores the conditional probabilities $P(X_{out}(2k, 0) = g | \mathbf{X}(n, 2k) = i)$ for $g = 0, \dots, N_{out}^{(0)} - 1$;
 (b) info node $(2k + 1, 0)$ passes the probability vector $\mathbf{p}_b = \mathbf{P}_{X_{out}(2k+1,0)|X_{in}(2k+1,0)}(j, :)$ (j -th row) to the combiner; \mathbf{p}_b stores the conditional probabilities $P(X_{out}(2k + 1, 0) = h | \mathbf{X}(n, 2k + 1) = j)$ for $h = 0, \dots, N_{out}^{(0)} - 1$;
3. the combiner generates vector

$$\mathbf{p}_c = \mathbf{p}_a \otimes \mathbf{p}_b, \quad (11.27)$$

³Actually the first layer, the set of input nodes, can be shared by the different architectures since only the relative position of the input nodes changes, see Sect. 11.2.1

which stores the conditional probabilities

$$P(X_{in}(k,1) = s | \mathbf{X}(n,2k) = i, \mathbf{X}(n,2k+1) = j) \text{ for } s = 0, \dots, N_{in}^{(1)} - 1, \text{ where } N_{in}^{(1)} = N_{out}^{(0)} \times N_{out}^{(0)},$$

4. info node $(k,1)$ generates the probability vector

$$\mathbf{p}_c \mathbf{P}_{X_{out}(k,1)|X_{in}(k,1)}, \quad (11.28)$$

which stores the conditional probabilities

$$P(X_{out}(k,1) = r | \mathbf{X}(n,2k) = i, \mathbf{X}(n,2k+1) = j) \text{ for } r = 0, \dots, N_{out}^{(1)}$$

5. in the following layer each combiner performs the Kronecker product of its two input vectors and each info node performs the product between the input vector and its conditional probability matrix $\mathbf{P}_{X_{out}|X_{in}}$;
6. the root information node at layer 3, having the input vector \mathbf{p} , outputs

$$\mathbf{p}_{out}(n) = \mathbf{p} \mathbf{P}_{X_{out}(0,3)|X_{in}(0,3)} \mathbf{P}_{Y|X_{out}(0,3)}, \quad (11.29)$$

which stores the estimated probabilities

$$P(Y = m | \mathbf{X}(n, :)) \text{ for } m = 0, \dots, N_{class} - 1.$$

According to the MAP criterion, the estimated class of the input point $\mathbf{X}(n, :)$ is

$$\hat{Y}(n) = \arg \max \mathbf{p}_{out}(n) \quad (11.30)$$

but we propose to use an improved method, described in Section 11.2.1.

11.2.1 The DIN ensemble

At the end of the training phase, when all the conditional matrices have been generated in each information node and combiner, the network is run with input matrix \mathbf{X}_{train} (N_{train} rows and D columns) and the probability vector \mathbf{p}_{out} is obtained for each input point $\mathbf{X}_{train}(n, :)$. As anticipated at the end of Section 11.1.4, the DIN classification accuracy depends on how the input features are combined together. By permuting the columns of \mathbf{X}_{train} , a different probability vector \mathbf{p}_{out} is typically obtained. We thus propose to generate an ensemble of DINs by randomly permuting the columns of \mathbf{X}_{train} , and then combine their outputs.

Since in the training phase $\mathbf{y}(n)$ is known, it is possible to get for each DIN v the probability $\mathbf{p}_{out}^v(n)$, and ideally $\mathbf{p}_{out}^v(n, \mathbf{y}(n))$, the estimated probability corresponding to the true class $\mathbf{y}(n)$, should be equal to one. The weights

$$w^v = \frac{\sum_{n=0}^{N_{train}-1} \mathbf{p}_{out}^v(n, \mathbf{y}(n))}{\sum_{n=0}^{N_{train}-1} \sum_{j=0}^{N_{mach}-1} \mathbf{p}_{out}^j(n, \mathbf{y}(n))} \quad (11.31)$$

thus represent the reliability of the v -th DIN.

In the running phase, feeding the N_{mach} machines each with the correctly permuted vector $\mathbf{X}(n, \cdot)$, the final estimated probability vector is determined as

$$\hat{\mathbf{P}}_{ens}(n) = \sum_{v=0}^{N_{mach}-1} w^v \hat{\mathbf{P}}_{out}^v(n) \quad (11.32)$$

and the estimated class is

$$\hat{Y}(n) = \arg \max \hat{\mathbf{P}}_{ens}(n). \quad (11.33)$$

11.3 The probabilistic point of view

This section is intended to underline the difference in probability terms formulation between the Naive Bayes classifier [61, 36] and the proposed scheme, since both use the assumption of conditional independence of the input features. Both these classifiers build in a simplified way the probability matrix $\mathbf{P}_{Y|X_0, \dots, X_D}$ with N_{class} rows and $\prod_{i=0}^{D-1} N_{in}^{(i)}$, where $N_{in}^{(i)}$ is the cardinality for the input feature X_i . In the next sections we show the different structure of these two probability matrices.

11.3.1 Assumption of conditionally independent features

The Naive Bayes assumption allows to write the output estimated probability of the naive bayes classifier as follows:

$$\begin{aligned} P(Y = m | \mathbf{x} = \mathbf{x}_0) &= \frac{P(\mathbf{x} = \mathbf{x}_0 | Y = m) P(Y = m)}{P(\mathbf{x} = \mathbf{x}_0)} \\ &= \frac{\left[\prod_{k=0}^{D-1} P(X_k = x_{k0} | Y = m) \right] P(Y = m)}{\sum_{s=0}^{N_{class}} \left[\prod_{k=0}^{D-1} P(X_k = x_{k0} | Y = s) \right] P(Y = s)} \end{aligned} \quad (11.34)$$

which is very easily implemented, without the need of generating the tree network. We rewrite this output probability in a fairly complex way to show the difference between the naive Bayes probability matrix and the DIN one. Consider the n th feature $x(n)$, that can take values in the set $\{c_n^0, \dots, c_n^{D_n-1}\}$. Define $\mathbf{P}_{x(n)|y=m} = [P(x(n) = c_n^0 | Y = m), \dots, P(x(n) = c_n^{D_n-1} | Y = m)]$, then

$$\mathbf{P}_{X_{in}|Y}(m, \cdot) = \otimes_{k=0}^{D-1} \mathbf{P}_{x(k)|y=m} \quad (11.35)$$

and thus obviously

$$\mathbf{P}_{X_{in}|Y} = \begin{bmatrix} \otimes_{k=0}^{D-1} \mathbf{P}_{x(k)|y=0} \\ \otimes_{k=0}^{D-1} \mathbf{P}_{x(k)|y=1} \\ \vdots \\ \otimes_{k=0}^{D-1} \mathbf{P}_{x(k)|y=N_{class}} \end{bmatrix} \quad (11.36)$$

We can write the joint probability matrix as

$$\mathbf{P}_{X_{in},Y} = \text{diag}(\mathbf{P}_Y)\mathbf{P}_{X|Y} \quad (11.37)$$

and the probability matrix of target class given observation as

$$\mathbf{P}_{Y|X_{in}} = (\mathbf{P}_{X_{in},Y}\text{diag}(\mathbf{P}_{X_{in}}^{\circ(-1)}))^T \quad (11.38)$$

The hypothesis of conditional statistical independence of the features is not always correct and thus we can incurr obviously into performance degratation.

11.3.2 The overall probability matrix

We now instead compute the output estimated probability for the DIN classifier. Consider again the sub-network in Fig. 11.3 made of info nodes a, b, c . Info node a is characterized by matrix \mathbf{P}_a , whose element $P_a(i, j)$ is $P(X_{out,a} = j|X_{in,a} = i)$; similar definitions hold for \mathbf{P}_b and \mathbf{P}_c . Note that \mathbf{P}_a and \mathbf{P}_b have N_0 rows and N_1 columns, whereas \mathbf{P}_c has $N_1 \times N_1$ rows and N_2 columns; the overall probability matrix between the inputs $X_{in,a}, X_{in,b}$ and the output $X_{out,c}$ is $\tilde{\mathbf{P}}$ with $N_0 \times N_0$ rows and N_2 columns. Then

$$\begin{aligned} & P(X_{out,c} = i|X_{in,a} = j, X_{in,b} = k) \\ &= \sum_{r=0}^{N_1-1} \sum_{s=0}^{N_1-1} P(X_{out,c} = i, X_{out,a} = r, X_{out,b} = s|X_{in,a} = j, X_{in,b} = k) \\ &= \sum_{r=0}^{N_1-1} \sum_{s=0}^{N_1-1} P(X_{out,c} = i|X_{out,a} = r, X_{out,b} = s)P(X_{out,a} = r|X_{in,a} = j)P(X_{out,b} = s|X_{in,b} = k) \\ &= \sum_{r=0}^{N_1-1} \sum_{s=0}^{N_1-1} P(X_{out,c} = i|X_{out,s} = r, X_{out,b} = s)\mathbf{P}_a(j, r)\mathbf{P}_b(k, s). \end{aligned} \quad (11.39)$$

It can be shown that

$$\tilde{\mathbf{P}} = (\mathbf{P}_a \otimes \mathbf{P}_b)\mathbf{P}_c \quad (11.40)$$

where \otimes identifies the Kronecker matrix multiplication; note that $\mathbf{P}_a \otimes \mathbf{P}_b$ has $N_0 \times N_0$ rows and $N_1 \times N_1$ columns. By iteratively applying the above rule, we can get the expression of the overall matrix $\tilde{\mathbf{P}}$ for the exact topology of Fig. 11.4, with 8 input nodes and four layers:

$$\begin{aligned} \tilde{\mathbf{P}} &= \left[\left\{ \left[(\mathbf{P}_{0,0} \otimes \mathbf{P}_{1,0})\mathbf{P}_{0,1} \right] \otimes \left[(\mathbf{P}_{2,0} \otimes \mathbf{P}_{3,0})\mathbf{P}_{1,1} \right] \right\} \mathbf{P}_{0,2} \right. \\ &\quad \left. \otimes \left\{ \left[(\mathbf{P}_{4,0} \otimes \mathbf{P}_{5,0})\mathbf{P}_{2,1} \right] \otimes \left[(\mathbf{P}_{6,0} \otimes \mathbf{P}_{7,0})\mathbf{P}_{3,1} \right] \right\} \mathbf{P}_{1,2} \right] \mathbf{P}_{0,3}. \end{aligned} \quad (11.41)$$

The overall output probability matrix $\mathbf{P}_{Y|X}$ can be finally computed as

$$\mathbf{P}_{Y|X_{in}} = \tilde{\mathbf{P}}\mathbf{P}_{Y|X_{out}(0,3)}. \quad (11.42)$$

The DIN then behaves like a one-layer system that generates the output according to matrix $\mathbf{P}_{Y|X_{in}}$, whose size might be impractically large. It is also possible to interpret the system as a sophisticated way of factorizing and approximating the exponentially large true probability matrix. In fact, the proposed layered structure needs smaller probability matrices, which makes the system computationally efficient. The equivalent probability matrix is thus different in the DIN (11.42) and Naive Bayes (11.38) case.

11.4 Experiments

In this section we analyse the results obtained with benchmark datasets. In particular we consider the DIN ensemble when a) each DIN is based on the probability matrices (the scheme described in this Chapter) and b) each information node of the DIN randomly generates the symbols, as described in the previous work Chapter 10 ([25]). We refer to these two variants in captions and labels as DIN(Prob) and DIN(Gen) respectively. The reason for this comparison is that conditional statistical independence is not required in the case DIN(Gen), and the classification accuracy could be different in the two cases. Note that Chapter 10 ([25]) considered just one DIN, not an ensemble of DINs. In the following we introduce three datasets on which we tested the method (subsections 11.4.1, 11.4.2, 11.4.3 respectively) and propose some examples of DINs architectures. Complete analysis of numerical results is described in Section 11.4.4. Sections 11.4.5, 11.4.6, analyse the impact of changing the maximum number of iterations of Blahut-Arimoto algorithm and Lagrangian coefficient β respectively. Finally a synthetic multiclass experiment is described in section 11.4.7.

11.4.1 UCI Congressional Voting Records dataset

The first experiment on real data was conducted on the UCI Congressional Voting Records dataset [93], that collects the votes given by each of the U.S. House of Representatives Congressmen on 16 key laws (year 1985). Each vote can take three values corresponding to (roughly, see [93] for more details) yes, no and missing value; each datapoint belongs to one of two classes (democrats or republican). The aim of the network is, given the list of 16 votes, decide if the voter is republican or democratic. In this dataset we thus have $D = 16$ features and a total of 435 datapoints splitted in N_{train} datapoints for the training and $N_{test} = 435 - N_{train}$ points for the testing. The architecture of the used network is the same as the one described in Sect. 11.1.4, except for the fact that there are 16 input features instead of 8 (the network has thus one more layer). The input cardinality in the first layer is $N_{in}^{(0)} = 3$ (yes/no/missing) and the output cardinality is set to $N_{out}^{(0)} = 2$. From the second layer on, the input cardinality for each information node is equal to $N_{in} = 4$

and $N_{out} = 2$. In the majority of the cases, the size of the probability matrices is therefore 4×2 or 2×2 . In this example we used $N_{mach} = 30$ and $N_{train} = 218$ (roughly 50% of the datapoints). The value of β has been set to 2.2.

11.4.2 UCI Kidney Disease dataset

The second considered dataset was the UCI Kidney Disease dataset ([75]). The dataset has a total of 24 medical features, consisting of mixed categorical, integer and real values, with missing values. Quantization of non-categorical features of the dataset was performed according to the thresholds in Section 11.4.2, agreed with a medical doctor.

The aim of the experiment is to correctly classify patients affected by chronic kidney disease. We performed 100 different trials training the algorithms using only $N_{train} = 50$ out of 400 samples for the training. Layer zero has 24 input nodes, then the outputs of layer zero are mixed two at a time to get 12 information nodes at layer 1, 6 at the layer 2, 3 at layer 3; the last 3 nodes are combined into a unique final node. The output cardinality of all nodes is equal to 2 (including the input nodes), the value of β was set equal to 5.6. Also in this case we used an ensemble of $N_{mach} = 30$ DINs.

Quantization

Hereafter we present the quantization⁴ scheme used for the numerical features of chronic kidney disease dataset.

- Age (Years) $\{< 10, < 18, < 45, < 70, < 120\}$
- Blood (mm/Hg) $\{< 80, < 84, < 89, < 99, < 109, \geq 110\}$
- Blood Glucose Random (mgs/dl) $\{< 79, < 160, < 200, \geq 200\}$
- Blood Urea (mgs/dl) $\{< 6, < 20, \geq 20\}$
- Serum Creatinine (mgs/dl) $\{< 0.5, < 1.2, < 2, \geq 2\}$
- Sodium (mEq/L) $\{< 136, < 145, \geq 145\}$
- Potassium (mEq/L) $\{< 3.5, < 5, \geq 5\}$
- Hemoglobine (gms) $\{< 12, < 17, \geq 17\}$
- Packed Cell Volume $\{< 27, < 52, \geq 52\}$

⁴A special thank to Prof. MD Gabriella Olmo who suggested a quantization of the continuous values of the features in the experiment which is correct from a medical point of view.

- White Blood Cell Count (cells/cumm) $\{< 3500, < 10500, \geq 10500\}$
- Red Blood Cell (millions/cmm) $\{< 2.5, < 6, \geq 6\}$

11.4.3 UCI Mushroom dataset

The last considered dataset was the UCI Mushroom dataset [21]. This dataset is made of 22 categorical features with different cardinalities, that describe some properties of mushrooms, and one target variable that defines whether the considered mushroom is edible or poisonous/unsafe. There are a total of 8124 entries in the dataset. We pad the dataset with two null features to reach the cardinality of 24 and use exactly the same architecture as the kidney disease experiment. We select $N_{train} = 50, \beta = 2.7$ and number of DINs equal to $N_{mach} = 15$.

11.4.4 Misclassification probability analysis

We hereafter report results in terms of misclassification probability between the proposed method and several classification methods implemented using MATLAB® Classification Learner. All the datasets were randomly splitted 100 times into training and testing subsets, thus generating 100 different experiments. The proposed method shows competitive results in the considered cases, as can be observed in Table 11.1. It is interesting to compare in terms of performance the proposed algorithm with respect to the Naive Bayes classifier, i.e. eqn. (11.34), and the Bagged Tree algorithm, that is the closest algorithm (conceptually) to the one we propose. In general, the two variants of the DINs performs similarly to the Bagged Trees, while outperform Naive Bayes. For Bagged Trees and KNN-Ensemble the same number of learners as DIN ensembles were used.

Classifier	Congressional Voting Records	Kidney Disease	Mushroom
Naive Bayes	0.10894	0.051	0.20641
Decision Tree	0.050691	0.062314	0.05505
Bagged Trees	0.043641	0.0268	0.038305
DIN Prob	0.050138	0.037229	0.020796
DIN Gen	0.049447	0.026286	0.022182
Linear Discriminant Classifier	0.059724	0.091029	0.069923
Logistic Regression	0.075161	0.096429	0.07074
Linear SVM	0.063226	0.049914	0.04513
KNN	0.08682	0.11369	0.037018
KNN-Ensemble	0.062811	0.036057	0.043967

Table 11.1: Mean misclassification probability (over 100 random experiments) for the three datasets with the considered classifiers.

11.4.5 The impact of number of iterations of Blahut-Arimoto on the performance

As anticipated in Section 11.1.5, the computational complexity of a single node scales with the number of iterations of Blahut-Arimoto algorithm. To the best of our knowledge it does not exist a provable convergence rate for the Blahut-Arimoto algorithm in the Information Bottleneck setting. We hereafter (Figure 11.5) present empirical results on the impact of limiting the number of iterations of Blahut-Arimoto algorithm (for simplicity the same bound is applied to all nodes in the networks). When the number of iterations is too small there is a drastic decrease in performance, that is due to the fact that the probability matrices in the information nodes have not yet converged, while 5-6 iterations are sufficient and a further increase in the number of iterations is not necessary in terms of performance improvements.

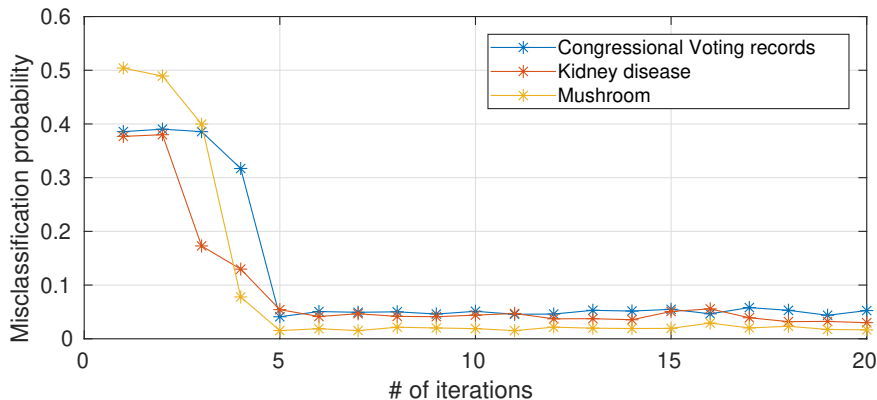


Figure 11.5: Misclassification probability versus number of iterations (average over 10 different trials) for the considered UCI datasets.

11.4.6 The role of β : underfitting, optimality and overfitting

As usual with almost all machine learning algorithms, the choice of hyperparameters is of fundamental importance. For simplicity, in all experiments described in previous sections we kept the value of β constant through the network. To gain some intuition, figure 11.6 shows the misclassification probability for different β for the three considered datasets (each time keeping β constant through the network). While the three curves are quantitatively different we can notice the same qualitative trend: when β is too small not enough information about the target variable is propagated, then by increasing β above a certain threshold the misclassification

probability drops. Increasing β too much however induces overfitting, as expected, and the classification error (slowly) increases again. Remember (from (11.15)) that the Lagrangian we are minimising is

$$\mathcal{L} = \mathbb{I}(X_{in}; X_{out}) - \beta \mathbb{I}(Y; X_{out}).$$

Information theory tells us that at every information node we should propagate only the sufficient statistic about the target variable Y . In practice this is reflected in the role of β : when it is too small we neglect the term $\mathbb{I}(Y; X_{out})$ and just minimise $\mathbb{I}(X_{in}; X_{out})$ (that corresponds to underfitting), while increasing β allows to pass more information about the target variable through the bottleneck. It is important to remember however that we do not have direct access to the true mutual information values but just to an empirical estimate based on a finite dataset. Especially when the cardinalities of inputs and outputs are high this translates into an increased probability of spotting spurious correlations that, if learned by the nodes, induce overfitting. The overall message is that β has an extremely important role in the proposed method, and its value should be chosen to modulate between underfitting and overfitting.

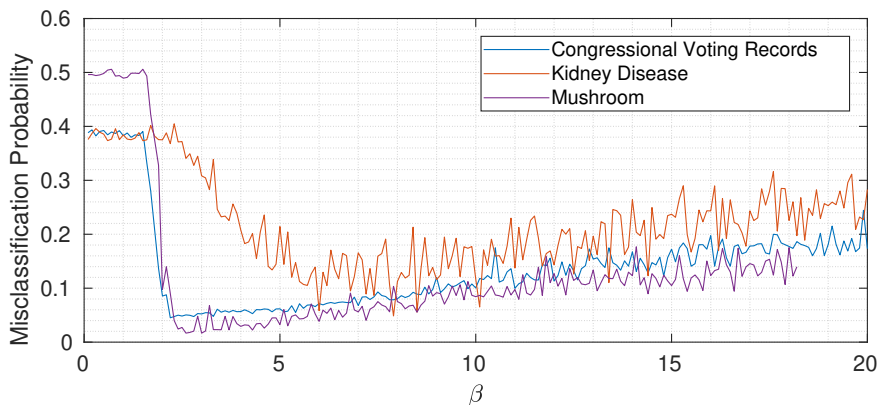


Figure 11.6: Misclassification probability versus β (average over 20 different trials) for the considered UCI datasets.

11.4.7 A synthetic multiclass experiment

In this section we present results on a multiclass synthetic dataset. We generated 64 dimensional feature vectors \mathbf{z} drawn from multivariate Gaussian distributions with mean and covariance depending on a target class y and a control parameter

ρ :

$$p(\mathbf{z}|y = l) = |2\pi\Sigma_l|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{z} - \mu_l)^T(\rho\Sigma_l)^{-1}(\mathbf{z} - \mu_l)\right) \quad l = 1, \dots, N_{class} \quad (11.43)$$

where for the considered experiment $N_{class} = 8$. The mean μ_l is sampled from a normal 64 dimensional random vector and Σ_l is randomly generated as $\Sigma_l = \mathbf{A}\mathbf{A}^T$ (where \mathbf{A} is sampled from a matrix normal distribution) and normalized to have unit norm. The other parameter ρ is inserted to modulate the signal to noise ratio of the generated samples: a smaller value of ρ corresponds to smaller feature variances and more distinct, less overlapping, pdfs $p(\mathbf{z}|y = l)$, and an easier classification task. We then perform quantization of the result using 1 bit, i.e. the input of the ensemble of DINs is the following random vector:

$$\mathbf{x} = U(\mathbf{z}) \quad (11.44)$$

where $U(\cdot)$ is the Heaviside step operator. The designed architecture has at the first layer 64 input nodes, then 32,16,4,2,1. The output cardinalities are equal to 2 for the first three layers, equal to 4 for the fourth and fifth layer and equal to 8 at the last layer. We select $N_{train} = 1000$, $\beta = 7$ (constant trough the network) and number of DINs equal to $N_{mach} = 10$. Figure 11.7 shows the classification accuracy (on a test set of 1000 samples) for different values of ρ . As expected when the value of ρ is small we can reach almost perfect classification accuracy, whereas by increasing it the performance drops to the point where the useful signal is completely buried in noise and the classification accuracy reaches the asymptotic level of $\frac{1}{8}$ (that corresponds to random guessing when the number of classes is equal to 8).

11.5 Conclusions

The proposed ensemble Deep Information Network (DIN) shows good results in terms of accuracy and represents a new simple, flexible and modular structure. The required hyperparameters are the cardinality of the alphabet at the output of each information node, the value of the Lagrangian multiplier β , and the structure of the tree itself (number of input information nodes of each combiner).

Simplistic architecture choices made for the experiments (such as equal cardinality of all node outputs, β constant through the network,...) performed comparably to finely tuned networks. We expect however, that, similarly to what happened in Neural Network applications, a domain specific design of the architectures will allow for consistent improvements in terms of performance on complex datasets.

Despite the local assumption of conditionally independent features, the proposed method always outperforms Naive Bayes. As discussed in section 11.3, the induced

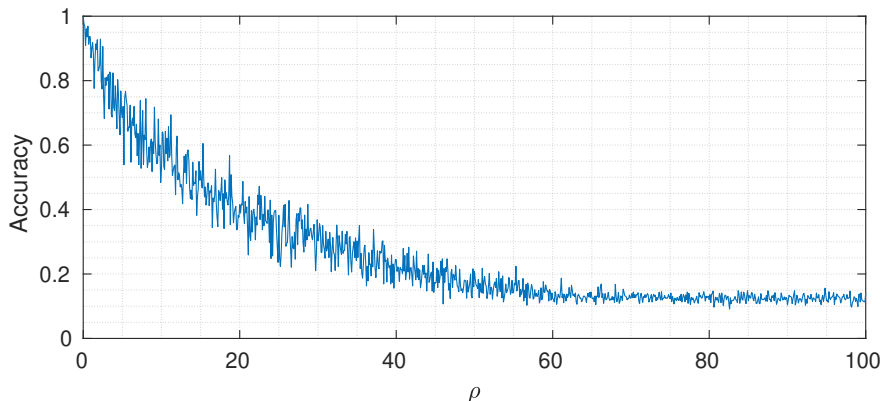


Figure 11.7: Varying of classification accuracy for different values of control parameter ρ

equivalent probability matrix is different in the two cases. Intuitively, we can understand the difference in performance under the point of view of probability matrix factorization. On one side we have the true, exponentially large, joint probability matrix of all features and target class; on the other side we have the Naive Bayes one, that is extremely simple in terms of complexity but obviously less performing; in between we have the proposed method, where the complexity is still reasonable but the quality of the approximation is much better. The DIN(Gen) algorithm does not require the assumption of statistical independence, but the classification accuracy is very close to that of DIN(Prob), which further suggests that the assumption can be accepted from a practical point of view.

The proposed method leaves open the possibility of devising a custom hardware implementation. Differently from classical decision trees, in fact, the execution times of all branches as well as the precise number of operations is fixed per datapoint and known a priori, helping in various system design choices. In fact with classical trees, where nodes utilization depend on the datapoint, we are forced to design the system for the worst case, even if in the vast majority of time not all nodes are used. Instead, with DIN, there is not such a problem.

Finally, a clearly open point is related to the quantization procedure of continuous random variables. One possible self-consistent approach could be devising an information bottleneck based method (similarly to the method for continuous random variables [14]).

Further studies on extremely large dataset will help understand principled ways of tuning hyperparameters and architecture choices and their relationship on performance.

Chapter 12

Conclusions of Part II

We proposed a practical algorithm designed for categorical datasets. In Chapters 10,11 we presented the Deep Information Networks (DIN) and their implementation variant. The algorithm is presented starting from clear and definite information theoretic principles, in particular the information bottleneck criterion. The connection with the concept of compression and minimal statistical sufficiency is explored in the introductory chapters.

The DIN shows good results in terms of classification accuracy. The experimental validation suggests that the DINs are a valid alternative to the classical decision trees. We moreover depart from the original DIN idea by exploring ensemble of networks and a different implementation based on a local assumption of conditionally independent features. Despite this assumption, the proposed method always outperforms Naive Bayes. We explained the differences among the different algorithms under the lens of probability matrices factorization.

Despite being the discussion on implementations at an extremely, we stress throughout the thesis the practicality of possible hardware implementations of the considered method. The DIN could be easily implemented on large scale in hardware having the following qualities: modularity, flexibility and locality of the computations. They are in fact composed of infonodes and combiners that can be arranged in any desired topology (provided there are no loops) without restriction imposed by non local communications. Moreover, the DIN algorithms are better suited for custom hardware implementation than their closest relative, decision trees. Differently from decision trees, in fact, the system design choices are simpler since both the execution times of all branches and the precise number of operations is fixed per datapoint and known in advance. On the contrary, with classical trees, nodes utilization depends on the single datapoint. An hardware engineer is thus forced to design the system considering the worst case scenario, possibly wasting In fact with classical trees, where nodes utilization depend on the datapoint, we are forced to design the system for the worst case, even if in the vast majority of time not all nodes are used.

Chapter 13

General Conclusions

This manuscript explored contributions to Efficient Machine Learning. The dissertation has been divided into two separate parts, explored independently.

The motivating problem of the first part has been uncertainty estimation for continuous parametric models, that is usually performed with algorithms that are extremely sensitive to hyperparameters. To partially overcome this problem, we presented a Stochastic Gradient based sampling algorithm, I-SGD, providing a sound theoretical characterization. Thanks to the proposed algorithm, the burden of hyperparameters optimization, in particular the learning rate scheduling, was lessened. We extended the idealised version of the algorithm to realistic settings, providing empirical validation against state of the art competitors.

The focus of future research on this topic will be the in depth study of the SG noise distribution. In particular, several challenging directions could be taken. The major one concerns the study of the interplay between the SG noise, the injected noise and the performance of the sampling methods. Is it possible, with a deeper theoretical understanding of the loss landscape and the noise covariance to design an even more efficient and robust sampler? What is the true bottleneck that prevents the usage across different scenarios of the same sampler, requiring the inefficient fine tuning procedure? For these reasons, an accurate understanding of the true noise distribution, the study of the characterization of the noise that should be injected and the impact on the overall performance would be of paramount importance.

The second part of the thesis focused on efficient algorithms for classifying discrete datasets. The driving target of the research has been the quest for algorithms easily deployable in hardware. We did propose a novel algorithm, Deep Information Networks (and an ensemble variant), based on a flexible and modular structure. We explored in depth the parallelizability properties of the considered method. The theoretical foundation of the proposed method is the Information Bottleneck principle, that provided a sound tool for the exploration of the method.

This first initial exploration of the method could be extended considering several interesting variations. Could we relax the constraints on the topology of the

networks, for example considering full mesh scenarios with possibly recurrent connections? Could such a choice further improve efficiency by allowing, for example, some form of asynchronous training across nodes? A second interesting direction concerns, similarly to the first part of the thesis, the investigation of modification to the algorithm to derive efficiently some of the hyperparameters, leveraging theoretical intuitions. Finally, we could consider whether some of the good properties of the considered algorithm, such as locality of computations, could be similarly exploited in architectures designed for continuous datasets, building for example deep neural networks with local computations of the cost function using the Information Bottleneck criterion.

Bibliography

- [1] Sungjin Ahn, Anoop Korattikara, and Max Welling. “Bayesian posterior sampling via stochastic gradient fisher scoring”. In: *Proceedings of the 29th International Conference on International Conference on Machine Learning*. 2012, pp. 1771–1778.
- [2] Babak Alipanahi et al. “Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning”. In: *Nature biotechnology* 33.8 (2015), pp. 831–838.
- [3] Dario Amodei et al. “Concrete problems in AI safety”. In: *arXiv preprint arXiv:1606.06565* (2016).
- [4] S. Arimoto. “An algorithm for computing the capacity of arbitrary discrete memoryless channels”. In: *IEEE Transactions on Information Theory* 18.1 (Jan. 1972), pp. 14–20. ISSN: 0018-9448. DOI: [10.1109/TIT.1972.1054753](https://doi.org/10.1109/TIT.1972.1054753).
- [5] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.
- [6] Michael K Bergman. *A Knowledge Representation Practionary*. Springer, 2018.
- [7] Christopher M. Bishop. *Pattern recognition and machine learning*. 1st ed. 2006. Corr. 2nd printing 2011. Springer, 2006. ISBN: 0387310738.
- [8] R. Blahut. “Computation of channel capacity and rate-distortion functions”. In: *IEEE Transactions on Information Theory* 18.4 (July 1972), pp. 460–473. ISSN: 0018-9448. DOI: [10.1109/TIT.1972.1054855](https://doi.org/10.1109/TIT.1972.1054855).
- [9] Mariusz Bojarski et al. “End to end learning for self-driving cars”. In: *arXiv preprint arXiv:1604.07316* (2016).
- [10] L. Breiman. “Bagging Predictors”. In: *Machine Learning* (1996).
- [11] L. Breiman. “Random Forests”. In: *Machine Learning* (2001).
- [12] William L Burke, William L Burke, and William L Burke. *Applied differential geometry*. Cambridge University Press, 1985.

- [13] Pratik Chaudhari and Stefano Soatto. “Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks”. In: *2018 Information Theory and Applications Workshop (ITA)*. IEEE. 2018, pp. 1–10.
- [14] Gal Chechik et al. “Information bottleneck for Gaussian variables”. In: *Journal of machine learning research* 6.Jan (2005), pp. 165–188.
- [15] Changyou Chen et al. “Bridging the gap between stochastic gradient MCMC and stochastic optimization”. In: *Artificial Intelligence and Statistics*. 2016, pp. 1051–1060.
- [16] Tianqi Chen, Emily Fox, and Carlos Guestrin. “Stochastic gradient hamiltonian monte carlo”. In: *International conference on machine learning*. 2014, pp. 1683–1691.
- [17] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.
- [18] Luc Devroye. “Nonuniform random variate generation”. In: *Handbooks in operations research and management science* 13 (2006), pp. 83–121.
- [19] Felix Draxler et al. “Essentially No Barriers in Neural Network Energy Landscape”. In: *International Conference on Machine Learning*. 2018, pp. 1309–1318.
- [20] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [21] Włodzisław Duch, Rafał Adamczak, and Krzysztof Grańbczewski. “Extraction of logical rules from neural networks”. In: *Neural Processing Letters* 7.3 (1998), pp. 211–219.
- [22] Rick Durrett. *Probability: theory and examples*. Vol. 49. Cambridge university press, 2019.
- [23] *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing, 2006.
- [24] Adriaan Daniël Fokker. “Die mittlere Energie rotierender elektrischer Dipole im Strahlungsfeld”. In: *Annalen der Physik* 348.5 (1914), pp. 810–820.
- [25] G. Franzese and M. Visintin. “Deep Information Networks”. In: *arXiv:1803.02251v1 [cs.LG]* (2018).
- [26] Giulio Franzese and Monica Visintin. “Probabilistic ensemble of deep Information networks”. In: *Entropy* 22.1 (2020), p. 100.
- [27] Yoav Freund and Robert Schapire. “A short introduction to boosting”. In: *Journal-Japanese Society For Artificial Intelligence* 14.771-780 (1999), p. 1612.

- [28] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *International Conference on Machine Learning, ICML*. 2016, pp. 1050–1059.
- [29] C. W. Gardiner. *Handbook of stochastic methods for physics, chemistry and the natural sciences*. Third. Vol. 13. Springer Series in Synergetics. Springer-Verlag, 2004. ISBN: 3-540-20882-8.
- [30] Timur Garipov et al. “Loss surfaces, mode connectivity, and fast ensembling of dnns”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 8789–8798.
- [31] Tomáš Gedeon, Albert E Parker, and Alexander G Dimitrov. “The mathematical structure of information bottleneck methods”. In: *Entropy* 14.3 (2012), pp. 456–479.
- [32] Mark Girolami and Ben Calderhead. “Riemann manifold langevin and hamiltonian monte carlo methods”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2 (2011), pp. 123–214.
- [33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [34] Chuan Guo et al. “On calibration of modern neural networks”. In: *arXiv preprint arXiv:1706.04599* (2017).
- [35] JM Hammersley and DC Handscomb. “Monte Carlo methods, methuen & co”. In: *Ltd., London* 40 (1964).
- [36] David J Hand and Keming Yu. “Idiot’s Bayes—not so stupid after all?” In: *International statistical review* 69.3 (2001), pp. 385–398.
- [37] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. “The Elements Of Statistical Learning”. In: *Aug, Springer* 1 (Jan. 2001). DOI: [10.1007/978-0-387-21606-5_7](https://doi.org/10.1007/978-0-387-21606-5_7).
- [38] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [39] Tin Kam Ho. “A Theory of Multiple Classifier Systems And Its Application to Visual Word Recognition”. PhD thesis. Citeseer, 1992.
- [40] Finn V Jensen et al. *An introduction to Bayesian networks*. Vol. 210. UCL press London, 1996.
- [41] Xiaoqian Jiang et al. “Calibrating predictive model estimates to support personalized medicine”. In: *Journal of the American Medical Informatics Association* 19.2 (2012), pp. 263–274.
- [42] Steven M Kay. *Fundamentals of statistical signal processing*. Prentice Hall PTR, 1993.

- [43] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [44] Itô Kiyosi. “Stochastic integral”. In: *Proc. Imperial Acad. Tokyo*. Vol. 20. 1944, pp. 519–524.
- [45] Peter E Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*. Vol. 23. Springer Science & Business Media, 2013.
- [46] Kolmogorov and Fomin. *Introductory real analysis*. Courier Corporation, 1975.
- [47] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).
- [48] H. Kushner and G.G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Stochastic Modelling and Applied Probability. Springer New York, 2003. ISBN: 9780387008943.
- [49] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).
- [50] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [51] Jacques Levy Vehel, Anne Philippe, and Caroline Robet. “Explicit and combined estimators for stable distributions parameters”. In: (Nov. 2018).
- [52] Chunyuan Li et al. “Preconditioned stochastic gradient Langevin dynamics for deep neural networks”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.
- [53] Lennart Ljung, Georg Pflug, and Harro Walk. *Stochastic Approximation and Optimization of Random Systems*. CHE: Birkhauser Verlag, 1992. ISBN: 3764327332.
- [54] YuGuang Long, LiMin Wang, and MingHui Sun. “Structure Extension of Tree-Augmented Naive Bayes”. In: *Entropy* 21.8 (July 2019), p. 721. ISSN: 1099-4300. DOI: [10.3390/e21080721](https://doi.org/10.3390/e21080721). URL: <http://dx.doi.org/10.3390/e21080721>.
- [55] Yi-An Ma, Tianqi Chen, and Emily Fox. “A complete recipe for stochastic gradient MCMC”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2917–2925.
- [56] Wesley J Maddox et al. “A simple baseline for bayesian uncertainty in deep learning”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 13132–13143.

- [57] Stephan Mandt, Matthew D Hoffman, and David M Blei. “Stochastic gradient descent as approximate bayesian inference”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 4873–4907.
- [58] Rowan McAllister et al. “Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning”. In: International Joint Conferences on Artificial Intelligence, Inc. 2017.
- [59] N Metropolis. “The beginning”. In: *Los Alamos Science* 15 (1987), pp. 125–130.
- [60] Seyed-Mohsen Moosavi-Dezfooli and Omar Fawzi Alhussein Fawzi. “Pascal Frossard.” In: *Universal adversarial perturbations.* " 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017.
- [61] K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
- [62] Radford M Neal et al. “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.
- [63] M Newman and G Barkema. *Monte carlo methods in statistical physics chapter 1-4*. Vol. 24. Oxford University Press: New York, USA, 1999.
- [64] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 427–436.
- [65] Mohammad Norouzi et al. “Efficient Non-greedy Optimization of Decision Trees”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 1729–1737. URL: <http://papers.nips.cc/paper/5886-efficient-non-greedy-optimization-of-decision-trees.pdf>.
- [66] Sam Patterson and Yee Whye Teh. “Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex”. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 3102–3110.
- [67] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.
- [68] VM Planck. “Über einen Satz der statistischen Dynamik und seine Erweiterung in der Quantentheorie”. In: *Sitzungsberichte der* (1917).
- [69] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.
- [70] J.R. Quinlan. “C4.5: Programs for Machine Learning”. In: *Morgan Kaufmann Publishers* (1993).

- [71] J.R. Quinlan. “Improved Use of Continuous Attributes in C4.5”. In: *Journal of Artificial Intelligence Research* (1996).
- [72] Riccardomannella and Peter McClintock. “ITO VERSUS STRATONOVICH: 30 YEARS LATER”. In: *Fluctuation and Noise Letters* 11 (May 2012). DOI: [10.1142/S021947751240010X](https://doi.org/10.1142/S021947751240010X).
- [73] Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.
- [74] Lior Rokach and Oded Z Maimon. *Data mining with decision trees: theory and applications*. Vol. 69. World scientific, 2008.
- [75] Asif Salekin and John Stankovic. “Detection of chronic kidney disease and selecting important predictive attributes”. In: *2016 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE. 2016, pp. 262–270.
- [76] Andrew M Saxe et al. “On the information bottleneck theory of deep learning”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124020.
- [77] Ohad Shamir, Sivan Sabato, and Naftali Tishby. “Learning and generalization with the information bottleneck”. In: *Theoretical Computer Science* 411.29-30 (2010), pp. 2696–2711.
- [78] Claude Elwood Shannon. “A mathematical theory of communication”. In: *ACM SIGMOBILE mobile computing and communications review* 5.1 (2001), pp. 3–55.
- [79] Umut Şimşekli. “Fractional Langevin Monte Carlo: Exploring Lévy driven stochastic differential equations for Markov Chain Monte Carlo”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3200–3209.
- [80] Umut Şimşekli, Levent Sagun, and Mert Gürbüzbalaban. “A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks”. In: *Proceedings of the 28th international conference on machine learning, ICML*. 2019, pp. 5827–5837.
- [81] Umut Şimşekli et al. “On the Heavy-Tailed Theory of Stochastic Gradient Descent for Deep Neural Networks”. In: *arXiv preprint arXiv:1912.00018* (2019).
- [82] Noam Slonim and Naftali Tishby. “Agglomerative information bottleneck”. In: *Advances in neural information processing systems*. 2000, pp. 617–623.
- [83] Paul Smolensky. “Connectionist AI, symbolic AI, and the brain”. In: *Artificial Intelligence Review* 1.2 (1987), pp. 95–109.
- [84] John Smythe et al. “Ito versus Stratonovich revisited”. In: *Physics Letters A* 97.3 (1983), pp. 95–98.

- [85] Jost Tobias Springenberg et al. “Bayesian optimization with robust Bayesian neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 4134–4142.
- [86] Susanne Still. “Information bottleneck approach to predictive inference”. In: *Entropy* 16.2 (2014), pp. 968–989.
- [87] Susanne Still. “Thermodynamic cost and benefit of data representations”. In: *arXiv preprint arXiv:1705.00612* (2017).
- [88] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and policy considerations for deep learning in NLP”. In: *arXiv preprint arXiv:1906.02243* (2019).
- [89] Neil C Thompson et al. “The Computational Limits of Deep Learning”. In: *arXiv preprint arXiv:2007.05558* (2020).
- [90] Tin Kam Ho. “Random decision forests”. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. 1995, 278–282 vol.1.
- [91] N. Tishby, F.C. Pereira, and W. Bialek. “The Information Bottleneck method”. In: *arXiv:physics/0004057v1* (2000).
- [92] N. Tishby and N. Zaslavsky. “Deep Learning and the Information Bottleneck Principle”. In: *arXiv:1503.02406v1 [cs.LG]* (2015).
- [93] *UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences*. <http://archive.ics.uci.edu/ml>. Accessed: 2010-09-30.
- [94] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.
- [95] Max Welling and Yee W Teh. “Bayesian learning via stochastic gradient Langevin dynamics”. In: *Proceedings of the 28th international conference on machine learning, ICML*. 2011, pp. 681–688.
- [96] Eugene Wong and Moshe Zakai. “On the convergence of ordinary integrals to stochastic integrals”. In: *The Annals of Mathematical Statistics* 36.5 (1965), pp. 1560–1564.
- [97] Jin Xiao, Changzheng He, and Xiaoyi Jiang. “Structure identification of Bayesian classifiers based on GMDH”. In: *Knowledge-Based Systems* 22.6 (2009), pp. 461–470.
- [98] Ruqi Zhang et al. “Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning”. In: *International Conference on Learning Representations, ICLR*. 2020.

BIBLIOGRAPHY

- [99] Xiangyu Zhang et al. “Accelerating Very Deep Convolutional Networks for Classification and Detection”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 38.10 (Oct. 2016), pp. 1943–1955.
- [100] Zhanxing Zhu et al. “The Anisotropic Noise in Stochastic Gradient Descent: Its Behavior of Escaping from Sharp Minima and Regularization Effects”. In: *International Conference on Machine Learning*. 2019, pp. 7654–7663.

This Ph.D. thesis has been typeset by means of the T_EX-system facilities. The typesetting engine was pdfL^AT_EX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete T_EX-system installation.