Robust and scalable probabilistic machine learning methods with applications to the airline industry

Rosa Candela

Robust and Scalable Probabilistic Machine Learning Methods with Applications to the Airline Industry

Rosa Candela

A doctoral dissertation submitted to:

SORBONNE UNIVERSITY

In Partial Fulfillment of the Requirements for the Degree of:

Doctor of Philosophy

Specialty : Computer Science, Telecommunications and Electronics

Committee in Charge:

PIETRO MICHIARDI	EURECOM	ADVISOR
MAURIZIO FILIPPONE	EURECOM	CO-ADVISOR
MARCO LORENZI	INRIA	REVIEWER
ELENA BARALIS	POLITECNICO DI TORINO	REVIEWER
MARCO MELLIA	POLITECNICO DI TORINO	EXAMINER
ALIX LHERITIER	AMADEUS	EXAMINER
MELEK ÖNEN	EURECOM	EXAMINER

February 2021

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures. Portions of the contents that appear in this dissertation have been published before in:

- R. Candela, P. Michiardi, M. Filippone, M.A. Zuluaga. *Model Monitoring and Dynamic Model Selection in Travel Time-series Forecasting.* European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD), Ghent 2020.
- R. Candela, G. Franzese, M. Filippone, P. Michiardi. *On the Convergence of Sparse Asynchronous SGD. Submitted to* International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto 2021.
- G. Franzese, R. Candela, D. Milios, M. Filippone, P. Michiardi. *Isotropic SGD: a Practical Approach to Bayesian Posterior Sampling.* arXiv preprint arXiv:2006.05087.

Rosa CANDELA
February 2021

Acknowledgements

In these three years I have been surrounded by many people, who gave me a great deal of support and assistance. I owe all of them a debt of gratitude and I would like to use the following lines to express my thankfulness.

First of all, I would like to express my gratitude to my supervisor Pietro Michiardi for proposing me this PhD and supporting me in every challenge I faced during these years. His motivation for research was important for pursuing these studies for me and his ideas were the source of the major contributions of this thesis. He has been a grounded and supportive figure for me both professionally and personally.

I am very grateful to my co-supervisor Maurizio Filippone for continuously sharing his ideas to improve my work, patiently explaining me theoretical concepts and supporting me every step of the way. His spirit of collaboration has been instrumental in creating a machine learning team inside our department.

Many thanks to my Amadeus supervisors Maria A. Zuluaga and Alix Lheritier for giving me the possibility to link my research work to a practical application. Their precious help and constant presence were crucial for succeeding in this work.

I would like to thank also all PhD students and research scientists I have met in these three years at the Data Science department of EURECOM. I found a stimulating environment and I spent nice moments with many of them.

Thanks to Amadeus ART team for the professional environment and the good times we had, even while working from home.

These years have been also the opportunity for personal growth and I would like to thank all friends who surrounded me during this period for the great moments. A special thank to Yves-Henri for his precious presence and support.

Finally, I would like to thank my family for supporting me throughout this journey. As with all my previous achievements, none of this would have been possible without their help, patience and encouragement.

Abstract

In the airline industry, price prediction plays a significant role both for customers and travel companies. On the customer side, due to the dynamic pricing strategies adopted by airlines, prices fluctuate over time and the resulting uncertainty causes worries among travellers. On the other side, travel companies are interested in performing accurate short- and long-term price forecasts to build attractive offers and maximize their revenue margin. However, price time-series comprise time-evolving complex patterns and non-stationarities, which make forecasting model deteriorate over time. Thus an efficient model performance monitoring is key to ensure accurate forecasts. Beside this, the steady growth of airline traffic leads to the development of massive datasets, which call for automatic procedures. In this thesis we introduce two machine learning applications which are crucial for the airline industry. The first one is meant to answer the question about the best moment to buy a ticket, through the use of specific metrics which help travellers taking the best decision about whether to buy or wait, while the second one focuses on the problem of model performance monitoring in industrial applications and it consists in a data-driven framework, built on top of the existing forecasting models, which estimates model performance and performs dynamic model selection.

Stochastic Gradient Descent (SGD) represents the workhorse optimization method in the field of machine learning. When dealing with complex models and massive datasets, we resort to distributed systems, whereby multiple nodes compute stochastic gradients using partitions of the dataset and model parameters are updated aggregating information from all nodes. In asynchronous systems, convergence of SGD is challenging because distributed workers might produce gradient updates for a loss computed on stale versions of the current model iterates. In this context, one technique that has achieved remarkable results, albeit in synchronous setups, is sparsification, in that it reduces communication overheads. In this thesis we fill the gap in the literature and study sparsification methods in asynchronous settings. For the first time, we provide a concise and simple convergence rate analysis when the joint effects of sparsification and asynchrony are taken into account, and show that sparsified SGD converges at the same rate of standard SGD.

Recently, SGD has played an important role also as a way to perform approximate Bayesian Inference, which is a principled way of developing probabilistic models. Stochastic gradient MCMC algorithms use indeed SGD with constant learning rate to obtain samples from the posterior distribution. Despite mathematical elegance and some promising results restricted to simple models, most of the existing works fall short in easily dealing with the complexity of the loss landscape of deep models, for which stochastic optimization poses serious challenges. Existing methods hence result often unpractical, because they require ad-hoc, sophisticated vanishing learning rate schedules, and hyper-parameter tuning. In this thesis we introduce a novel, practical approach to posterior sampling, which makes the SG noise isotropic using a fixed learning rate and that requires weaker assumptions than existing algorithms.

Table of Contents

List of Figures

List of Tables

Glossary

CDE Conditional Density Estimation. 15, 16, 19

CDF Cumulative Distribution Function. 23, 24, 27

CNNs Convolutional Neural Networks. 37, 38

DTD Days Before Departure. ii, 11, 12

GPs Gaussian Processes. ii, 37, 38, 41, 43–45

LSTM Long Short-Term Memory. 37, 40, 41, 43

MCMC Markov Chain Monte Carlo. 4, 6, 52, 53, 96

ML Machine Learning. 1, 2, 5

RMSE Root Mean Squared Error. ii, iv, 41–43

SGD Stochastic Gradient Descent. 3–6, 49–51, 53–56, 96, 98

sMAPE Symmetric Mean Absolute Percentage Error. ii, iv, 36, 37, 41–45

VaR Value at Risk. 23–25

1
Introduction

Over the past two decades Machine Learning (ML) has played a central role in numerous fields of information technology and its presence in our life continues to expand ever more widely. With the ever increasing amounts of data becoming available, smart data analysis will become even more pervasive as a necessary ingredient for technological progress. Classical examples of machine learning applications include image recognition, speech recognition, web page ranking, automatic translation, named entity recognition, recommender systems, forecasting, medical diagnoses.

All machine learning models share the same principle: use a set of training data to learn a task and predict the output for new input data. We can classify three main types of ML algorithms. In supervised learning the model is provided with example inputs that are labeled with their desired outputs and the goal is to predict label values on additional unlabeled data, based on the learned patterns. In unsupervised learning data is unlabeled, so the learning algorithm is left to find commonalities among its input data, either to discover hidden patterns within a dataset or to do feature learning. A third class of algorithms is represented by reinforcement learning, where the model interacts with an environment and there is a feedback loop between the learning system and its experiences.

Based on the type of problem, different ML models can be adopted. We distinguish two main type of algorithms: deterministic and probabilistic. Deterministic models yield a single solution describing the outcome of some experiment given appropriate inputs. A probabilistic model is, instead, meant to give a distribution of possible outcomes. Belong to the first class some very well known algorithms such as Linear Regression, Random Forest, Support Vector Machines, K-Nearest Neighbors, Decision Trees, (Deep) Neural Networks. Some examples for probabilistic models are Logistic Regression, Bayesian Classifiers, Hidden Markov Models, Gaussian Processes, Bayesian Neural Networks. For a complete description of ML techniques we refer the reader to [1, 2].

The widespread success of ML in several fields of our life is accompanied by the ever large availability of data, which represent the input of such models. The three primary data sources are: social data, machine data and transactional data. Social data comes from the public web, but also from the Likes, Tweets, Comments, Video Uploads, and general media that are uploaded and shared via the world's social media platforms. Machine data is

defined as information which is generated by industrial equipment, sensors that are installed in machinery, and even web logs which track user behavior. This type of data is expected to grow exponentially as the internet of things grows ever more pervasive and expands around the world. Transactional data is generated from all the daily transactions that take place both online and offline. Invoices, payment orders, storage records, delivery receipts. All these sources bundled in one term give the well known Big Data concept.

Among the numerous fields which benefit from ML techniques we find the airline industry, where the amount of available data has seen a great increase in the last few decades. According to the International Air Transport Association (IATA), in 2009 the number of scheduled passengers boarded by the global airline industry has been of 2.48 billion and it reached 4.54 in 2019. Beside this, airlines continue to increase the number of city-pair routes and in 2019 almost 22 thousand city pairs were regularly serviced by airlines. This opens the way to the possibility of applying ML techniques both to increase airlines' revenue and improve traveller's experience. Examples of ML applications for the airline industry are reinforcement learning for revenue management, dynamic pricing, recommender systems for hotels and ancillary services, price prediction, baggage weight prediction, flight delay prediction.

Among these, price prediction plays a significant role both for customers and travel companies. The former are indeed interested in knowing the price evolution to take the best decision on when to buy and travel, the latter want instead to offer attractive tour packages and maximize their revenue margin. Regarding the application on the customer side, this is the result of the large uncertainty derived from airline pricing strategies, which make prices fluctuate a lot over time. Airlines, indeed, put in place proprietary software to compute ticket prices on any given day, based on several factors, including seasonality, availability of seats, competitive moves by other airlines [3]. Thus an ever increasing number of people becomes interested in understanding how corporations vary prices, to choose the best moment to book a flight. As a result, several Online Travel Agencies nowadays try to satisfy users, by incorporating such information on their website, either in the form of suggestion or price forecasting, which opens the way to the development of suited machine learning applications.

On the other side then, travel companies are interested in performing accurate short- and long-term price forecasts to build attractive offers and maximize their revenue margin. Here again the power of machine learning comes in handy, but with some challenges to be faced. Price time-series comprise time-evolving complex patterns, non-stationarities or, more generally, distribution changes over time, which make forecasting models deteriorate over time. Thus an efficient model performance monitoring is key to ensure accurate forecasts over time. Beside this, the steady growth of airline traffic leads to the development of massive datasets, which call for automatic procedures. This implies the building of machine learning systems, where the model, representing the core for the application to run, is surrounded by an infrastructure which performs monitoring, maintenance and improvement over time [4].

In light of the above, in the first part of this thesis we will introduce two machine learning

applications which are crucial for the airline industry. The first one is meant to answer the question about the best moment to buy a ticket, through the use of specific metrics which help travellers taking the best decision about whether to buy or wait, while the second one focuses on the problem of model performance monitoring in industrial applications and it consists in a data-driven framework, built on top of the existing forecasting models, which estimates model performance and performs dynamic model selection.

So far we have seen how machine learning can be used to improve airline industry applications. Hereafter, we will focus on one of the core components of machine learning, that is optimization. The essence of most machine learning algorithms is indeed to build an optimization model and learn the parameters in the objective function from the given data. Since the beginning of machine learning phenomenon, a series of effective optimization methods were put forward, which have improved the performance and efficiency of machine learning methods. From the perspective of the gradient information in optimization, popular optimization methods can be divided into three categories [5]: first-order optimization methods, which are represented by the widely used stochastic gradient methods; high-order optimization methods, in which Newton's method is a typical example; and heuristic derivative-free optimization methods, in which the coordinate descent method is a representative.

Despite the research advances seen by high-order optimization methods, Stochastic Gradient Descent (SGD) still represents the workhorse optimization method in the field of machine learning. The idea of the gradient descent methods is that variables update iteratively in the (opposite) direction of the gradients of the objective function. The update is performed to gradually converge to the optimal value of the objective function. The learning rate determines the step size in each iteration, and thus influences the number of iterations to reach the optimal value. The mini-batch version [6], in particular, evaluates the gradient on a small subset of samples and hence it allows to achieve the best trade-off between number of iterations and cost per iteration. The success of SGD comes also from its two important statistical properties: it is an unbiased estimator of the full gradient and it provides convergence guarantees [7].

SGD represents also the core of large-scale machine learning, whereby multiple nodes compute stochastic gradients using partitions of the dataset and model parameters are updated aggregating information from all nodes. Large-scale machine learning has received a lot of attention in recent years, due to the development of complex models trained on massive datasets. Modern deep learning models can have indeed up to billions of parameters and common datasets include millions of samples. This has led to the use of distributed systems, where multiple nodes perform model training in parallel and achieve a good speedup. Using the common *Parameter Server* paradigm, stochastic gradients computed by the workers are aggregated by a central nodes, which update the model's parameters and send them back to the nodes. The aggregation step can be done either each time a worker sends its gradient, as in asynchronous systems, or only when all nodes have completed the training step, as in synchronous architectures. The analysis of the convergence behavior of SGD, both in synchronous and asynchronous settings has been widely studied in the literature [8, 9, 10, 11].

In asynchronous systems, convergence is challenging because distributed workers might produce gradient updates for a loss computed on stale versions of the current model iterates. In this context, communication overheads have been considered as a key issue to address, and a large number of works have been proposed to mitigate such overheads [12, 13]. One technique that has achieved remarkable results, albeit in synchronous setups, is sparsification. The key idea of this method is to apply smaller and more efficient gradient updates, by applying a sparsification operator to the stochastic gradient. In Chapter 6 we fill the gap in the literature and study sparsification methods in asynchronous settings. For the first time, we provide a concise and simple convergence rate analysis when the joint effects of sparsification and asynchrony are taken into account, and show that sparsified SGD converges at the same rate of standard SGD.

One last important role of SGD we want to highlight in this work relates to Bayesian Inference. Latest advances in machine learning research have led to the use of such models in an ever increasing number of applications, including ones previously requiring human expertise. However, the adoption of machine learning models to sensitive applications, such as self-driving cars, automated medical diagnosis, is still under study and heavily-publicised incidents have cast a heavy shadow over their robustness and reliability [14, 15]. This phenomenon has hence sparked a resurgence of interest in probabilistic modelling, which is inherently intended to capture the uncertainty or lack of knowledge a model might have about a learned task. To this end, probabilistic models provide uncertainty estimates accompanying predictions, which give information on how likely the latter are to be correct. Bayesian inference represents a principled way of developing probabilistic models. Using Bayes' theorem to find the posterior distribution of a model's parameters, one can combine prior beliefs on the values they can take with the likelihood of observing available data using the designated model configuration. This allows one to compute the predictive distribution on new data and use the resulting variance to estimate the uncertainty associated to the predictions. Although conceptually simple, applying Bayesian inference poses a number of computational challenges, due to high-dimension operations and equations that cannot be solved analytically. To this end, several methods have been proposed in the literature, including Markov Chain Monte Carlo (MCMC) and Variational Inference (VI). Although widely used, these methods require either long time or large number of parameters to converge. Recently, another family of approaches that has taken hold in the field of approximate Bayesian inference are the so-called stochastic gradient MCMC algorithms, which use SGD with constant learning rate to obtain samples from the posterior distribution [16, 17, 18]. Despite mathematical elegance and some promising results restricted to simple models, most of these works fall short in easily dealing with the complexity of the loss landscape of deep models, for which stochastic optimization poses serious challenges. Existing methods are often unpractical, as they require ad-hoc, sophisticated vanishing learning rate schedules, and hyper-parameter tuning. In this regard, we introduce in chapter 7 a novel, practical approach to posterior sampling, which makes the Stochastic Gradient noise isotropic using a fixed learning rate and that requires weaker assumptions than existing algorithms.

The content of this thesis is organised in two main parts. The first one, whose title is Robust

Machine Learning Methods for the Airline Industry, describes how machine learning can be of paramount importance in the airline industry and it includes the following chapters:

- Chapter 2 introduces Part I by providing examples of ML applications in the airline industry. In particular, the tasks of price prediction and model performance monitoring are discussed.

- Chapter 3 shows how ticket prices represent a source of worries for travellers, due to their arbitrary and unpredictable evolution. We therefore investigate how some machine learning models handle uncertainty estimation and we propose four approaches to help travellers to deal with the uncertainty in ticket price evolution using such models. In particular, the first one uses a probabilistic model to estimate the conditional density estimation of the price in the future. The second and the third approaches instead provide the user with more explicit information to decide when to buy a ticket, by estimating the amount of money that is lost when choosing the wrong date. Finally, the last approach gives to the user financial protection against adverse future price fluctuations.

- In Chapter 4 we consider the issues related to monitoring, maintenance and improvement of complex production-deployed ML systems for time-series forecasting. We thus propose a data-driven framework which builds a monitoring system on top of deployed models. The framework estimates the forecasting error of time-series forecasting models over time and uses it as surrogate measure of the model's future performance. In this way, continuous monitoring is achieved by using the predicted forecasting error as a measure to detect degrading performance. Simultaneously, the predicted forecasting error enables model maintenance by allowing to rank multiple models based on their predicted performance, i.e. model comparison, and then select the one with the lowest predicted error measure, i.e. model selection. In the chapter we show an example of usage with price forecasts.

The second part is called Scalable Probabilistic Machine Learning and it studies the use of SGD for optimization in large-scale machine learning and Bayesian Inference. In particular:

- Chapter 5 provides the context of Part II, by discussing SGD convergence properties. We then introduce the concept of large-scale machine learning and we give background information about Bayesian inference.

- In Chapter 6 we focus on asynchronous systems, where multiple nodes compute stochastic gradients using partitions of the datasets and convergence of SGD is challenging because workers might produce gradient updates for a loss computed on stale versions of the current model iterates. We thus study the role of sparsification, a technique used to reduce communication overheads in synchronous setups. For the first time, we provide a concise and simple convergence rate analysis when the joint effects of sparsification and asynchrony are taken into account, and show that sparsified SGD converges at the same rate of standard SGD. Our empirical analysis of sparsified

SGD complements our theory. We consider several delay distributions and show that, in practice, applying sparsification does not harm SGD performance. These results carry over when the system scales out.

- In Chapter 7 we consider SGD as a means of performing approximate Bayesian inference, by using a constant learning rate to obtain samples from the posterior distribution. Most of the stochastic gradient MCMC algorithms recently proposed indeed fall short in easily dealing with the complexity of the loss landscape of deep models, for which stochastic optimization poses serious challenges. Here we introduce a novel and practical approach to posterior sampling, which makes the Stochastic Gradient noise isotropic using a fixed learning rate and that requires weaker assumptions than existing algorithms. We run an extensive experimental campaign, where we compare our approach to a number of alternatives which have been successfully applied to the Bayesian deep learning setting. The results indicate that the proposed approach is competitive to the state-of-the-art, in terms of accuracy and uncertainty.

Finally, in Chapter 8 we summarise the principal themes and contributions presented in this work and we expand upon the directions for future work.

Part I

Robust Machine Learning Methods for the Airline Industry

2
Context

In this first part of the thesis we will describe how machine learning can be of paramount importance in the airline industry, by providing some practical examples of applications.

2.1 ML applications in the airline industry

Among the numerous industrial fields that have taken advantage of machine learning to improve their products it is worth to mention the airline industry. If we look at the air traffic trend, we see an impressive growth in the last 30 years, which results in a total of 4.54 billion of passengers registered by the global airline industry in 2019. This gives us an idea of the amount of data airline industry applications have at their disposal to improve both airline and travelers life, but also the need for automatic procedures capable of handling such massive datasets. Here comes the role of machine learning and, more generally, of artificial intelligence.
The objectives of ML applications in travel industry are manifold. In this work we set on the customer side and we distinguish three main travel-centric pillars. The first one is understanding the behavior of travellers, to better propose offers that might enrich their experience or alternatives that could improve their journey and satisfaction. Secondly, by analyzing behavior and by clustering travelers, we can predict their actions through choice modelling. Finally, we can improve the travel experience, by increasing the efficiency of various travel related processes. To this end, examples of ML applications are: recommender systems for hotels [19] and ancillary services, price prediction, airlines fare family design, baggage weight prediction, flight delay prediction, disruption management, airport takeoff sequencing optimization, seat allocation optimization. Other applications related to airlines are reinforcement learning for revenue management [20], dynamic pricing. In this work we focus on price prediction, as explained in the following.

2.2 Price prediction

Modern airlines develop their pricing strategies mostly based on yield management theories [21]. Sophisticated mathematical models are used to determine the real-time airfare based

on information such as unsold seats or recent market demand, which are usually guarded commercial secrets. Consequently, it is often difficult for an ordinary customer to estimate how the price will change in the future, which results in a source of worry. This paved the way to two main applications of price prediction. The first one is related to price forecasting, whereby the objective is to predict the future price. Although conceptually simple, airfare forecasting is a challenging problem, in that prices can be subjected to drastic fluctuations and non-stationary patterns, which compels forecasting model to be updated. The second application on price prediction is more customer-oriented and it is related to finding the optimal purchase time. This is done by providing suggestions whether it is better to *buy now* or *wait* during a flight search. In this respect, we will discuss in Chapter 3 how ML methods can help travelers to deal with uncertainty in ticket price evolution, proposing also some real use-cases that demonstrate the utility of having such decision-making tools.

2.3 Model performance monitoring

The travel industry is one of the field that lends itself to the use of machine learning to improve its product and we have seen several examples of ML applications. However, development and deployment of ML models only represent the first steps of system's life cycle. In [4] authors describe ML systems as made of many building blocks, where ML code represents only a small fraction, but it is surrounded by a vast and complex infrastructure, which includes data collection, data verification, machine resource management, analysis tool, process management tool, feature extraction, configuration, serving infrastructure and monitoring.

When dealing with industrial applications, monitoring, maintenance and improvement of complex production-deployed ML systems carry most of the costs and difficulties in time. Model monitoring refers to the task of constantly tracking a model's performance to determine when it degrades, becoming obsolete. Once a degradation in performance is detected, model maintenance and improvement take place to update the deployed model by rebuilding it, re-calibrating it.

Coming back to the price forecasting application, we need to consider some additional challenges for model monitoring. Firstly, such applications involve the analysis of multiple time-series which are modeled independently. For example, according to the 2019 world air transport statistics report, there were almost 22 thousands city pairs directly connected by airlines. So for scalability purposes, we need something able to automatically monitor the deployed models. Secondly, as mentioned above, price time-series comprise time-evolving complex patterns, non-stationarities, which make forecasting models more prone to deteriorate over time. Regarding the monitoring of in-production models, another challenge need to be considered. Usually industrial applications already use ML models, which have been developed recently or not, so introducing a monitoring system can be very expensive.

In this respect, we will introduce in Chapter 4 a data-driven framework, which builds a monitoring system on top of deployed models, with an example of price forecasting application.

3
Dealing with Uncertainty in Ticket Price Evolution

Airline industry is known to use convoluted dynamic pricing strategies with the goal of maximizing its revenue. Airlines have many fare classes for seats on the same flight, use different sales channels and frequently vary the price per seat over time based on some factors such as seasonality, availability of seats, competitive moves by other airlines [3]. They usually put in place proprietary software to compute ticket prices on any given day, whose algorithms are jealously guarded trade secrets. All this makes prices fluctuate a lot over time and gives cause for concerns among travellers, which would like to pay the cheapest price. In last few years, indeed, an ever increasing number of people seems interested in understanding how corporations vary prices over time, which has opened the way to the development of decision-making tools to help customers in finding the best moment to buy tickets [3, 22].

We can have an idea of travellers' behavior when buying a ticket by looking at the distribution of the Days Before Departure (DTD) people purchase tickets. In Figure 3.1 we show the results from the MIDT (Marketing Information Data Transfer) dataset, which contains information about indirect booking activities of several travel agencies and airline carriers. In 2019 the number of bookings between 3 months and one year before departure represents more than 16% of the total bookings. This tells us that in general there is a trend to buy tickets well in advance, on the belief that the price will increase getting close to the departure date. However, for the reasons we mentioned before about airlines' strategy, the rule to buy the ticket is not simply the earlier the better. We plot an example of price evolution with DTD in Figure 3.2, where we compare the observed curve with a theoretical one. The latter is meant to simulate the general people belief, according to which price will only increase getting close to the departure date, and it has been generated using a naive heuristic which at each time step adds a certain amount to the previous price. We observe that the real curve is far away from the theoretical one, in that it has a very irregular behavior, with several price increases and decreases.

So how can we help travellers to deal with this uncertainty in ticket price evolution? One simple approach is to provide information on the price evolution, just predicting the future

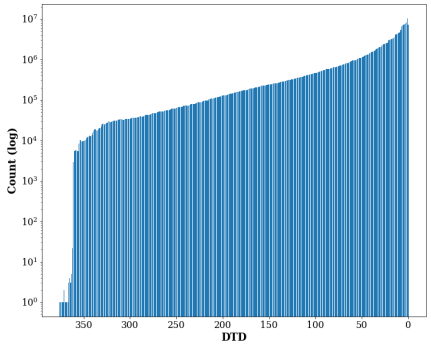


Figure 3.1 – Time before departure distribution in MIDT dataset.

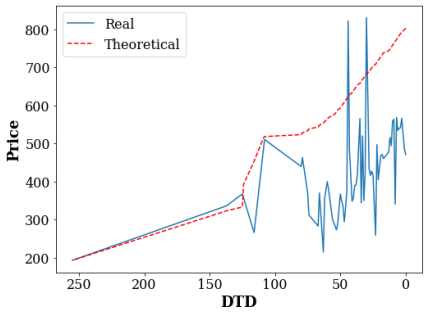


Figure 3.2 – Comparison between real and simulated price evolution with DTD.

price. Machine learning practitioners are used to do this, using simple or advanced regression algorithms. However, given the importance and difficulty underlying the problem, it is worth to have information about the prediction's reliability. To this end, we can do a step ahead and enrich price predictions with uncertainty estimation, which makes the user more confident of his decision based on the model output.

In light of this, we start this chapter with a description of how uncertainty estimation is handled by different ML methods. Later, we introduce four approaches which illustrate how we can help travellers to deal with uncertainty in ticket price evolution, using such models.

3.1 Uncertainty in ML methods

Estimate uncertainty in predictions is a crucial concept in multiple scenarios. First, uncertainty plays a central role on deciding when to abstain from prediction. This is the case when we want to identify rare examples which significantly differ from the majority of the data, as in anomaly, outliers and out-of-distribution examples detection [23, 24, 25], or when it comes to detect adversaries [26], delegate high-risk predictions to humans [27, 28]. Uncertainty is also the backbone of active learning [29], the problem of deciding what examples should humans annotate to maximally improve the performance of a model. Finally, uncertainty estimation is important when analyzing noise structures, for example to discover causal relations from large quantities of data [30], and in the estimation of predictive intervals. The latter are indeed defined as intervals containing the true value about some target variable, given the values for some input variable, with a given probability.

In general we can distinguish two main types of uncertainty [31]: aleatoric and epistemic. Aleatoric uncertainty (from the Greek word *alea*, meaning "rolling a dice") describes the variance of the conditional distribution of the target variable given the features. This type of uncertainty accounts hence for the stochasticity of the data and it arises due to hidden variables or measurement errors. This means that it cannot be reduced by collecting more data under the same experimental conditions. Epistemic uncertainty (from the Greek word *episteme*, meaning "knowledge") quantifies instead the errors associated to the lack

of experience of the model at certain regions of the feature space. Therefore, this type of uncertainty is inversely proportional to the density of training examples, and could be reduced by collecting data in those low density regions.

3.1.1 How ML methods deal with uncertainties

We will now describe how different ML methods handle quantification of uncertainty. Given that our primary application is price prediction, we will focus our discussion on the regression problem.
The standard setup is the following. Given a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, composed by feature vectors $\mathbf{x}_i \in \mathbb{R}^D$ and targets $y_i \in \mathbb{R}$, the goal is to estimate a function $f : \mathbb{R}^D \to \mathbb{R}$, which best describes the relationship between the vector of explanatory variables $\mathbf{x}$ and the dependent variable $\mathbf{y}$. Considering a linear regression model with Gaussian noise we have:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}, \quad \mathbf{y} = f(\mathbf{x}) + \varepsilon,$$

where $\mathbf{w}$ is a vector of weights (parameters) of the linear model. We assume here that the observed values $\mathbf{y}$ differ from the function values $f(\mathbf{x})$ by additive noise, and we also assume that this noise follows an independent, identically distributed Gaussian distribution with zero mean and variance σ_N^2, $\varepsilon \sim N\left(0, \sigma_N^2\right)$. The solution of the regression problem is given by the minimization of the loss $\mathcal{L} = \sum_{i=1}^N [y_i - \mathbf{w}^\top x_i] = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|$:

$$\nabla_{\mathbf{w}} \mathcal{L} = 0 \Rightarrow \quad \hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \tag{3.1}$$

where $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n)^T$ and $\mathbf{y} = (y_1, ..., y_n)^T$.
Minimizing the quadratic loss $[y_i - \mathbf{w}^\top x_i]^2$ is equivalent to maximizing the Gaussian likelihood function:

$$\exp(-\gamma \mathcal{L}) = \exp(-\gamma \left\|\mathbf{y} - \mathbf{X}\mathbf{w}\right\|^2) \propto \mathcal{N}\left(\mathbf{y} | \mathbf{X}\mathbf{w}, \frac{1}{2\gamma}\right)$$

For this reason, the solution $\hat{\mathbf{w}}$ in Equation (3.1) is commonly known as maximum likelihood estimator, in that it provides the model under which the observed data is most probable.
In machine learning we distinguish two classes of models: deterministic and probabilistic. The first ones, used by most of machine learning practitioners, through the maximum likelihood solution produce a single estimate of the function f, which then provides a point prediction $\hat{y}$ about the target, the second ones instead, combining the likelihood with a prior, give a probability distribution over the weights $\mathbf{w}$, which allows one to obtain a probability distribution over the prediction $\hat{y}$. Probabilistic models are indeed used when quantification of uncertainty is needed.

3.1.2 Bayesian regression

Bayesian linear regression [32] is an example of probabilistic model, which predicts a whole distribution over the target variable, offering a natural measure of prediction uncertainty.

In Bayesian linear regression, the posterior distribution over the weights $p(\mathbf{w} \mid \mathbf{X}, \mathbf{y})$ is computed using Bayes' theorem:

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})}{\int p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})d\mathbf{w}} = \frac{p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y} \mid \mathbf{X})}, \tag{3.2}$$

where the likelihood $p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})$ represents a measure of fitness, the prior $p(\mathbf{w})$ comprises anything we know about parameters $\mathbf{w}$ before we see any data and the marginal likelihood $p(\mathbf{y} \mid \mathbf{X})$ is a normalization constant, to ensure that the posterior satisfies the probability's property $\int p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) = 1$.

Once obtained the posterior distribution over the weights, we can compute the predictive distribution on new data $(\mathbf{x}^*, y^*)$ as follows:

$$p\left(y_* \mid \mathbf{X}, \mathbf{y}, \mathbf{x}_*\right) = \int p\left(y_* \mid \mathbf{x}_*, \mathbf{w}\right) p\left(\mathbf{w} \mid \mathbf{X}, \mathbf{y}\right) d\mathbf{w} \tag{3.3}$$

Intuitively, using Equation (3.3) we take the average of predictions of infinitely many models.

In order to estimate the uncertainty coming from this distribution, we consider its variance $\sigma^2\left(y_* \mid \mathbf{x}_*\right)$. According to the law of total variance, if X and Y are random variables on the same probability space, and the variance of Y is finite, this amounts to:

$$\mathrm{Var}(Y) = \mathrm{Var}(\mathbb{E}[Y \mid X]) + \mathbb{E}[\mathrm{Var}(Y \mid X)] \tag{3.4}$$

Using this expression, we can then decompose the total variance $\sigma^2\left(y_* \mid \mathbf{x}_*\right)$ as follows:

$$\sigma^2\left(y_* \mid \mathbf{x}_*\right) = \sigma^2_{p(\mathbf{w})}\left(\mathbb{E}\left[y_\star \mid \mathbf{w}, \mathbf{x}_\star\right]\right) + \mathbb{E}_{p(\mathbf{w})}\left[\sigma^2\left(y_* \mid \mathbf{w}, \mathbf{x}_*\right)\right], \tag{3.5}$$

where $\mathbb{E}\left[y_\star \mid \mathbf{w}, \mathbf{x}_\star\right]$ and $\sigma^2\left(y_\star \mid \mathbf{w}, \mathbf{x}_\star\right)$ are, respectively, the mean and variance of y_* according to $p\left(y_\star \mid \mathbf{w}, \mathbf{x}_\star\right)$.

In Equation (3.5), $\sigma^2_{p(\mathbf{w})}\left(\mathbb{E}\left[y_\star \mid \mathbf{w}, \mathbf{x}_\star\right]\right)$ is the variance of $\mathbb{E}\left[y_\star \mid \mathbf{w}, \mathbf{x}_\star\right]$ when $\mathbf{w} \sim p(\mathbf{w})$. This term ignores any contribution to the variance of y_* from ε and only considers the effect of $\mathbf{w}$. Therefore, it corresponds to the epistemic uncertainty in Equation (3.3). By contrast, the term $\mathbb{E}_{p(\mathbf{w})}\left[\sigma^2\left(y_* \mid \mathbf{w}, \mathbf{x}_*\right)\right]$ represents the average value of $\sigma^2\left(y_* \mid \mathbf{w}, \mathbf{x}_*\right)$ when $\mathbf{w} \sim p(\mathbf{w})$. This term ignores any contribution to the variance of y_* from $\mathbf{w}$ and, therefore, it represents the aleatoric uncertainty in Equation (3.3).

Therefore, adopting a Bayesian inference approach allows one to obtain a probability distribution over the target, whose predicting variance can be used for uncertainty estimation. Moreover, we can use the prior in Equation (3.2) to include our beliefs about the model parameters before looking at the observations. Here we have seen the example of Bayesian linear regression, where we assume that the relation between $\mathbf{y}$ and $\mathbf{x}$ is linear. However, often we need to model more complex systems. In this case, if the prior is not conjugate with the likelihood, that is they do not belong to the same probability distribution family, we resort to approximations [2, 33].

3.1.3 Conditional Density Estimation

Another approach which allows one to capture uncertainty in predictions is Conditional Density Estimation (CDE). Conditional density estimation is a general framing of supervised learning problems, subsuming both classification and regression. While the objective of most regression algorithms is to accurately predict the expected value of a label y conditional on observing associated features $\mathbf{x}$, $\mathbb{E}[y \mid \mathbf{x}]$, these methods generally have implicit or explicit probabilistic interpretations. CDE generalizes regression by modeling a full density $p(y \mid \mathbf{x})$.

Once obtained the distribution of y given x, we can compute its cumulative distribution function (CDF) and use it to construct prediction intervals. By denoting the model CDF with F, the predictive interval is defined as:

$$U_q(y \mid \mathbf{x}) = \left[F^{-1}\left(1 - \frac{q}{2} \mid \mathbf{x}\right), F^{-1}\left(1 + \frac{q}{2} \mid \mathbf{x}\right)\right] \tag{3.6}$$

For a model to reliably estimate its predictive uncertainty, the model CDF F should be estimated to be consistent with the data-generating CDF $F^*(y|\mathbf{x})$, such that, for example, the 95% predictive interval $U_{0.95}(y|\mathbf{x})$ indeed contains the observations $y \sim F^*(y|\mathbf{x})$ 95% of the time. This consistency property is known in the probabilistic forecast literature as calibration [34], and defines a mathematically rigorous condition for a model to achieve reliable estimation of its predictive uncertainty.

A number of methods exist for performing CDE which, in the limit of very large models and datasets, are able to capture arbitrary conditional distributions. These date back to adaptive mixtures of local experts [35] and mixture density networks (MDNs) [36]. MDNs are conceived to perform conditional density estimation, by finding the mixture of Gaussian distributions that model the target variable y, given input $\mathbf{x}$:

$$p(y \mid \mathbf{x}) = \sum_{j=1}^{K} \pi_j(\mathbf{x}) \mathcal{N}\left(y; \mu_j(\mathbf{x}), \Sigma_j(\mathbf{x})\right),$$

where $\pi_j(\mathbf{x}), \mu_j(\mathbf{x})$, and $\Sigma_j(\mathbf{x})$ are j-th mixture weight function, mean function, and variance function, respectively and K is the number of components. Indeed, in standard regression problems we assume that the conditional distribution $p(y \mid \mathbf{x})$ is Gaussian. However, in practical machine learning this is often not true, for example because the target distribution is multimodal. Thus such assumption may lead to poor performance. To account for these limitations, [36] proposed to parametrize the mixture of distributions by a deep neural network (DNN), creating the mixture density network, whose output is:

$$p(y \mid \theta) = \sum_{j=1}^{K} \pi_j \mathcal{N}\left(y \mid \mu_j, \Sigma_j\right),$$

where $\theta = \{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{K}$ is a set of parameters of the Gaussian mixture model (GMM). In other words, an MDN can be seen as a mapping f from an input $\mathbf{x} \in \mathbf{X}$ to the parameters

$\theta \in \Theta$ of a GMM of an output, i.e., $f : \mathbf{X} \mapsto \Theta$. Thus, the resulting (multimodal) conditional probability distribution allows one to model complex patterns found in real-world data. By estimating the probability distribution $p(y \mid \mathbf{x})$, it is possible to quantify uncertainty using the variance of the obtained GMM. In [37] authors use the law of total variance in Equation (3.4) to obtain:

$$\mathbb{V}(y \mid \mathbf{x}) = \sum_{j=1}^{K} \pi_j(\mathbf{x}) \left\| \mu_j(\mathbf{x}) - \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mu_k(\mathbf{x}) \right\|^2 + \sum_{j=1}^{K} \pi_j(\mathbf{x}) \Sigma_j(\mathbf{x}) \tag{3.7}$$

In Equation (3.7) the first term estimates our ignorance about the model prediction, so it corresponds to the epistemic uncertainty, while in the second one we find the predicted variance of the mixture models, which captures the noise inherent in data and so it refers to the aleatoric uncertainty.

More recent work, arising primarily from approaches in unsupervised learning, includes conditional variants of variational Autoencoders (VAEs) and generative adversarial networks (GANs) [38]. The general idea of autoencoders consists in setting an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimisation process. So, at each iteration the autoencoder architecture (the encoder followed by the decoder) is fed with some data, then the encoded-decoded output is compared with the initial data and the error is back-propagated through the architecture to update the weights of the networks. Thus, intuitively, the overall autoencoder architecture (encoder plus decoder) creates a bottleneck for data that ensures only the main structured part of the information can go through and be reconstructed. Generative Adversarial Networks differ from VAEs because they do not work with any explicit density estimation. In GANs the idea is to sample from a simple distribution like Gaussian and then learn to transform this noise to data distribution using universal function approximators such as neural networks. This is achieved by adversarial training of two networks, Generator and Discriminator. A generator model learns to capture the data distribution and a discriminator model estimates the probability that a sample came from the data distribution rather than model distribution.

Another research line of CDE is represented by nonparametric conditional density estimation, where only minimal assumptions are made about the smoothness of $p(y \mid \mathbf{x})$ and any parametric form is assumed. Freedom from parametric assumptions is very often desirable when dealing with complex data, as we rarely have knowledge of true distributional forms. In kernel conditional density estimation (KCDE), the probability distribution $\hat{f}(y \mid \mathbf{x})$ is estimated as follows:

$$\hat{f}(y \mid \mathbf{x}) = \frac{\sum_i K_{h_1}(y - y_i) K_{h_2}(\|\mathbf{x} - x_i\|)}{\sum_i K_{h_2}(\|\mathbf{x} - x_i\|)},$$

where $K_h(t)$ is a kernel function, i.e. a compact, symmetric probability distribution such as the Gaussian or Epanechnikov, d is the dimension of $\mathbf{x}$, n is the number of data points, and h is the bandwidth controlling the kernel widths. Due to the difficulty of selecting good bandwidths in the presence of large datasets and higher dimensionality, most KCDE

models have been applied only to bivariate data. In [39] authors propose a new method to speedup classical nonparametric kernel conditional density estimation, which focuses on efficiently selecting bandwidths to maximize cross-validated likelihood.
More recently, [40] introduces normalising flows in conditional density estimation. Normalizing flows, unlike VAEs, allow indeed tractable marginal likelihood estimation. These methods construct complex distributions by transforming a probability density through a series of invertible mappings. By repeatedly applying the rule for change of variables, the initial density flows through the sequence of invertible mappings. At the end of this sequence we obtain a valid probability distribution and hence this type of flow is referred to as a normalizing flow.

3.1.4 Quantile regression

Another solution to model the conditional density $p(y \mid \mathbf{x})$ is to estimate its quantiles as function of $\mathbf{x}$, which is known as quantile regression [41]. Quantiles represent indeed useful summary statistics to characterize the conditional distribution of the response. As an example, we may be interested in observing the effects of covariates on the upper or lower tails of the response distribution.
For $0 < \tau < 1$ the τ-th conditional quantile function $f_\tau : \mathcal{X} \to \mathbb{R}$ is estimated by the following minimization problem:

$$\hat{f}_\tau = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell_\tau \left(y_i - f\left(\boldsymbol{x}_i\right) \right), \quad \ell_\tau(\xi) = \begin{cases} \tau\xi, & \text{if } \xi \geq 0 \\ (\tau - 1)\xi, & \text{if } \xi < 0 \end{cases} \tag{3.8}$$

where $\mathcal{F}$ is a certain class of functions. A family of conditional quantile functions f_τ at all the continuum of $\tau \in (0, 1)$ provide a full description of the conditional density $p(y \mid \mathbf{x})$.
In Equation (3.8) ℓ_τ is the so-called *pinball loss*. This uses the conditional quantile $\tau \in (0, 1)$ to asymmetrically weight the residual ξ. Accordingly, the relationship between the pinball loss $\ell_\tau(\xi)$ for quantile τ and the residual ξ is illustrated in Figure 3.3.

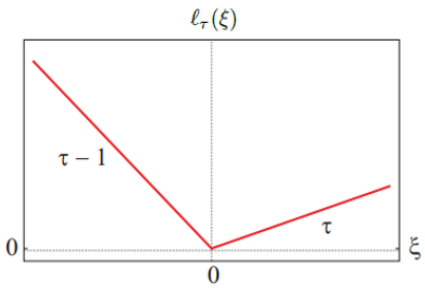


Figure 3.3 – Pinball loss

In [42], authors define the quantile estimator with the following lemma.

Lemma 3.1.1 *Let $\hat{y} = f(\boldsymbol{x_i})$ and let $\tau \in (0, 1)$, the minimizer $\hat{y}_\tau$ of $\sum_{i=1}^{n} \ell_\tau(y_i - \hat{y})$ with respect to $\hat{y}$ satisfies:*

- *The number of samples, $n\_$, with $y_i < \hat{y}_\tau$ is bounded from above by τn*

- *The number of samples, n_+, with $y_i > \hat{y}_\tau$ is bounded from above by $(1-\tau)n$*
- *For $n \to \infty$, the fraction $\frac{n_-}{n_+}$, converges to τ if $Pr(y)$ does not contain discrete components.*

A good quantile estimator $\hat{y}_\tau$ needs to satisfy the quantile property as well as possible:

$$\Pr\{|\Pr\{y < \hat{y}_\tau\} - \tau| \geq \varepsilon\} \leq \delta, \tag{3.9}$$

In other words, the probability that the true values $y = \{y_1, ..., y_n\}$ are smaller than the quantiles estimates shall not deviate from τ by more than ε with high probability. In order to assess the performance of a quantile estimator, we thus look also at the so-called *ramp loss*, i.e. the empirical fraction of observed values which are below the quantile estimates:

$$r_\tau = \frac{1}{n}\sum_{i=1}^{n} I(y_i, \hat{y}_\tau), \quad I(y_i, \hat{y}_\tau) = \begin{cases} 1, & \text{if } y_i < \hat{y}_\tau \\ 0, & \text{otherwise} \end{cases}$$

Ideally, the value of ramp loss should be close to τ.
After some initial works on linear quantile regression, motivated to improve the prediction accuracy as well as the interpretation of the final model, many researchers have considered nonparametric methods. However, the use of conditional estimators increases the model space. To this end, [42] proposes to add a regularization in the optimization function of Equation (3.8), obtaining:

$$\hat{f}_\tau = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell_\tau(y_i - f(\mathbf{x}_i)) + \frac{\lambda}{2}\|g\|_{\mathcal{H}}^2 \tag{3.10}$$

where $f = g + b$, $g \in \mathcal{H}$, $b \in \mathbb{R}$ and $\mathcal{H}$ is the Reproducing Kernel Hilbert Space (RKHS) on $\mathcal{X}$. This minimizer is proved to satisfy the quantile property.
In many situations it is useful to estimate multiple quantile regression functions. However, when we want to estimate several conditional quantiles, two or more estimated conditional quantile functions can cross or overlap. This embarrassing phenomenon called *quantile crossings* occurs because each conditional quantile function is independently estimated. Such a kind of quantile crossing violates the basic principle of distribution functions so that their associated inverse functions should be monotone increasing. Although this phenomenon typically only occurs in outlying regions of the input space when the observations are scarce, it is nevertheless an undesirable phenomenon for utilization and interpretation of these quantile regression functions. For kernel quantile regression, [42] proposes to impose non-crossing constraints on the data points. In particular, using the connection between RKHS and feature spaces, $f(\mathbf{x})$ can be written as $f(\mathbf{x}) = \langle \phi(\mathbf{x}), w \rangle + b$. Then, the non-crossing constraints enforced at l points $\{x_j\}_{j=1}^{l}$ in the input domain $\mathcal{X}$ become linear constraints in $\mathcal{H}$:

$$\langle \phi(x_j), \omega_h \rangle + b_h \leq \langle \phi(x_j), \omega_{h+1} \rangle + b_{h+1}, \text{ for all } 1 \leq h \leq n-1, 1 \leq j \leq l$$

Although the approach can help to reduce the chance of crossing, the dimension of the

optimization problem for simultaneous multiple quantile estimation can be large for certain applications. In [43] authors introduce a stepwise estimation scheme to ensure noncrossing of the regression functions. In particular, given a quantile regression function at a particular quantile τ, constraints are added in the estimation procedure to ensure the next quantile regression function at $\tau + 1$ does not cross the current one. The procedure continues till quantile regression functions at all desired levels are obtained. The work in [44] instead proposes a kernel-based multiple QR estimation technique, namely simultaneous non-crossing quantile regression (SNQR), which uses kernel representations for QR functions and applies constraints on the kernel coefficients to avoid crossing. Recently, in [31] authors introduce a novel approach to estimate multiple quantiles at once. The idea is to estimate the entire conditional distribution of the target variable y, by solving

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{\tau \sim U[0,1]} [\ell_\tau(y_i, f(x_i, \tau))],$$

where ℓ_τ is the pinball loss defined in Equation (3.8). This expression can be minimized using stochastic gradient descent, sampling fresh random quantile levels $\tau \sim U[0, 1]$ for each training point and mini-batch during training. By estimating all quantiles jointly, SQR models the entire conditional distribution of the target variable, establishing a connection with the CDE approach seen in Section 3.1.3. Finally, SQR is empirically shown to alleviate the undesired phenomena of quantile crossings.

Similarly to CDE, quantile regression allows one to estimate aleatoric uncertainty trough the construction of prediction intervals defined in Equation (3.6).

3.2 Uncertainty in Ticket Price Evolution

Once we have seen different methods to estimate the uncertainty around predictions, by modelling the distribution of the target, we will now describe how this can be applied to price prediction, in order to help travellers in dealing with uncertainty in ticket price evolution. To this end, we propose four different approaches. The first one uses a probabilistic model to estimate uncertainty information on the future price. We do this by estimating the CDE of the price. Among the models seen in Section 3.1.3, we choose MDNs for their simplicity and good performance. The second and third approaches instead aim at providing the user with more explicit information to decide when to buy a ticket, by estimating the amount of money that is lost when choosing the wrong date. Finally, the last approach aims at giving to the user financial protection against adverse future price fluctuations. In the last three approaches we compare a standard regression model with quantile regression. However, before going to the details of these approaches, we introduce now the dataset used in the experiments and we provide details of the main machine learning models we compared in the evaluation protocol.

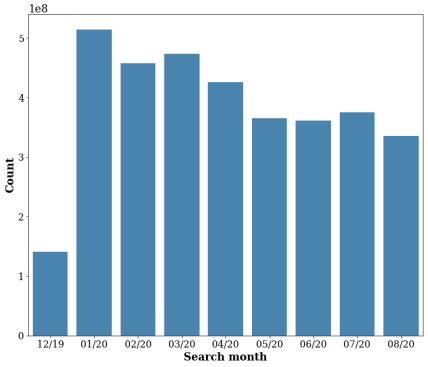


Figure 3.4 – Number of searches per month.

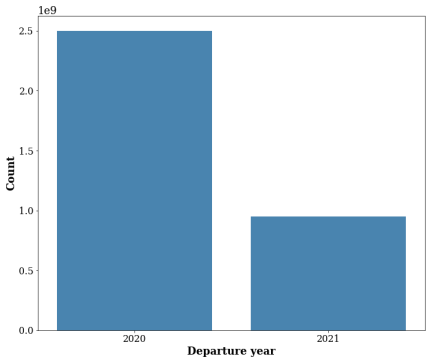


Figure 3.5 – Data distribution over departure year.

3.2.1 Data

The main dataset used in our experiment contains flight searches that are automatically performed by a shooter, over 5-6 hours per day. This leads to more than 3 billion of samples available in the dataset.

Available search data go from December 2019 to August 2020. In Figure 3.4 we show how samples are distributed over the months. These searches, however, relate to flights with departure date ranging from January 2020 to August 2021. We plot the distribution of the data over departure year in Figure 3.5. The difference between the departure date and the search date gives us the time before departure people look at the flight and its corresponding price. We call it DTD, days to departure, and we show its distribution in Figure 3.6. This has a different shape from the one seen in Figure 3.1 (where there is a log scale) coming from MIDT dataset. Here the distribution is much more uniform, in that there is a small difference between the number of searches with DTD around 300 days and the ones with DTD smaller than 60 days. MIDT, instead, contains tickets actually bought, thus Figure 3.1 reflects the real people behavior. Indeed, people do not buy tickets with too much anticipation and they try to avoid high prices. Therefore, if we study ticket price evolution using MIDT we have data biased by this behavior. Therefore we perform our experiments using the dataset containing flight searches.

The dataset includes a total of 529 different origins and destinations (OnDs). However, there is a predominance of German airports and this can be seen in Figure 3.7, where we plot the data distribution over the top 10 OnDs in terms of number of samples. Each flight search is accompanied by a number of attributes. The set of features used in this work are listed in Table 3.1, where the prefix `dep` refers to the outbound flight and the prefix `ret` refers to the return flights. Some of these features come from the raw data, others have been created during the data preprocessing phase.

Train-test split. For our experiments, we adopt two methods of train and test split, keeping a 75:25 split ratio. The first one is based on the search date, so that we use flight searches up to June 2020 as training set and the remaining as test set. This is a realistic approach, in that it avoids to use future data to predict the past. However, by doing this

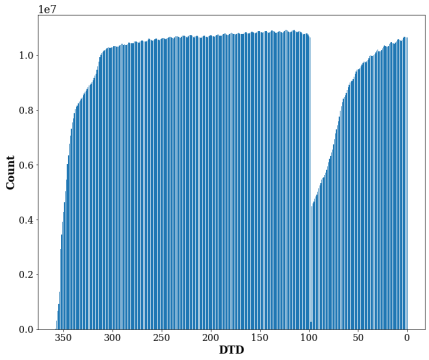


Figure 3.6 – Distribution of days before departure (DTD).

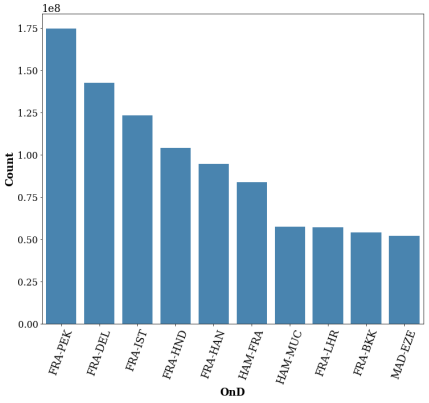


Figure 3.7 – Number of flight searches per OnD, for the top 10 in terms of samples.

`origin`	`destination`	`num_stops`
`dep_day_of_week`	`ret_day_of_week`	`dep_week_of_year`
`ret_week_of_year`	`dep_month`	`ret_month`
`dep_hour`	`ret_hour`	`dep_year`
`ret_year`	`search_month`	`search_year`
`dtd`	`stay`	`price`

Table 3.1 – List of features - GEBECO dataset

we obtain two splits with different generating distributions and this violates the common assumption made in machine learning, according to which data points are realizations of independent and identically distributed (IID) random variables. To handle this, it is indeed common practice to shuffle at random the training and testing examples, which brings the training and testing distributions closer together. We do this also in our experiments.

Target variable. Among the new features that have been added for this work, one which is worth describing is the difference between a flight's price at search time t and $t+\alpha$, where $1 \leq \alpha \leq$`dep_date`, which we call `delta`. This is indeed an important variable to study the price evolution in the period before departure, from which we derive the target variable in our experiments.

In order to add this feature, we proceed in the following way. Given a flight and all the prices observed in the search data, we randomly sample couples of observations and we build a new dataset, where each row contains the flight information, but also a `lag` variable indicating α and a `delta` variable containing the price difference between the sample at time t and the one at time $t+\alpha$. However, to avoid having correlated samples in the dataset, we choose to create rows only with t multiple of α. So for example, if $\alpha = 7$, our dataset contains couples such as $\{p_7 - p_0, p_{14} - p_7, p_{21} - p_{14}, \ldots\}$ instead of $\{p_7 - p_0, p_8 - p_1, p_9 - p_2, \ldots\}$. By doing this, we are sure to use each price in only two pairs and thus we reduce correlations in the data.

3.2.2 Models

In this section we introduce the models we compared in the proposed approaches to help the traveller in deciding the best moment to buy the ticket, giving also the implementation

details.

Random Forest. We use this model for the regression task, consisting in directly predicting the metric that helps travellers in deciding whether to buy or wait. We implement it using Python sklearn library [45] and we select number of trees and maximum depth using grid search with cross-validation.

Mixture Density Networks. As seen in Section 3.1.3, this model performs conditional density estimation, by finding the mixture of Gaussian distributions that model the target variable. In particular, a DNN is used to find the parameters of the GMM, together with their mixture weights. For the experiments, we use a TensorFlow implementation, where we set the number of GMMs to 10. The number of hidden layers and hidden nodes of the neural network are selected with a cross-validation procedure.

Simultaneous Quantile Regression. As seen in Section 3.1.4, this model performs quantile regression by estimating all quantiles at once. The formulation of the problem enables the use of stochastic gradient descent and the possibility to implement it as an additional output layer of any neural network. In our experiments, we use the official implementation provided by [31]. As before, the number of hidden layers and hidden nodes of the neural network, as well as the dropout rate, are selected with a cross validation procedure.

Linear quantile regression. This model estimates the conditional τ-quantile of the response variable, assuming a linear relationship between target and explanatory variables. Unlike the previous model, here quantiles are estimated separately. For our experiments we use the *R* package `quantreg` [46] and we select the independent variables which give the best performance, in terms of pinball loss and ramp loss, by applying a grid search.

Unconditional. This is a very simple model, in that the quantile estimation is a constant function corresponding to the quantile computed on the training set. We call it unconditional, because quantiles are computed using only the target variable and any explanatory variable is taken into account. This kind of model usually performs best in terms of quantile property (Equation (3.9)), but it does not guarantee a good conditional quantile estimation.

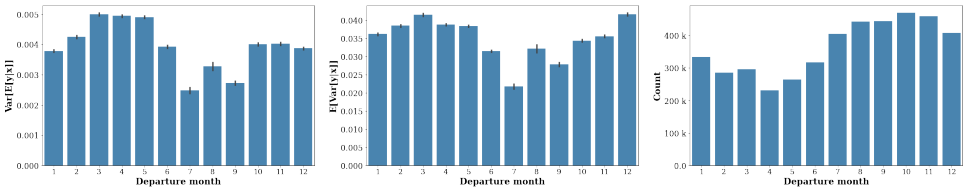
3.2.3 Simple approach: Future price prediction

When thinking about price evolution, the first simple approach one may conceive is to directly predict the future price. However, having a simple point prediction is not enough to take an important decision. Thus we can take a step further and estimate the full distribution of the future price. This allows us to estimate the uncertainty in the predictions, which helps the user in deciding whether to trust the model or not. To do this, we look at the variance of the predictions and using the law of total variance in Equation (3.4), we separate aleatoric and epistemic uncertainty. If for example, we obtain high variance, but low epistemic uncertainty, we can trust the model, because we know that aleatoric uncertainty only depends on the noise inherent in data.

For our experiments, we run a MDN to estimate the conditional distribution of the price in α days, given the flight's information, including today's price. We evaluate the model

using the mean negative log-likelihood (MNLL), which takes values -1.865, -1.985 and -1.676 for $\alpha = 1, 7, 30$ respectively. This says us that the model is able to estimate the price distribution, but results are better for $\alpha = 1$ and $\alpha = 7$, which makes sense, since in those cases the future price is closer to the actual one.

Once assessed the performance of the model, we can use the MDN output to estimate the uncertainty using Equation (3.7). In particular, in Figure 3.8 we show how different each type of uncertainty measure, aleatoric and epistemic, behaves with respect to a given input feature, the departure month in this case. As expected, the aleatoric uncertainty is much larger than the epistemic one, in that predicting the price 30 days later is a complex task, which creates noise in the data, and although the large dataset, this cannot be avoided. Then we observe that the epistemic uncertainty is higher for the months of March, April and May, which correspond to the period where we have less training data, while it is smaller for July and September where, on the contrary, there are more training samples.



(a) Epistemic uncertainty (b) Aleatoric uncertainty (c) Number of training samples

Figure 3.8 – Uncertainty results with respect to departure month, for $\alpha = 30$.

Predicting the price in α days represents a simple approach to deal with uncertainty in ticket price evolution and we have seen that it is possible to estimate uncertainty in the predictions. However, from a traveller point of view, having information about aleatoric or epistemic uncertainty it is not very useful, because he needs some knowledge to understand the results and, even if he has that, this does not help in taking a decision. For this reason, hereafter we propose three other approaches where we help travellers to deal with uncertainty in a more practical way, by providing information on the amount of money they will gain or loose if they wait α days to buy the ticket.

3.2.4 Value at Risk approach

The second approach we propose is based on the Value at Risk (VaR) concept, which is one of the most common measurements of financial risk due to uncertainty, thus playing a central role in investment decisions and banking management. In this context we talk about long position, when the investor is hoping for the price to rise. The typical stock purchase is a long stock asset purchase. A short position is instead the exact opposite of a long position, in that here the investor hopes for, and benefits from, a drop in the price.

Suppose that at the time index t we are interested in the risk of a financial position for the next α periods. Let $\Delta V(\alpha)$ be the change in value of the assets in the financial position from time t to $t + \alpha$. This quantity is a random variable at the time index t. If we denote with $F_\alpha(x)$ the Cumulative Distribution Function (CDF) of $\Delta V(\alpha)$, the VaR of a long

position over the time horizon α with probability τ is defined as

$$\tau = \Pr[\Delta V(\alpha) \leq \text{VaR}] = F_\alpha(\text{VaR}) \tag{3.11}$$

Since the holder of a long financial position suffers a loss when $\Delta V(\alpha) < 0$, the VaR defined in Equation (3.11) typically assumes a negative value when τ is small. The negative sign signifies a loss. From the definition, the probability that the holder would encounter a loss greater than or equal to VaR over the time horizon α is τ. An alternative interpretation of VaR is as follows. With probability $(1-\tau)$, the potential loss encountered by the holder of the financial position over the time horizon α is less than or equal to VaR. The holder of a short position suffers a loss when the value of the asset increases, i.e. $\Delta V(\alpha) > 0$. The VaR is then defined as

$$\tau = \Pr[\Delta V(\alpha) \geq \text{VaR}] = 1 - \Pr[\Delta V(\alpha) \leq \text{VaR}] = 1 - F_\alpha(\text{VaR})$$

For a small τ, the VaR of a short position typically assumes a positive value. The positive sign signifies a loss. The previous definitions show that VaR is concerned with tail behavior of the CDF $F_\alpha(x)$. For a long position, the left tail of $F_\alpha(x)$ is important. Yet a short position focuses on the right tail of $F_\alpha(x)$. Notice that the definition of VaR in Equation (3.11) continues to apply to a short position if one uses the distribution of $-\Delta V(\alpha)$. Therefore, it suffices to discuss methods of VaR calculation using a long position. For any univariate CDF $F_\alpha(x)$ and probability τ, such that $0 < \tau < 1$, the quantity

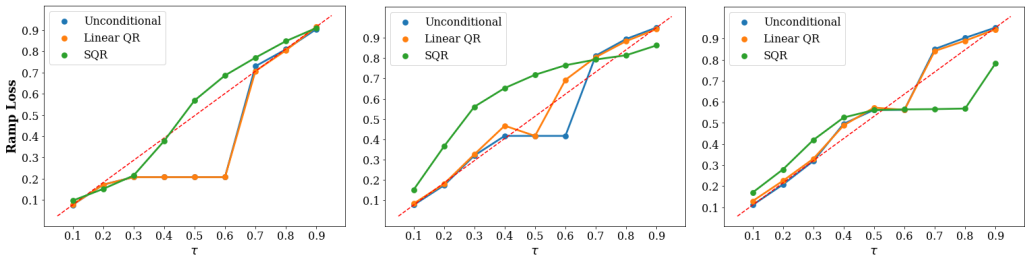
$$x_\tau = \inf\{x \mid F_\alpha(x) \geq \tau\}$$

is called the τth quantile of $F_\alpha(x)$, where inf denotes the smallest real number satisfying $F_\alpha(x) \geq \tau$. If the CDF $F_\alpha(x)$ of Equation (3.11) is known, then VaR is simply its τth quantile (i.e., VaR $= x_\tau$).

The most common approach to VaR is called the variance–covariance methodology, which assumes log-normal returns. However, log returns are frequently found not normally distributed [47], which motivates the employment of a distribution-free approach. The work in [48] hence formulates VaRs based on a conditional autoregressive Value at Risk model (CAViaR) and it estimates them through nonlinear quantile regression, while [49] calculates VaRs estimating a conditional quantile model.

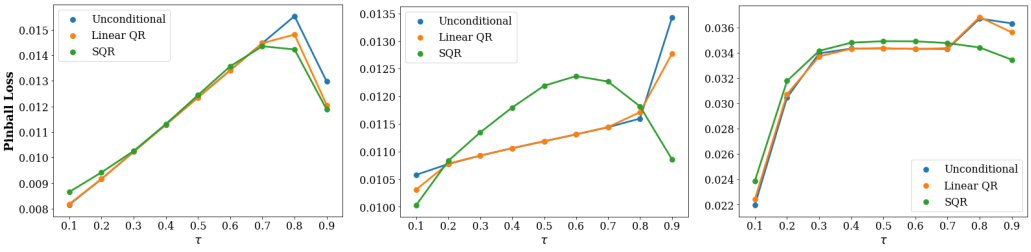
On the traveler side, since he will loose money if the price rises, the VaR is the smallest amount the price will increase with probability $1-\tau$ in a given time period α. Let us consider an example. Assume that a customer searches at time t for a flight with departure at time $t+60$ from a given origin to a give destination. A common search engine will show him a price, but the user does not know if the price will increase or decrease in the 60 days left before departure. Using the VaR measure, we can enrich the search output and give information on the smallest amount of money Δ the user will pay in addition with probability $1-\tau$ at time $t+\alpha$. In this case, people will decide to buy the ticket or wait α days, based on the provided probability and their risk appetite.

Given the definition of VaR as quantile, we estimate it using quantile regression, where the target is represented by the price difference between time t and time $t + \alpha$. We assess the model performance looking at the standard metrics seen in Section 3.1.4, i.e. ramp loss and pinball loss and we compare the three quantile regression models seen in Section 3.2.2, Unconditional, Linear QR, SQR, changing the value of α. Figure 3.9 shows that in general we obtain calibrated predictions, which means the estimated quantiles correspond to the empirical ones. Comparing the three models, their performance vary according to the dataset and so the α value. Beside the ramp loss, we can look at the pinball loss in Figure 3.10. The results show better performance for small quantiles. This is an interesting point, because it means that high probabilities VaRs are more accurate.



(a) $\alpha = 1$ (b) $\alpha = 7$ (c) $\alpha = 30$

Figure 3.9 – Ramp loss results with different values of lag α.



(a) $\alpha = 1$ (b) $\alpha = 7$ (c) $\alpha = 30$

Figure 3.10 – Pinball loss results with different values of lag α.

3.2.5 Regret approach

The third approach we propose to help customers dealing with ticket price uncertainty consists in providing an advice to the traveller on whether he should buy now or in α days. Let us take again an example. At time t a traveller searches for a flight with departure at $t + 60$ and he receives a price from the search engine. In order to help the traveller, we enrich the output with an information suggesting whether it is better to buy at time t or at time $t + \alpha$, where $1 \leq \alpha < 60$. The proposed application is different from the previous one, in that it directly suggests the best moment to buy the ticket, among the two considered time instants, while before the choice was left to the user, based on the probability, the amount of increase and his risk appetite.

Some work related to this already exists in the literature. In [3] authors propose HAMLET, a multi-strategy algorithm which combines reinforcement learning, rule based learning and time series analysis. The predictive rule produces a binary output, corresponding to the advice 'buy' or 'wait', based on a list of attributes of the flight. Despite the promising results reported in the work, the implementation of the method poses computational challenges due to the growing number of routes, not enough historical data being available for each of them. In [22] instead, authors introduce a prediction application which makes use of heterogeneously sampled price series to cluster flights with similar behavior. A supervised learning algorithm is then used to help the customer in deciding when to purchase his ticket, based on the predicted price evolution.

Our work differs from these in that we propose a simple strategy based on the prediction of the financial loss incurring when the ticket purchase is delayed by α days. Besides this, another difference with respect to previous work is that we deal only with fixed lags, i.e. the decision on whether to buy or wait is relative to α and we do not consider the whole period before departure at once.

Our idea is based on the *regret*, which we define as the amount of money that is lost, in relative terms, when relying on the output provided by a given strategy to decide the day when to buy, instead of the hindsight decision. In order to define our notion of regret, we define the following quantities:

$$\Delta = \frac{p_{t+\alpha} - p_t}{p_t} \tag{3.12}$$

$$R_t = \max\left(\frac{p_t - p_{t+\alpha}}{p_{t+\alpha}}, 0\right) = \max\left(\frac{-\Delta}{1+\Delta}, 0\right) \tag{3.13}$$

$$R_{t+\alpha} = \max\left(\frac{p_{t+\alpha} - p_t}{p_t}, 0\right) = \max\left(\Delta, 0\right), \tag{3.14}$$

where Δ is the price difference between time t and $t+a$, normalized by the actual price, R_t is called *regret now* and it measures the amount of money that is lost, in relative terms, if the ticket is bought at time t instead of the true best moment, while we call $R_{t+\alpha}$ *regret later* to measure the financial loss if the purchase is done at time $t+\alpha$, instead of the true best moment.

We then use these quantities to define a specific decision strategy. In particular, we propose an approach based on the expected utility theory applied in the long run. Expected utility theory [50] is a tool used for analyzing situations where individuals must make a decision without knowing which outcomes may result from that decision, i.e., decision making under uncertainty. These individuals will choose the action that will result in the highest expected utility, which is defined as the sum of the products of probability and utility over all possible outcomes. One reason for maximizing expected utility is that it makes for good policy in the long run. This is what the author affirms in [51]. He relies on two mathematical facts about probabilities: the strong and weak laws of large numbers. Both the weak and strong laws of large numbers say, roughly, that over the long run, the average amount of utility gained per trial is overwhelmingly likely to be close to the expected value of an individual

trial.

Applying this concept to our problem, we end up with the decision strategy in Table 3.2. The idea here is to choose the action that minimizes the expected regret that, as already said, represents the amount of money that is lost, in relative terms, if we follow the decision given by the model, instead of buying at the true best moment. Therefore, we estimate the regrets at time t and $t + \alpha$, using Equation (3.13) and Equation (3.14), and we use their expected values to decide whether to buy or wait α days. According to what we just said above, this allows us to minimize the average regret in the long term. In order to implement this strategy, we set a regression problem, where the target variable is $\Delta R = \mathbb{E}[R_t] - \mathbb{E}[R_{t+\alpha}]$. If ΔR is positive, it means that we have a lower regret buying in α days, if instead it is negative it is more convenient buying at time t. In our experiments, we estimate ΔR with Random Forest model. We evaluate the model by computing the average regret over the data samples, which represent several searches.

This approach we just explained to suggest the best moment to buy the ticket between the two proposed ones is very natural and rather simple to implement, because it is sufficient to compute the quantities in Equation (3.13) and Equation (3.14) and run a regression a model. Another approach, which is less direct, consists in looking at the distribution of Δ in Equation (3.12). Given an input probability τ, which depends on the user, he can decide to wait α days if $P(\Delta \leq 0) > \tau$. Denoting with F the CDF of Δ, this can be also written as follows:

$$P(\Delta \leq 0) > \tau \Leftrightarrow F(0) \geq \tau \Leftrightarrow 0 \geq F^{-1}(\tau) = q(\tau),$$

where $q(\tau)$ is the quantile function of Δ. Using this notation, the resulting decision strategy is in Table 3.3.

Regret
if $\mathbb{E}[R_t] > \mathbb{E}[R_{t+\alpha}]$ **then**
wait α *days*
else
buy now
end if

Table 3.2 – Decision strategy with R_t and $R_{t+\alpha}$

Quantile regression
if $q(\tau) < 0$ **then**
wait α *days*
else
buy now
end if

Table 3.3 – Decision strategy with $q(\tau)$

This approach has some limitations with respect to the previous one. Firstly, it significantly depends on the parameter τ, which represents the user's risk appetite, thus providing less generalization than before. Secondly, even if we fix τ, the distribution of Δ plays an important role. If, for example, we have a high probability to have negative values of Δ, let us say $P(\Delta \leq 0) = 0.81$, and consequently a non negligible probability, 0.19, of having positive Δ, if we fix $\tau = 0.8$, the model will suggest us to wait, because $P(\Delta \leq 0) > \tau$. However, if the distribution of Δ is such that the negative values are very small, close to 0, and the positive ones are much larger, by following the model's suggestion, we take the risk (19 % of probability) of loosing a large amount of money, to save a small amount (small negative Δ). To overcome this issue, we should introduce some thresholds on the

	$\alpha = 1$	$\alpha = 7$	$\alpha = 30$
RF	**0.0087**	**0.0076**	**0.0301**
NOW	0.0088	0.0133	0.0447
LATER	0.0177	0.0118	0.0341
RANDOM	0.0132	0.0125	0.0395

Table 3.4 – Expected regret results for different values of lag α, time-based split.

	$\alpha = 1$	$\alpha = 7$	$\alpha = 30$
RF	**0.0092**	**0.0094**	**0.0185**
NOW	0.0111	0.0152	0.0437
LATER	0.0205	0.0153	0.0361
RANDOM	0.0158	0.0152	0.0398

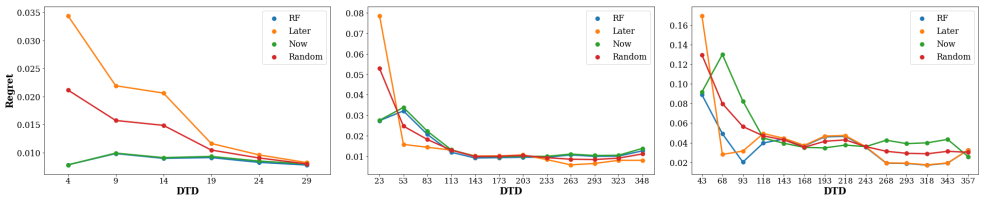
Table 3.5 – Expected regret results for different values of lag α, random split.

gain (negative Δ) and loss (positive Δ). In this case the definition of the decision strategy becomes more complicate and it deserves a deeper study. For this reason, here we limit to present the approach and we leave the implementation of the method for future work.

We thus focus on the regression approach, where we run Random Forest (RF) to predict the quantity ΔR. Beside this, we also compare three naive strategies: one which decides always to buy at time t (NOW), another which suggests always to buy at $t+\alpha$ (LATER) and, finally, a random strategy which alternates between the two choices randomly (RANDOM). Note that the NOW strategy corresponds to the behavior of a standard user which searches for a flight and books it immediately, thinking that the price will only increase afterwards. We evaluate all strategies using the average regret over the test set and we test three values of α. In Table A.1 we show the results using a time-based train-test split, while results in Table A.2 are obtained by adopting a random split. In general, the machine learning approach gives better results than all the naive strategies and this proves the benefit of using the proposed strategy based on the minimization of the expected regret. The difference in performance is very large for $\alpha = 7$ and $\alpha = 30$, while for $\alpha = 1$ the Random Forest regret is very close to the one given by the NOW strategy. Indeed, by running the T-test with a significance level of 0.05, we find that in the latter case, the regret values do not differ significantly between RF and NOW. Finally, the different behavior of the naive strategies with respect to α shows the price instability issue and motivates the introduction of such decision-making tools, able to estimate the best moment to buy a ticket. Regarding, instead, the use of a random train-test split to preserve the IID assumption, we obtain that for $\alpha = 1$ and $\alpha = 7$ this has no effect, while with $\alpha = 30$ RF gives much better results.

Finally, in order to deeply understand the performance of the proposed strategy with respect to the price evolution, we measure the regret on different ranges of days before departure (DTD). We show the results in Figure 3.11, where we represent a range with the corresponding average. Performance are different according to α. When the decision relates whether to buy today or tomorrow ($\alpha = 1$), the output depends strongly on DTD and, as expected, if we are very close to the departure we can save money buying the ticket today, while if there is still one month left it makes no difference having one day of lag. Moreover, in this case the proposed strategy based on the minimization of the expected regret achieves results comparable to the best naive strategy. For $\alpha = 7$ the dependency of the

regret from DTD is relevant only up to around three months before the departure, where adopting only one naive strategy is not enough, but following the decision provided by the machine learning model is convenient as using the NOW strategy. The most interesting case is instead $\alpha = 30$, where it results really crucial to estimate the best moment to buy a ticket, in that prices vary a lot in a 30 days range. In this case again any of naive strategies guarantees stable results, but RF achieves always better or comparable results with respect to the best strategy, specially when we are close the departure date.



(a) $\alpha = 1$ (b) $\alpha = 7$ (c) $\alpha = 30$

Figure 3.11 – Behavior of expected regret with different DTD ranges, represented by their average.

As discussed at the beginning of this section, an alternative approach to the minimization of the expected regret is to use quantile regression to estimate the distribution of Δ and then adopt the resulting strategy in Table 3.3. However, as explained above, this approach has some limitations. While we leave the improvement of such strategy for future work, we put results of the previous experiments using the actual implementation in Appendix A.

3.2.6 Options approach

The last approach we propose to help travellers to deal with uncertainty in ticket price evolution is based on options. A European call option [52] is a contract with the following conditions: the holder of the option has at a prescribed time in the future, the exercise date, the right to buy a prescribed asset for a prescribed amount, the exercise price, the holder of the option is in no way obliged to buy the underlying asset, the exercise price and date are determined at the time when the option is written. For European options the holder has the right to exercise the option only at the exercise date, while an American call option gives the holder the right to exercise the option at any time before the exercise date. An option contract is overpriced if the seller has an edge over the buyer, while it is underpriced if buyer has an expected profit. The Fair Value for an option contract is hence a price at which both buyer and seller have zero expected profit. The expected profit of an option is equal to its all possible values on the expiration date weighted by their probabilities:

$$\mathbb{E}[Pr] = \int V_s \times P_s \times ds$$

where $V_s = \max(0, S - X)$, with S = stock price and X = exercise price, is called *payoff* and P_s is the probability of X to reach S at expiration. In other words, the payoff of an option is zero if one cannot derive any economic benefit out of it and this is the case when

the final stock price is less than the exercise price, otherwise it is equal to the difference between the stock price and the exercise price.

Going back to the ticket price application, we can sell an option at a given time t to buy a ticket at price p_t, but in α days. In this case the payoff of the option corresponds to $\Delta^+ = \max(0, \Delta)$ and the expected profit amounts to:

$$\mathbb{E}[\Delta^+] = \int_{\Delta=-\infty}^{\infty} \max(0, \Delta) dF_{\Delta}(\Delta) \tag{3.15}$$

In order to estimate Equation (3.15), we test two different ways. The first one is to use a supervised learning approach, where given the input features of the search, a regression model directly estimate $\mathbb{E}[\Delta^+]$. We do this using again Random Forest (RF). Nevertheless, it is also possible to estimate the integral in Equation (3.15) using quantile regression in the following way:

$$\begin{aligned} \tau &= F_{\Delta}(\Delta) \\ \Delta &= F_{\Delta}^{-1}(\tau) \equiv q(\tau) \\ \mathbb{E}[\Delta^+] &= \int_{\tau=0}^{1} \max(q(\tau), 0) d\tau \end{aligned} \tag{3.16}$$

In this case we approximate Equation (3.16) with 1000 Monte Carlo samples and we compare two quantile regression models, SQR and Unconditional.

In order to assess the performance of the proposed method we evaluate the mismatch between the estimated expected profit in Equation (3.15) or Equation (3.16) and the actual profit calculated using the true values: $m = \mathbb{E}[\Delta^+] - \Delta^+$. In Table 3.6 and Table 3.7 we show the average mismatch over the test set, using a time-based train-test split and a random one. Results indicate that in general the mismatch between expected and actual profit has small values close to zero, thus the proposed pricing model is fair. Moreover, there is not a real difference between the two approaches, except that since at the end we deal with expected values, the use of a quantile regression model does not bring any additional value with respect to a simple regression model. Finally, Table 3.7 shows that having IID samples leads to optimal results for RF.

	$\alpha = 1$	$\alpha = 7$	$\alpha = 30$
RF	0.007	**0.001**	**0.009**
SQR	**0.001**	0.002	0.020
Unconditional	0.003	0.008	**0.009**

Table 3.6 – Average mismatch (absolute value) for different values of lag α, time-based split.

	$\alpha = 1$	$\alpha = 7$	$\alpha = 30$
RF	**0.000**	**0.000**	**0.000**
SQR	0.003	0.007	0.006
Unconditional	0.002	0.003	0.008

Table 3.7 – Average mismatch (absolute value) for different values of lag α, random split.

3.3 Conclusion

In this chapter we saw how uncertainty plays a crucial role in ticket price evolution. Airline companies use indeed convoluted dynamic pricing strategies to maximize their revenues. At the same time, this strong price instability causes concerns among travellers, which in last years have shown an increasing interest in understanding how prices vary over time, in order to get the best deal. In light of this, we started the chapter describing how different machine learning methods handle uncertainty estimation and then we proposed four different approaches to deal with uncertainty in ticket price evolution.

The first approach we introduced consists in estimating the price in the future. However, instead of providing a point prediction, we estimated the full distribution of the price, using a CDE model. This allowed us to use the predicted variance to quantify the uncertainty related to the noise of the data, i.e. the aleatoric uncertainty, and the one representing the ignorance of the model, i.e. epistemic uncertainty. If this represents the simplest approach one may conceive when talking about price evolution, it requires however some knowledge from the user in order to interpret the results and it makes the traveller's decision difficult about whether he should buy the ticket at that given moment or wait some more days.

For this reason, we proposed three other approaches which help travellers to deal with uncertainty in a more practical way, by considering the choice of buying the ticket today or in α days. The second approach, based on the Value at Risk concept, provides information on the smallest amount of money the user would pay in addition, with a certain probability, if he waits α days. To do this, we used quantile regression and we compared several models. This approach has the advantage of directly providing money information, which may attract those people who really pay attention to money, but at the same time it leaves to them the choice of buying or waiting.

The third approach we proposed, instead, provides directly a suggestion about whether to buy at that given moment or wait α more days. We implemented it using the regret, which we defined as the amount of money that is lost, in relative terms, when relying on the output provided by the model to decide when to buy, instead of the optimal decision in hindsight. Results indicated that when the decision refers to today or tomorrow a naive strategy which suggests always not waiting is enough, otherwise when the lag is equal to 7 or 30 days, using a machine learning approach which minimizes the expected regret allows the user to save money.

Finally, the last approach assumes that today we sell an option to buy a ticket in α days, but with today's price. This indeed would allow the user to buy an option to protect himself against adverse fluctuations of the price. We evaluated the model by estimating the expected profit of the option and comparing it with the true profit. Results indicated that the mismatch between estimated expected profit and actual profit is very small, thus the proposed pricing model is fair. This approach, as the second one, can be more appreciated by those people who are interested in the purely economic aspects.

To sum up, in this chapter we introduced four approaches to deal with uncertainty in ticket price evolution. The choice of using one rather than the other depends on the specific

use-case. However, from a normal traveller point of view, the third approach turns out to be the most practical, in that it provides a clear suggestion about whether to buy the ticket at that given moment or wait some more days.

4

Model Monitoring and Dynamic Model Selection in Travel Time-Series Forecasting

Travel industry actors, such as airlines and hotels, nowadays use sophisticated pricing models to maximize their revenue, which results in highly volatile fares [21]. For customers, price fluctuation are a source of worry due to the uncertainty of future price evolution. This situation has opened the possibility to new businesses, such as travel meta-search engines or online travel agencies, providing decision-making tools to customers [22]. In this context, accurate price forecasting over time is a highly desired feature, as it allows customers to take informed decisions about purchases, and companies to build and offer attractive tour packages, while maximizing their revenue margin.

The exponential growth of computer power along with the availability of large datasets has led to a rapid progress in the machine learning (ML) field over the last decades. This has allowed the travel industry to benefit from the powerful ML machinery to develop and deploy accurate models for price time-series forecasting. Development and deployment, however, only represent the first steps of a ML system's life cycle. Currently, it is the monitoring, maintenance and improvement of complex production-deployed ML systems which carry most of the costs and difficulties in time [4, 53]. Model monitoring refers to the task of constantly tracking a model's performance to determine when it degrades, becoming obsolete. Once a degradation in performance is detected, model maintenance and improvement take place to update the deployed model by rebuilding it, recalibrating it or, more generally, by doing model selection.

While it is relatively easy and fast to develop ML-based methods for accurate price forecasting of different travel products, maintaining a good performance over time faces multiple challenges. Firstly, price forecasting of travel products involves the analysis of multiple time-series which are modeled independently, i.e. a model per series rather than a single model for all. According to the 2019 World Air Transport Statistics report, almost 22K city pairs are directly connected by airlines through regular services [54]. As each city

pair is linked to a time-series, it is impossible to manually monitor the performance of every associated forecasting model. For scalability purposes, it is necessary to develop methods that can continuously and automatically monitor and maintain every deployed model. Secondly, time-series comprise time-evolving complex patterns, non-stationarities or, more generally, distribution changes over time, making forecasting models more prone to deteriorate over time [55]. Poor estimations of a model's degrading performance can lead to business losses, if detected too late, or to unnecessary model updates incurring system maintenance costs [4], if detected too early. Efficient and timely ways to model monitoring are therefore key to continuously accurate in-production forecasts. Finally, a model's degrading performance also implies that the model becomes obsolete. As a result, a specific model might not always be the right choice for a given series. Since time-series forecasting can be addressed through a large set of different approaches, the task of choosing the most suitable forecasting method requires finding systematic ways to carry out model selection efficiently. One of the most common ways to achieve all of this is cross-validation [56]. However, this approach is only valid at development and cannot be used to monitor and maintain models in-production due to the absence of ground truth data.

In this work we introduce a data-driven framework to continuously monitor and maintain time-series forecasting models' performance in-production, i.e in the absence of ground truth, to guarantee continuous accurate performance of travel products price forecasting models. Under a supervised learning approach, we predict the forecasting error of time-series forecasting models over time. We hypothesize that the estimated forecasting error represents a surrogate measure of the model's future performance. As such, we achieve continuous monitoring by using the predicted forecasting error as a measure to detect degrading performance. Simultaneously, the predicted forecasting error enables model maintenance by allowing to rank multiple models based on their predicted performance, i.e. model comparison, and then select the one with the lowest predicted error measure, i.e. model selection. We refer to it as a model monitoring and model selection framework.

The remaining of this chapter is organized as follows. Section 4.1 discusses related work. Section 4.2 reviews the fundamentals of time-series forecasting and performance assessment. Section 4.3 describes the proposed model monitoring and maintenance framework. Section 4.4 describes our datasets and presents the experimental setup. Experiments and results are discussed in section 4.5. Finally, in section 4.6 we summarize our work and discuss key findings.

4.1 Related Work

Maintainable industrial ML systems. Recent works from tech companies [57, 58, 53] have discussed their strategies to deal with some of the so-called *technical debts* [4] in which ML systems can incur when in production. These works mainly focus on the hard- and soft-ware infrastructure used to mitigate these *debts*. Less emphasis is given to the specific methods put in place.

Concept drift. The phenomenon of time-evolving data patterns is known as concept drift.

As time-series are not strictly stationary, it is a common problem of time-series forecasting usually addressed through regular model updates. Most works have focused on its detection, what we denote model monitoring, without performing model selection as they are typically limited to a single model [59, 60]. The exception to this is the work of [61, 62], where a weighted sliding-window is used to combine the forecasts of multiple candidate models into a single value.

Performance assessment without ground truth. An alternative to cross-validation is represented by information criteria. The rationale consists in quantifying the best trade-off between models' goodness of fit and simplicity. Information criteria are mostly used to compare nested models, whereas the comparison of different models requires to compute likelihoods on the same data. Being fully data-driven, our framework avoids any constraint regarding the candidate models, leading to a more general way to perform model selection. Specifically to time-series forecasting, Wagenmakers et al. [63] achieve performance assessment in the absence of ground truth using a concept similar to ours. They estimate the forecasting error of a new single data point by adding previously estimated forecast errors, obtained from already observed data points. The use of the previous errors makes it sensible to unexpected outlier behaviors of the time-series.

Meta-learning. Meta-learning has been proposed as a way to automatically perform model selection. Its performance has been recently demonstrated in the context of time-series forecasting. Both [64, 65] formulate the problem as a supervised learning one, where the meta-learner receives a time-series and outputs the "best" forecasting model. Authors in [66] share our idea that forecasting performance decays in time, thus they train a meta-learner to model the error incurred by the base models at each prediction step as a function of the time-series features. Differently from [65], our approach does not seek to select a different model family for each time-series, and avoids model selection at each time step [66], since these two represent expensive overheads for in-production maintenance. Instead, we maintain a fast forecasting procedure and select the best model for a given time period in the future, which length can be relatively high (6-9 months, for instance).

4.2 Time-series forecasting and performance measures

A univariate time-series is a series of data points $\mathcal{T} = \{y_1, \ldots, y_T\}$, each one being an observation of a process measured at a specific time t. Univariate time-series contain a single variable at each time instant, while multivariate time-series record more than one variable at a time. Our application is concerned with univariate time-series, which are recorded at discrete points in time, e.g., monthly, daily, hourly. However, extension to the multivariate setting is straightforward.

Time-series forecasting is the task consisting in the use of these past observations (or a subset thereof) to predict future values $\mathcal{T}_h = \{\hat{y}_{T+1}, \ldots, \hat{y}_{T+h}\}$, with h indicating the forecasting horizon. The number of well-established methods to perform time-series forecasting is quite large. Methods go from classical statistical methods, such as Autoregressive Moving Average (ARMA) and Exponential smoothing, to more recent machine learning models

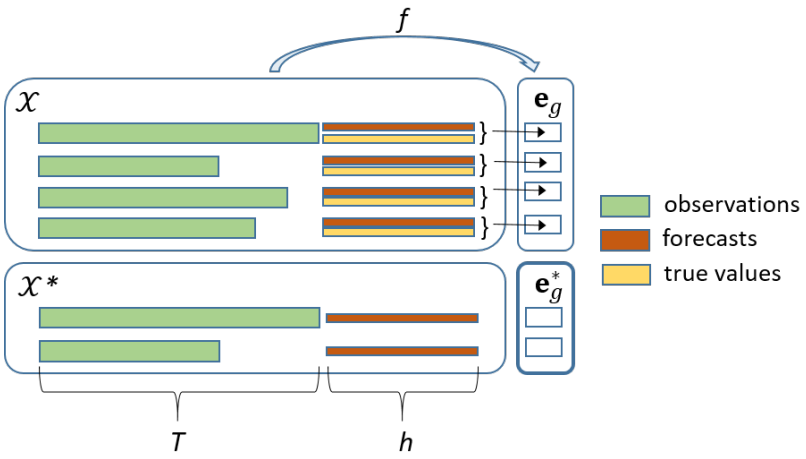


Figure 4.1 – Illustration of the proposed method. $\mathcal{X}$ and $\mathcal{X}^*$ contain multiple time-series, each of these composed of T_i observations (green) and h forecasts (red) estimated by a *monitored model*, g. $\mathbf{e}_g$ represents the forecasting performance of the *monitored model*. It is computed using the true values (yellow). A *monitoring model* is trained to learn the function f mapping $\mathcal{X}$ to $\mathbf{e}_g$. With the learned f, the *monitoring model* is able to predict $\mathbf{e}^*_g$, the predicted forecasting performance of the *monitored model* given $\mathcal{X}^*$.

which have shown outstanding performance in different tasks, including time-series forecasting.

The performance assessment of forecasting methods is commonly done using error measures. Despite decades of research on the topic, there is still not an unanimous consensus on the best error measure to use among the multiple available options [67]. Among the most used ones, we find Symmetric Mean Absolute Percentage Error (sMAPE) and Mean Absolute Scaled Error (MASE). These two have been adopted in recent time-series forecasting competitions [68].

4.3 Monitoring and model selection framework

Let us denote $\mathcal{X} = \{\mathcal{T}^{(i)}, \mathcal{T}_h^{(i)}\}_{i=1}^N$ the input training set. A given input i is formed by the observed time-series $\mathcal{T}^{(i)}$ and h forecasted values, $\mathcal{T}_h^{(i)}$. The values in $\mathcal{T}_h^{(i)}$ are obtained by a given forecasting model which we hereby denote a *monitored model*, g. Let $\mathbf{e}_g = \{e_g^{(i)}\}_{i=1}^N$ be a collection of N performance measures assessing the accuracy of the forecasts $\mathcal{T}_h^{(i)}$ estimated by g. A given performance measure $e_g^{(i)}$ is obtained by comparing the forecasts $\mathcal{T}_h^{(i)}$ from g to the true values.

Lets define a *monitoring model* as a model that is trained to learn a function f mapping the input time-series $\mathcal{X}$ to the target $\mathbf{e}_g$. Given a new set of time-series $\mathcal{X}^* = \{\mathcal{T}^{*(i)}, \mathcal{T}^{*(i)}_h\}_{i=1}^{N^*}$, formed by a time-series of observations $\mathcal{T}^{*(i)}$, $|\mathcal{T}^{*(i)}| = T_i^*$, and h forecasts $\mathcal{T}^{*(i)}_h$ obtained by g, the learned *monitoring model* predicts $\mathbf{e}^*_g$, i.e. the predicted performance measure of g given $\mathcal{X}^*$ (Figure 4.1).

The predicted performance measures $\mathbf{e}^*_g$ represent a surrogate measure of the performance

of a given g within the forecasting horizon h. As such it is used for the two tasks: model monitoring and selection. Model monitoring is achieved by using $\mathbf{e}_g^*$ as an alert signal. If the estimated performance measure of the *monitored model* is poor, this means the model has become stale. To achieve model selection, $\mathbf{e}^*$ are used to rank multiple *monitored models* and choose the one with the best performance If the two tasks are executed in a continuous fashion over time, it is possible to guarantee accurate forecasts in an automated way.
In the following, we describe the performance measure $\mathbf{e}$ that we use in our framework, as well as the *monitoring* and *monitored models* that we chose to validate our hypotheses.

4.3.1 Performance measure

As previously discussed, performance accuracy of time-series forecasts is measured using error metrics. In this framework, we use the sMAPE. It is defined as:

$$\text{sMAPE} = \frac{1}{h}\sum_{t=1}^{h} 2\frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|}, \tag{4.1}$$

where h is the number of forecasts (i.e. forecasting horizon), y is the true value and $\hat{y}$ is the forecast.
In the literature, there are multiple definitions of the sMAPE. We choose the one introduced in [69] because it is bounded between 0 and 2; specifically, it has a maximum value of 2 when either y or $\hat{y}$ is zero, and it is zero when the two values are identical. The sMAPE has two important drawbacks: it is undefined when both y, $\hat{y}$ are zero and it can be numerically unstable when the denominator in Eq. 4.1 is close to zero. In the context of our application, this is not a problem since it is unlikely to have prices with value zero or very close to it.

4.3.2 Monitoring models

The formulation of our framework is generic in the sense that any supervised technique that can solve regression problems can be used as a *monitoring model*. In this work, we decided to focus on latest advances in deep learning. We consider four alternative *monitoring models*: Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), Bayesian CNNs and Gaussian Processes (GPs). The latter two models differ from the former ones in that they also provide uncertainties around the predictions. This can enrich the output provided by the monitoring framework, in that whenever an alert is issued because of poor performance, this is equipped with information about its reliability This section illustrates the basic ideas of each of the selected *monitoring models*.

Long Short-Term Memory networks. LSTM [70] networks are a type of Recurrent Neural Networks (RNNs) that solve the issue of the vanishing gradient problem [71] present in the original RNN formulation. They achieve this by introducing a cell state into each hidden unit, which memorizes information. As RNNs they are a well-established architecture to model sequential data. By construction, LSTMs can handle sequences of varying

length, with no need for extra processing like padding. This is useful in our application, whereby time-series in the datasets have different lengths.

Convolutional Neural Networks. CNNs [72] are particular class of deep neural networks where the weights (filters) are designed to promote local information to propagate from the input to the output at increasing levels of granularity. We use the original LeNet [73] architecture, as it obtains generally good results in image recognition problems, while being considerably faster to train with respect to more modern architectures. CNNs are not originally conceived to work with time-series data. We adapt the architecture to work with time-series by using 1D convolutional filters. Unlike RNNs, this model does not support inputs of variable size, so we to resort to padding: where necessary we append zeros to a time-series to make them uniform in length. We denote this model LeNet.

Bayesian Convolutional Neural Networks. Bayesian CNNs [74] represent the probabilistic version of CNNs, used in applications where quantification of the uncertainty in predictions is needed. Network parameters are assigned a prior distribution and then inferred using Bayesian inference techniques. Due to the intractability of the problem, the use of approximations is required. Here we choose Monte Carlo Dropout [74] as a practical way to carry out approximate Bayesian CNNs. By applying dropout at test time we are able to sample from an approximate posterior distribution over the network weights. We use this technique on the LeNet CNN with 1D filters to produce probabilistic outputs. We denote this model Bayes-LeNet.

Gaussian processes. GPs [32] form the basis of probabilistic nonparametric models. Given a supervised learning problem, GPs consider an infinite set of functions mapping input to output. These functions are defined as random variables with a joint Gaussian distribution, specified by a mean function and a covariance function, the latter encoding properties of the functions with respect to the input. One of the strengths of GP models is the ability to characterize uncertainty regardless of the size of the data. Similarly to CNNs, in this model input sequences must have the same length, so we resort to padding.

4.3.3 Monitored models

Similar to *monitoring models*, given the generic nature of the proposed framework, there is no constraint on the type of *monitored models* that can be used. Any time-series forecasting method can be monitored. For this proof of concept, we consider six different *monitored models*. We select five of them from the ten benchmarks provided in the M4 competition [68], a recent time-series forecasting challenge. These are: Simple Exponential Smoothing (`ses`), Holt's Exponential Smoothing (`holt`), Dampen Exponential Smoothing (`damp`), Theta (`theta`) and a combination of `ses` - `holt` - `damp` (`comb`). Besides these five methods, we included a simple Random Forest (`rf`) , in order to enrich the benchmark with a machine learning-based model. We refer the reader to [75, 68] for further details on each of these approaches.

4.4 Experimental setup

This section presents the data, provides details about the implementation of our methods to ease reproducibility and concludes by describing the evaluation protocol carried during the experiments.

4.4.1 Data

Flights and hotels datasets. We focus on two travel products: direct flights between city pairs and hotels. Our data is an extract of prices for these two travel products obtained from the Amadeus for Developers API [1], an online web-service which enables access to travel-related data through several APIs. It was collected over a two-years and one-month period. Table 4.1 presents some descriptive features of the datasets.
Using the service's Flight Low-fare Search API, we collected daily data for one-way flight prices of the top 15K most popular city pairs worldwide. The collection was done in two stages. A first batch, corresponding to the top 1.4K pairs (FLIGHTS), was gathered for the whole collection period. The second batch, corresponding to the remaining pairs (FLIGHTS-EXT), was collected only over the second year. For hotels, we used the Hotel API to collect daily hotel prices for a two-night stay at every destination city contained in the top city pairs used for flight search. These represent 3.2K different time-series.
Both APIs provide information about the available offers for flights/hotels, that meet the search criteria (origin-destination and date, for flights; city, date and number of nights, fixed to 2, for hotels) at the time of search. As such, it is possible to have multiple offers (flights or hotel rooms) for a given search criteria. When multiple offers were proposed, we averaged the different prices to have a daily average flight price for a given city pair, in the case of flights, or daily average hotel price for a given city, in the case of hotels. In the same way, it is possible to have no offers for a given search criteria. Days with no available offers were reported as missing data. Lack of offers can be caused by sold outs, specific flight schedules (e.g. no daily flights for a city pair) or seasonal patterns (e.g. flights for a part of the year or seasonal hotel closures). More rarely, they could even be due to a failure in the query sent to the API. As a result, the number of available observations is smaller than the length of the collection period (see Table 4.1).

Public benchmarks. In addition to travel products data, we decided to include data coming from publicly available benchmarks. Benchmark data are typically curated and avoid problems present in real data, such as those previously discussed regarding missing data, allowing for an objective assessment and more controlled setup for experimentation. We included two sets from the M4 time-series forecasting challenge competition [68] dataset, YEARLY and WEEKLY. Table 4.1 presents statistics on the number of time-series and the available number observations per time-series for these two datasets. Here, the number of available observations is equivalent to the time-series length as no time-series contains

[1] `https://developers.amadeus.com/`

Table 4.1 – Information about number of time-series, and minimum (min-obs), maximum (max-obs), mean (mean-obs) and standard deviation (std-obs) of the available number of time-series observations per dataset.

Name	# time-series	min-obs	max-obs	mean-obs	std-obs
FLIGHTS	1,415	431	745	734	23
FLIGHTS-EXT	13,810	50	347	346	13
HOTELS	3,207	1	658	368	128
YEARLY	23,000	13	835	31	25
WEEKLY	359	80	2,597	1022	706

missing values.

4.4.2 Implementation

The LSTM network was implemented in Tensorflow. It is composed of one hidden layer with 32 hidden nodes. It is a dynamic LSTM, in that it allows the input sequences to have variable lengths, by dynamically creating the graph during execution. The two CNN-based *monitoring models* use the LeNet architecture. We modified both convolutional and pooling layers with 1D filters, given that the input of the model consists in sequences of one dimension. We added dropout layers to limit overfitting. In the Bayesian CNN, we applied a dropout rate of 0.5, also at testing time, to obtain 100 Monte Carlo samples as approximation of the true posterior distribution. The GP model used the implementation of Sparse GP Regression from the GPy library[2]. The inducing points [76] were initialized with K-means and were then fixed during optimization. We used a variable number of inducing points depending on the size of the input and a RBF kernel with Automatic Relevance Determination (ARD). In all experiments we used 75% data for training and 25% for test and the Adam optimizer with default learning rate [77]. Only in the dataset FLIGHTS-EXT we used mini-batches of size 128 to speed up the training. For the *monitored models*, we used the implementation available from the M4 competition benchmark Github repository[3] and we used the Python sklearn package [45] implementation of R andom Forest. All code has been made publicly available[4].

4.4.3 Evaluation protocol

For flight and hotel data we set $h = \{90, 180\}$, which means we are predicting the price for h days ahead. These are two commonly used values in travel, representing 3 and 6 months ahead of the planned trip, so it is important to have accurate predictions over those horizons. For the M4 competition datasets, we use the horizon given by the challenge organizers: $h = 6$ for YEARLY and $h = 13$ for WEEKLY. For each dataset, we reserve the first T_i data points of the *i-th* time-series, where T_i depends on the time-series's length, as input of the *monitored models* to obtain h forecasts. Where missing values were found, in flights or hotels, these were replaced with the nearest non-missing value in the past. We

[2] `http://github.com/SheffieldML/GPy`
[3] `https://github.com/M4Competition/M4-methods`
[4] `https://github.com/robustml-eurecom/model_monitoring_selection`

build $\mathcal{X}$ and $\mathcal{X}^*$, by taking 75% and 25% from the total number of time-series, respectively. We thus compute the forecasting errors $\mathbf{e}$ using the sMAPE in Eq. 4.1 for the training set $\mathcal{X}$. Finally, we predict the performance measure $\mathbf{e}^*$ for the time-series in $\mathcal{X}^*$, using the four *monitoring models*.

We compare our model monitoring and selection framework with the standard cross-validation method, which we here denote `baseline`, where a model's estimated performance is obtained "offline" at training time with the available data. Specifically, given T observations, we use the last h observations as validation set to evaluate the model. This implies to reduce the number of observations available to train the forecasting models, which can be problematic when either T is small or h is large.

4.5 Experiments and Results

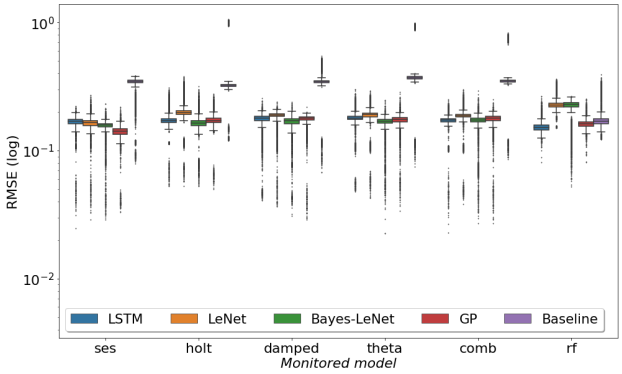
We first study the proposed framework's ability to achieve model monitoring (Sec 4.5.1). Then, we demonstrate how the predicted forecasting errors can be used to carry out model selection and how it positions w.r.t state-of-the-art methods doing the same task (Sec 4.5.2). In Sec 4.5.3, we illustrate the performance of the joint model monitoring and selection framework in our target application.

4.5.1 Model monitoring performance

We evaluate if the *monitoring models*' predicted sMAPEs can be used for model monitoring by estimating if the predicted measure represents a good estimate of a *monitored model*'s future forecasting performance. We assess the quality of the predicted forecasting errors by estimating the Root Mean Squared Error (RMSE) between the predicted sMAPEs and the true sMAPEs, for every *monitored model*. The true sMAPE is obtained using the *monitored model*'s predictions and the time-series' observations in through Eq. 4.1. As a reference, we report the `baseline` RMSE, which is obtained by comparing the estimated sMAPE at training with the observed values at testing.Figure 4.2 left summarizes the obtained results on all datasets.

The overall average error incurred by the *monitoring models* is low. This suggests that the forecasting error predictions are accurate, meaning that it is reliable to carry out model monitoring. When compared to it, the *monitoring models* consistently perform better than standard cross-validation when estimating the future performance of the forecasting *monitored models*. There is an exception to this when the *monitored model* is the Random Forests (`rf`). In this case, the `baseline` is not the worst performing approach. However, it is still surpassed in performance by both LSTM and GPs.

Figure 4.3 details the results obtained for flights and hotels time-series. Table 4.2 stratifies the results for travel product time-series in terms of the forecasting horizon. results show that Bayes-LeNet obtains the lowest RMSEs, whereas GPs follows closely and reports lower standard deviation. Overall, our approach outperforms the `baseline` for large forecasting horizons, e.g. $h = 180$, while the methods get closer as the forecasting horizon decreases.



!h

Figure 4.2 – RMSE between predicted and measured forecasting error (sMAPE) on all datasets (log scale). The reported baseline RMSE is obtained by comparing the estimated sMAPE at training with the observed values at testing.

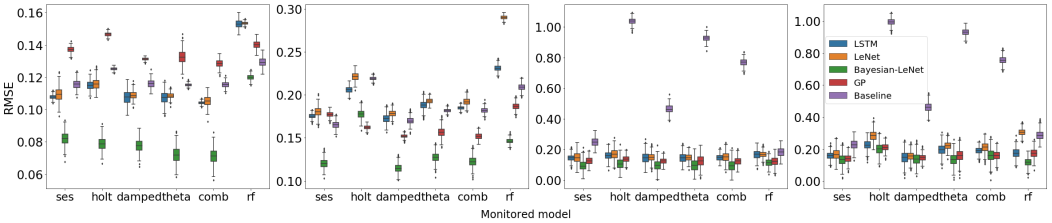


Figure 4.3 – RMSE between predicted and measured forecasting error (sMAPE). From left to right FLIGHTS and FLIGHTS-EXT (top) with 1) $h = 90$, 2) $h = 180$, hotels 3) $h = 90$, 4) $h = 180$.

This is consistent with our hypothesis that data properties change over time. Using a validation set composed of time points close to the unseen data gives consistent information about the model's performance, because the two sets of data (validation and unseen data) have similar properties. However, increasing h has the effect of pushing away the validation time points from the unseen data. In this case, it is better to rely on the forecast error prediction rather than on an error measure obtained during training.

4.5.2 Model selection performance

In this experiment, we assess the capacity of the proposed method to assist model selection in the absence of ground truth. *Monitored models* are ranked by estimating the average predicted sMAPE over a given time-series and ordering the resulting values in ascending order. In this way, we obtain a list of *monitored models* from the best to the worst one. The best performing *monitored model* is selected. We compare the ground truth ranking with the one obtained by each of the *monitoring models* and the `baseline`. We apply a Wilcoxon test [78] to the ranking results to verify if there are significant differences between each of the ranked *monitored models*. Table 4.3 presents obtained results in hotels and flights.
Overall, the obtained rankings are consistent with the ground truth, proving the ability of the method to carry out model selection, by identifying the model with the lowest error measure. Moreover, comparing our approach with the `baseline`, we find that our framework largely outperforms the latter, in that the ranking resulting from the `baseline` is very

Table 4.2 – RMSE between predicted and true sMAPEs for flights and hotel time-series.

Monitoring model	Flights		Hotels	
	$h = 90$	$h = 180$	$h = 90$	$h = 180$
LSTM	0.116 ± 0.017	0.151 ± 0.031	0.193 ± 0.021	0.182 ± 0.039
LeNet	0.117 ± 0.017	0.155 ± 0.031	0.209 ± 0.039	0.224 ± 0.062
Bayes-LeNet	**0.084 ± 0.017**	**0.100 ± 0.035**	**0.135 ± 0.022**	**0.148 ± 0.044**
GP	0.136 ± 0.007	0.126 ± 0.028	0.164 ± 0.014	0.165 ± 0.036
`baseline`	0.119 ± 0.006	0.604 ± 0.328	0.190 ± 0.020	0.609 ± 0.302

Table 4.3 – Comparison between true and predicted model rankings, in ascending order of sMAPE. Underlined values indicate pairs of forecasting models not significantly different, according to Wilcoxon test.

	Ground Truth		Monitoring models								Baseline	
			LSTM		LeNet		Bayes-LeNet		GPs			
HOTELS - $h = 180$												
	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE
1	damp	0.244 (0.153)	ses	208 (0.015)	ses	0.208 (0.032)	ses	0.212 (0.087)	damp	0.230 (0.119)	ses	0.326 (0.202)
2	ses	0.246 (0.164)	damp	0.220 (0.033)	damp	0.211 (0.056)	damp	0.224 (0.130)	ses	0.231 (0.121)	rf	0.413 (0.333)
3	theta	0.269 (0.217)	theta	0.233 (0.059)	theta	0.231 (0.024)	comb	0.249 (0.166)	comb	0.251 (0.149)	damp	0.462 (0.391)
4	comb	0.270 (0.207)	comb	0.234 (0.057)	rf	0.236 (0.047)	theta	0.268 (0.234)	theta	0.252 (0.160)	comb	0.746 (0.569)
5	rf	0.316 (0.300)	holt	0.280 (0.124)	comb	0.278 (0.145)	rf	0.324 (0.329)	rf	0.291 (0.207)	theta	0.938 (0.620)
6	holt	0.325 (0.277)	rf	0.292 (0.210)	holt	0.298 (0.189)	holt	0.325 (0.162)	holt	0.299 (0.190)	holt	1.047 (0.660)
HOTELS - $h = 90$												
	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE
1	damp	0.242 (0.175)	ses	0.203 (0.022)	ses	0.217 (0.065)	ses	0.238 (0.088)	damp	0.221 (0.137)	comb	0.237 (0.166)
2	ses	0.243 (0.174)	damp	0.218 (0.026)	damp	0.223 (0.073)	damp	0.239 (0.122)	comb	0.238 (0.155)	ses	0.239 (0.177)
3	comb	0.253 (0.189)	theta	0.223 (0.022)	theta	0.227 (0.063)	comb	0.259 (0.108)	theta	0.240 (0.151)	damp	0.250 (0.194)
4	theta	0.254 (0.190)	comb	0.224 (0.030)	comb	0.229 (0.047)	theta	0.263 (0.132)	ses	0.244 (0.180)	theta	0.251 (0.201)
5	holt	0.275 (0.217)	holt	0.244 (0.052)	holt	0.252 (0.096)	holt	0.282 (0.190)	holt	0.265 (0.185)	holt	0.277 (0.235)
6	rf	0.293 (0.285)	rf	0.254 (0.103)	rf	0.263 (0.059)	rf	0.298 (0.176)	rf	0.266 (0.191)	rf	0.311 (0.296)
FLIGHTS - $h = 180$												
	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE
1	rf	0.238 (0.163)	rf	0.203 (0.007)	rf	0.199 (0.039)	rf	0.219 (0.075)	rf	0.213 (0.108)	rf	0.259 (0.200)
2	ses	0.247 (0.144)	theta	0.217 (0.026)	ses	0.215 (0.012)	theta	0.220 (0.097)	theta	0.226 (0.098)	damp	0.277 (0.150)
3	theta	0.248 (0.175)	ses	0.218 (0.024)	damp	0.216 (0.074)	ses	0.233 (0.098)	ses	0.227 (0.100)	ses	0.278 (0.151)
4	damp	0.249 (0.144)	damp	0.219 (0.022)	theta	0.217 (0.042)	damp	0.240 (0.090)	damp	0.229 (0.098)	theta	0.281 (0.155)
5	comb	0.250 (0.148)	comb	0.221 (0.027)	comb	0.219 (0.016)	comb	0.241 (0.094)	comb	0.231 (0.054)	comb	0.283 (0.160)
6	holt	0.260 (0.162)	holt	0.223 (0.034)	holt	0.222 (0.036)	holt	0.250 (0.088)	holt	0.238 (0.119)	holt	0.299 (0.199)
FLIGHTS - $h = 90$												
	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE	model	sMAPE
1	comb	0.174 (0.102)	comb	0.154 (0.081)	theta	0.160 (0.067)	comb	0.151 (0.086)	damp	0.159 (0.073)	ses	0.187 (0.110)
2	damp	0.175 (0.106)	damp	0.155 (0.076)	comb	0.164 (0.085)	damp	0.161 (0.086)	comb	0.160 (0.082)	theta	0.188 (0.109)
3	theta	0.176 (0.105)	theta	0.157 (0.042)	holt	0.166 (0.076)	theta	0.163 (0.087)	theta	0.162 (0.086)	damp	0.189 (0.110)
4	ses	0.177 (0.106)	ses	0.158 (0.028)	rf	0.174 (0.066)	holt	0.188 (0.074)	ses	0.163 (0.094)	comb	0.190 (0.112)
5	holt	0.179 (0.113)	holt	0.159 (0.036)	ses	0.183 (0.087)	ses	0.212 (0.070)	holt	0.171 (0.119)	holt	0.195 (0.118)
6	rf	0.232 (0.150)	rf	0.200 (0.025)	damp	0.212 (0.044)	rf	0.287 (0.083)	rf	0.210 (0.094)	rf	0.207 (0.137)

different from the true one. Even in predictions with a small forecasting horizon ($h = 90$), the `baseline`'s ranking performance remains sub-optimal . Looking at the four *monitoring models*, we find that they have a different behavior depending on the dataset. Specifically, GPs result to be slightly more reliable than Bayesian-LeNet, as the latter in some cases swapped the first and second model of the ranking. LSTM's performance is close to the two probabilistic models, although the latter two globally have a better performance in terms of RMSE (see Table 4.2).

Having showed the reliability of the rankings, we evaluate if these can be effectively used to maintain accurate forecasts over time by doing model selection at fixed periods of time. Specifically, given a forecasting horizon, we divide it in smaller periods. At each time point, we use the predicted forecasting error to rank the *monitored models* and thus perform model selection by picking the best ranked model. We use the public benchmark data to guarantee curated data and we limit the experiments to the best two *monitoring models*, Bayesian-LeNet and GPs (Table 4.2). We compare our model selection with the results obtained using the same *monitored model* along the forecasting horizon. Figure 4.4 left shows the average forecasting performance, measured through the real sMAPE, on the WEEKLY dataset. The

proposed model selection scheme allows to have the lower forecasting errors, i.e. a better performance, along the whole forecasting horizon. Among the two *monitoring models*, GPs result in smoother curves.

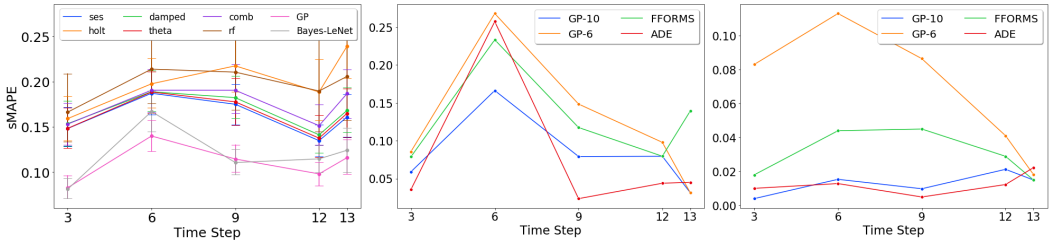


Figure 4.4 – Measured average forecasting performance (sMAPE) using the proposed method for model selection in the WEEKLY dataset with fixed forecasting models over the whole horizon. Average performance with Bayes-LeNet and GPs as *monitoring models* (left). Error bars denote standard deviation. Using GPs as *monitoring model* with six (GP-6) and ten *monitored models* (GP-10), worst (center) and best (right) model selection performances in comparison with `ADE` and `FFORMS`.

Finally, we compare with two state-of-the-art meta-learning methods, arbitrated dynamic ensembler [66], `ADE`, and Feature-based FORecast-Model Selection [65], `FFORMS`, with the best performing *monitoring model* in our approach. The characteristics of these two methods allows them to be used to achieve good forecasting model's performance. `FFORMS` uses 12 different base models, whereas `ADE` uses up to 40 different models. To remain competitive with these two methods that use a larger number of base models, we add three standard forecasting models, Arima (`arima`), Random Walk (`rwf`) and TBATS (`tbats`) [79], and a feed-forward neural network (`nn`), to our set of *monitored models*. We present sMAPE results over two time-series from the WEEKLY dataset: one where our method performs worst (Fig. 4.4 center) and the one where it performs best (Fig. 4.4 right). We show the results of our approach using the original six *monitored models* and the enlarged set. Using the original six *monitored models*, our performance is worse than the two meta-learning models. However, by enlarging the set of *monitored models*, our method performs better than `FFORMS` and achieves a performance comparable to `ADE` with much less monitored/base models.

4.5.3 Model monitoring and selection performance

Finally, we illustrate the performance of the proposed model monitoring and selection framework by using it to guarantee continuous price forecasting accuracy of our two travel products: flights and hotels. In this context, the predicted sMAPE is used as a surrogate measure of the quality of the forecasts estimated by the *monitored models*. When the predicted sMAPE surpasses a given threshold, model selection is performed. Otherwise, the *monitored model* is kept. We use the best performing *monitoring model*, GPs. Since this is a probabilistic method, in addition to having a high predicted sMAPE, we add the condition of having a low uncertainty in the prediction. In our experiments, we set the sMAPE threshold at 0.02 for flights and 0.01 for hotels. The uncertainty was set at 0.01

for both. For this experiment, we removed `rf` from the *monitored models* pool as it is the method giving the poorest performance. It is important to remark that differently from other approaches removing a method from the *monitored models* pool simply requires to stop generating forecasts with the removed model. No re-training of the *monitoring models* is required.

Figure 4.5 illustrates the results obtained in terms of the average performance (sMAPE) for HOTELS with forecasting horizon $h = 90$. Our experiment here is quite restrictive, in the sense that no *monitored model* is re-trained along the forecasting period. In this way, we show that even under this restrictive setting the proposed framework is able to improve the performance of simple models. This suggests that through the use of this framework it is possible to extend the moment where *monitored models* need to be re-trained by simply using the ranking information to pick a new model. Delaying model re-training represents important cost savings.

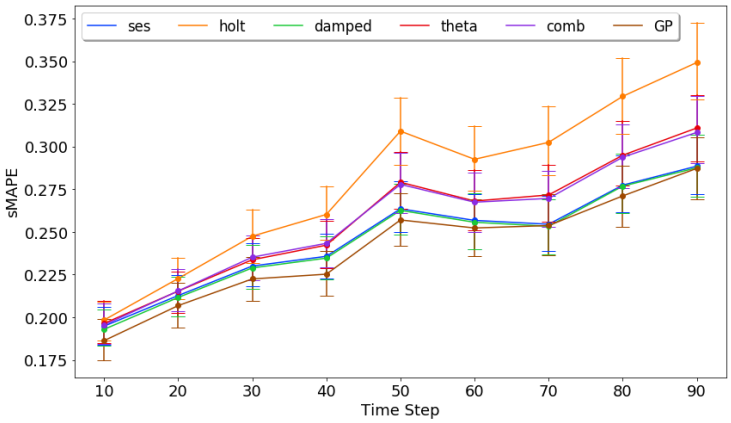


Figure 4.5 – Average forecasting performance in terms of sMAPE using the proposed model monitoring and selection framework (GPs as *monitoring model*) and using forecasting fixed models over the whole horizon. Error bars denote the standard deviation.

4.6 Conclusion

In this chapter we introduce a data-driven framework to constantly monitor and compare the performance of deployed time-series forecasting models to guarantee accurate forecasts of travel products' prices over time. The proposed approach predicts the forecasting error of a forecasting model and considers it as a surrogate of the model's future performance. The estimated forecasting error is hence used to detect accuracy deterioration over time, but also to compare the performance of different models and carry out dynamic model selection by simply ranking the different forecasting models based on the predicted error measure and selecting the best. In this work, we have chosen to use the sMAPE as forecasting performance measure, since it is appropriate for our application but, it cannot be used in settings where the time-series could present zero-valued observations. However, the framework is general enough that any other measure could be used instead.

The proposed framework has been designed to guarantee accurate price forecasts of different travel products price and it is conceived for travel applications that might be already deployed. As such, it was undesirable to propose a method that performs forecasting and monitoring altogether, as in meta-learning, since this would require deprecating already deployed models to implement a new system. Instead, thanks to the proposed fully data-driven approach, *monitoring models* are completely independent of those doing the forecasts, i.e. the *monitored models*, thus allowing a transparent implementation of the monitoring and selection framework.

Although our main objective is to guarantee stable accurate price forecasts, the problem we address is relevant beyond our concrete application. Sculley *et al.* [4] introduced the term hidden technical debt to formalize and help reason about the long term costs of maintainable ML systems. According to their terminology, the proposed model monitoring and selection framework addresses two problems: 1) the monitoring and testing of dynamic systems, which is the task of continuously assessing that a system is working as intended; and 2) the production management debt, which refers to the costs associated to the maintenance of a large number of models that run simultaneously. Our solution represents a simple, flexible and accurate alternative to these problems.

Part II

Scalable Probabilistic Machine Learning

5
Context

So far we have seen how machine learning can be used to improve two main airline industry applications. In this second part of the thesis, we will focus on one of the most important components of machine learning, that is optimization.

5.1 Optimization algorithms for ML

Optimization algorithms [7] represent the core of every machine learning model. Considering a prediction function parameterized by a real vector $\mathbf{x} \in \mathbb{R}^d$, these parameters are iteratively updated according to a loss function:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t), \tag{5.1}$$

where η_t is the step size and $\nabla f(\mathbf{x}_t)$ is the gradient of the loss function $f(\mathbf{x}_t)$, which measures the performance of the model. This is the *batch* version of the very well-known gradient descent algorithm, where the loss function is evaluated considering all data points. When the input dataset or the model become large, we resort to the *stochastic* version of gradient descent, where the model updates take the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t, i) \tag{5.2}$$

Each iteration of this method is very cheap, in that it involves only the computation of the gradient $\nabla f(\mathbf{x}_t, i)$ corresponding to one sample. Thus Stochastic Gradient Descent (SGD) trades a larger number of iterations to converge for a cheaper cost per iteration. Finally, another version of SGD allows one to control the number and the cost per iteration, by evaluating the gradient on a small subset of samples $\xi_t \subseteq \{1, \ldots, n\}$, called *mini-batch*, chosen randomly in each iteration:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta_t}{|\xi_t|} \sum_{i \in \xi_t} \nabla f(\mathbf{x}_t, i) \tag{5.3}$$

5.1.1 Stochastic Gradient Descent

Mini-batch SGD [6] comes with good statistical properties and it is the preferred choice for optimization in most of machine learning models. The first important property is that SGD is an unbiased estimator of the full gradient:

$$\mathbb{E}\left[\frac{1}{|\xi_t|}\sum_{i\in\xi_t}\nabla f\left(\mathbf{x}_t, i\right)\right] = \frac{1}{n}\sum_{i=1}^{n}\nabla f\left(\mathbf{x}_t, i\right) = \nabla f\left(\mathbf{x}_t\right) \tag{5.4}$$

Secondly, under the following assumptions, it is possible to establish SGD convergence guarantees [7]. We define here $\mathbf{g}\left(\mathbf{x}_t, \xi_t\right) = \frac{1}{|\xi_t|}\sum_{i\in\xi_t}\nabla f\left(\mathbf{x}_t, i\right)$.

Assumption 1 *The objective function $f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and the gradient function of f, namely, $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$ is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.,*

$$\|\nabla f(w) - \nabla f(\bar{w})\|_2 \leq L\|w - \bar{w}\|_2 \text{ for all } \{w, \bar{w}\} \subset \mathbb{R}^d \tag{5.5}$$

Assumption 2 *The objective function and Stochastic Gradient (SG) satisfy the following:*

(a) The sequence of iterates $\{\mathbf{x}_t\}$ is contained in an open set over which f is bounded below by a scalar f_{inf} .

(b) There exist scalars $\mu_G \geq \mu > 0$ such that, for all $t \in \mathbb{N}$,

$$\begin{gathered}\nabla f\left(\mathbf{x}_t\right)^T \mathbb{E}_{\xi_t}\left[g\left(\mathbf{x}_t, \xi_t\right)\right] \geq \mu \left\|\nabla f\left(\mathbf{x}_t\right)\right\|_2^2 \text{ and} \\ \left\|\mathbb{E}_{\xi_t}\left[g\left(\mathbf{x}_t, \xi_t\right)\right]\right\|_2 \leq \mu_G \left\|\nabla f\left(\mathbf{x}_t\right)\right\|_2 .\end{gathered} \tag{5.6}$$

(c) There exist scalars $M \geq 0$ and $M_V \geq=$ such that, for all $t \in \mathbb{N}$,

$$\mathbb{V}_{\xi_t}\left[g\left(\mathbf{x}_t, \xi_t\right)\right] \leq M + M_V \left\|\nabla f\left(\mathbf{x}_t\right)\right\|_2^2 . \tag{5.7}$$

Intuitively, Assumption 1 ensures that the gradient of f does not change arbitrarily quickly with respect to the parameter vector. Assumption 2(a) requires the objective function to be bounded below over the region explored by the algorithm. Assumption 2(b) states that, in expectation, the vector $-g\left(\mathbf{x}_t, \xi_t\right)$ is a direction of sufficient descent for f from $\mathbf{x}_k$ with a norm comparable to the norm of the gradient. Assumption 2(c) states that the variance of $g\left(\mathbf{x}_t, \xi_t\right)$ is restricted, but in a relatively minor manner.

Theorem *Under Assumption 1 and Assumption 2, if we further assume that the objective function f is twice differentiable, that the mapping $\mathbf{x} \mapsto \|\nabla f(\mathbf{x})\|_2^2$ has Lipschitz-continuous derivatives and the step size satisfies requirements $\sum_{t=1}^{\infty}\eta_t = \infty$ and $\sum_{t=1}^{\infty}\eta_k^2 < \infty$ [6], it is possible to show [7] that:*

$$\lim_{t\to\infty}\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_t\right)\right\|_2^2\right] = 0 \tag{5.8}$$

This result is valid in case of non-convex objective functions, which we find in many important machine learning problems.

5.1.2 Large-scale machine learning

The latest advances in machine learning, together with the increasing availability of massive datasets, have given rise to a new research field called large-scale machine learning, where deep learning models with billions of parameters are trained on very big datasets. This has led to the use of distributed systems, where multiple nodes perform model training in parallel. There are three main alternatives of distributed systems: we can run a model on multiple cores of the same CPU, on multiple CPUs/GPUs of the same machine or we can use multiple machines of a cluster.
Regarding the distributed architecture, two main paradigms are possible: *Parameter Server* and *AllReduce*. In the former multiple nodes compute stochastic gradients using partitions of the dataset and a parameter server maintains a globally shared model, while in the latter nodes exchange gradients and perform the aggregation step. In our work we focus on the parameter server paradigm, since it represents the most commonly used distributed architecture.
Distributed systems can be synchronous or asynchronous. In synchronous systems, the aggregation step is performed only when the parameters server has received all gradients from workers. This guarantees higher statistical efficiency, but the presence of stragglers slows down the learning algorithm. In asynchronous systems [10, 80, 81, 82], model parameters are updated as soon as the parameter server receives a new gradient from the workers. This provides fast model updates, but the different speed of training can lead to the use of stale parameters, which might affect convergence speed. In this context, communication overheads have been considered as a key issue to address, and a large number of works have been proposed to mitigate such overheads. Quantization techniques reduce the number of bits to represent the gradients before communication [83, 84, 85], sparsification methods select a subset of the gradient components to communicate [86, 87, 88, 89, 90, 91], and loss-less methods use large mini-batches to increase the computation-communication ratio [92, 93]. Given the remarkable results achieved by sparsification methods in synchronous setups, we introduce in Chapter 6 a convergence analysis of sparsified SGD in asynchronous setting.

5.2 Optimization algorithms for Bayesian Inference

Latest advances in machine learning research have led to the use of such models in an ever increasing number of applications, including ones previously requiring human expertise. However, the adoption of machine learning models to sensitive applications, such as self-driving cars, automated medical diagnosis, is still under study and heavily-publicised incidents have cast a heavy shadow over their robustness and reliability [14, 15]. Although the cause of such deficiencies can be attributed to several factors, a persistent concern is that machine learning lacks the generalisation capabilities innate to human reasoning that enable greater adaptability and flexibility when affronted with a new task or unexpected setting. This highlights the importance of applying greater caution when relying on the raw

outcomes of such models for critical decision making. This shift in perspective has sparked a resurgence of interest in probabilistic modelling, which is inherently intended to capture the uncertainty or lack of knowledge a model might have about a learned task [94]. To this end, probabilistic models provide uncertainty estimates accompanying predictions, which give information on how likely the latter are to be correct.

5.2.1 Bayesian Modelling

In statistical inference we find two main schools of thought. The frequentist approach constructs a hypothesis relying exclusively on the data available at training time, regardless of any prior assumptions on the expected outcomes of the hypothesis. On the other hand, Bayesian inference [95, 96] represents a more principled way of developing probabilistic models. One of the primary motivations for using Bayesian inference is best described by way of its connection to the principle of Occam's razor [97, 98]. In few words, Occam's razor denotes a preference for simpler explanations that sufficiently describe observations over more complex alternatives; thinking about machine learning, this can be interpreted as favouring models which generalise well as opposed to others that overfit the data available at training time. Bayes' theorem is formally given as

$$p\left(\theta \mid \mathcal{D}, \mathcal{M}_i\right) = \frac{p\left(\mathcal{D} \mid \theta, \mathcal{M}_i\right) p\left(\theta \mid \mathcal{M}_i\right)}{p\left(\mathcal{D} \mid \mathcal{M}_i\right)} \tag{5.9}$$

where θ denotes the parameters characterising a model $\mathcal{M}_i$ in a finite set of candidates M and $\mathcal{D}$ is the available data. The distribution $p\left(\mathcal{D} \mid \theta, \mathcal{M}_i\right)$ represents our *prior* beliefs on the values which the model's parameters are expected to take, while $p\left(\theta \mid \mathcal{M}_i\right)$ measures the *likelihood* of observing the available data using the designated model configuration. Dividing by the model *evidence* $p\left(\mathcal{D} \mid \mathcal{M}_i\right)$, which also serves as a normalising constant, yields the *posterior* probability of the parameters θ by updating our prior assumptions on what values they should take with the likelihood of generating the observed data. In this case, fitting the model amounts to identifying the setting of θ which maximises the evidence for model $\mathcal{M}_i$ (using say maximum a posteriori inference).

5.2.2 Approximate Bayesian inference

Although conceptually simple, applying Bayesian inference poses a number of computational challenges. Firstly, in high-dimensional space the evidence term at the denominator of Equation (5.9) involves sums and integrals that are computationally hard. Besides this, in some problems prior and likelihood of Equation (5.9) are not conjugate, which means we do not know the form of the posterior to compute it analytically.
We therefore need to resort to approximation methods in order to perform Bayesian inference in practice. The two most popular approximation methods for this purpose are Markov Chain Monte Carlo (MCMC) and Variational Inference (VI). MCMC consists in sampling directly from the posterior distribution, thus avoiding to deal with intractable computations. The advantage of MCMC is that the samples can approximate the exact

posterior arbitrarily well if we do a sufficient number of iterations. The downside of MCMC is that in practice we do not know how many times is sufficient, and getting a good approximation using MCMC can take a very long time. On the other hand, Variational inference [99] identifies an approximate posterior in a predefined family of probability densities that most closely matches the true, but computationally intractable, posterior. To do this we minimize the KL-divergence between approximate and true posterior, by optimizing a lower bound $\mathcal{L}$ on the marginal likelihood:

$$\mathcal{L} \geq \mathcal{E} - D_{KL}[q(\theta \mid \mathcal{D})||p(\theta \mid \mathcal{D})], \tag{5.10}$$

where $\mathcal{L} = \log[p(\mathcal{D} \mid \theta)]$ and $\mathcal{E} = \mathbb{E}_{q(\theta|\mathcal{D})}(\log[p(\mathcal{D} \mid \theta)])$. The first term of the evidence lower bound favours models that properly fit the data, while the subtracted term penalises models that deviate too far from the prior. When the variational posterior matches the true formulation, the D_{KL} term collapses to zero.

Although widely used, these methods require either long time or large number of parameters to converge. Recently, another family of approaches has taken hold in the field of approximate Bayesian inference. These are the so-called stochastic gradient MCMC algorithms, which use SGD with constant learning rate to obtain samples from the posterior distribution. Different versions have been proposed, SG Langevin dynamics [16], SG Hamiltonian Monte Carlo [17], SG thermostats [18], SG Fisher scoring [100], but they all employ stochastic gradients of $\log p(\theta, \mathcal{D})$ to improve convergence and computation of existing sampling algorithms. A complete classification of these algorithms can be found in [101]. Despite mathematical elegance and some promising results restricted to simple models, most of these works fall short in easily dealing with the complexity of the loss landscape of deep models [102, 103], for which stochastic optimization poses serious challenges. Existing methods are often unpractical, as they require ad-hoc, sophisticated vanishing learning rate schedules, and hyper-parameter tuning.

In this regard, we introduce in Chapter 7 a novel, practical approach to posterior sampling, which makes the SG noise isotropic using a fixed learning rate and that requires weaker assumptions than existing algorithms.

6
Convergence Analysis of Sparsified Asynchronous SGD

The analysis of Stochastic Gradient Descent [6] and its variants has received a lot of attention recently, due to its popularity as an optimization algorithm in machine learning; see [7] for an overview. SGD addresses the computational bottleneck of gradient descent by relying on stochastic gradients, which are cheaper to compute than full gradients. SGD trades a larger number of iterations to converge for a cheaper cost per iteration. The *mini-batch* variant of SGD allows one to control the number and the cost per iteration, making it the preferred choice for optimization in deep learning [104, 7].

We consider the problem of optimizing the d-dimensional parameter vector $\mathbf{x} \in \mathbb{R}^d$ of a model and its associated finite-sum *non-convex* loss function $f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}, i)$, where $f(\mathbf{x}, i)$, $i = 1, \ldots, n$ is the loss function for a single training sample i. SGD iterations have the following form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \mathbf{g}(\mathbf{x}_t, i),$$

where $\mathbf{x}_t, \mathbf{x}_{t+1} \in \mathbb{R}^d$ are the model iterates, $\eta_t > 0$ is the learning rate/step size and $\mathbf{g}(\mathbf{x}_t, i) = \nabla f(\mathbf{x}_t, i)$ is a stochastic gradient.

In this work, we are interested in the increasingly popular distributed setting, whereby SGD runs across several machines, which contribute to the model updates $\mathbf{x}_{t+1}$ by computing stochastic gradients of the loss using locally available training data [80, 105, 82, 81, 106]. The analysis of the convergence behavior of SGD, both in synchronous [8, 9, 7] and asynchronous [10, 11, 107, 108, 7] settings has been widely studied in the literature. In this work, we focus on the asynchronous setting, which is particularly challenging because distributed workers might produce gradient updates for a loss computed on *stale* versions of the current model iterates [105, 109, 110, 111, 112].

In this context, communication overheads have been considered as a key issue to address, and a large number of works have been proposed to mitigate such overheads [12, 13, 113, 84, 85, 86, 87]. In particular, sparsification methods [89, 90, 113] have achieved remarkable results, albeit for synchronous setups. The key idea is to apply smaller and more efficient gradient updates, by applying a sparsification operator to the stochastic gradient, which results in updates of size $k \ll d$.

In this work, we fill the gap in the literature and study sparsification methods in *asynchronous settings*. For the first time, we provide a concise and simple convergence rate analysis when the joint effects of sparsification and asynchrony are taken into account, and show that sparsified SGD converges at the same rate of standard SGD. Our empirical analysis of sparsified SGD complements our theory. We consider several delay distributions and show that, in practice, applying sparsification does not harm SGD performance. These results carry over when the system scales out, which is a truly desirable property.

6.1 Related work

The analysis of SGD [6] and its convergence properties has recently attracted a lot of attention, especially in the field of machine learning [7, 114], where SGD is considered the workhorse optimization method. Large scale models and massive datasets have motivated researchers to focus on distributed machine learning, whereby multiple machines compute stochastic gradients using partitions of the dataset and a parameter server maintains a globally shared model.

Asynchronous systems [10, 80, 81, 82] provide fast model updates, but the use of stale parameters might affect convergence speed. One way to reduce the staleness effect is to give a smaller weight to stale updates. In [106, 109] gradient contributions are dampened through a dynamic learning rate. Stale-synchronous parallel (SSP) models [105, 109] limit instead the maximum staleness, discarding updates that are too "old". Interestingly, the work in [115], suggests to view staleness as a form of implicit momentum, and study, under a simple model, how to adjust explicit, algorithmic momentum to counterbalance the effects of staleness.

Synchronous systems [8] guarantee higher statistical efficiency, but the presence of stragglers slows down the learning algorithm. One solution is provided by the so called local SGD models [110, 12, 13], which reduce the synchronization frequency by allowing nodes to compute local model parameters, which are averaged in a global model update. A second family of approaches seeks to improve synchronous systems by reducing the cost of communicating gradients upon every iteration. Quantization techniques reduce the number of bits to represent the gradients before communication [83, 84, 85], sparsification methods select a subset of the gradient components to communicate [86, 87, 88, 89, 90, 91], and loss-less methods use large mini-batches to increase the computation-communication ratio [92, 93].

Our work, along the lines of [107, 90], argues instead that staleness vanishes, asymptotically. Similarly, recent work [116] uses an elegant analysis technique to study the role of stale gradient updates and sparsification, albeit their effects are considered in isolation. In this work, instead, we provide a concise and simple convergence rate analysis for the joint effects of sparsification and staleness.

6.2 Contributions

We study finite-sum non-convex optimization of loss functions of the form $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$, and assume that f is continuously differentiable and bounded below, that $\nabla f(\mathbf{x})$ is L-Lipschitz smooth, that the variance of stochastic gradients is bounded, and that the staleness induced by asynchrony is also bounded. We analyze a mini-batch asynchronous SGD algorithm and apply a sparsification operator $\Phi_k[\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t)]$ with $k \ll d$, which can be coupled with an error correction technique, often called *memory* [89].

We prove ergodic convergence of the gradient of $f(\mathbf{x})$, for an appropriately chosen learning rate. In particular, we focus on memory-less variants, which are simpler to analyze, and show that asynchronous sparsified SGD converges at the same rate as standard SGD.

In this chapter, the main theoretical contribution is as follows. Let the sparsification coefficient be $\rho = k/d$. Then, it holds that:

$$\min_{0\leq t\leq T} \mathbb{E}\left[\left\|\nabla f(\mathbf{x}_t)\right\|^2\right] \leq \frac{\left(\sum_{t=0}^{T-1}\left(\frac{\eta_t^2 L}{2}\sigma^2\right)\right) + \Lambda + C}{\sum_{t=0}^{T-1}\left(\eta_t \rho\mu - \frac{\eta_t^2 L}{2}\right)},$$

where $\Lambda = f(\mathbf{x}_0) - \inf_{\mathbf{x}} f(\mathbf{x})$ and C, μ are finite positive constants (whose role will be clarified later). In particular for a suitable constant learning rate $\eta_t = \eta = \frac{\rho\mu}{L\sqrt{T}}$ we can derive as a corollary that:

$$\min_{0\leq t\leq T} \mathbb{E}\left[\left\|\nabla f(\mathbf{x}_t)\right\|^2\right] \leq \left(\frac{\sigma^2}{2} + \frac{(\Lambda + C)L}{(\rho\mu)^2}\right)\frac{1}{\sqrt{T}},$$

up to a negligible approximation for large T (details in the supplement).

We define sparsified SGD formally in Section 6.3, both in its memory and memory-less variants, and outline our proof for the memory-less case in Section 6.5. In Section 6.6 we provide an empirical study of the convergence behavior of the two variants of sparsified SGD, using simple and deep convolutional networks for image classification tasks. Our experiments show that sparsification does not harm SGD performance, even in the challenging scenario of an asynchronous, distributed system. Although we do not provide convergence guarantees for sparsified SGD with memory, our empirical results indicate that error correction dramatically improves the convergence properties of the algorithm.

6.3 Sparsified Asynchronous SGD

In this Section we define two variants of sparsified SGD algorithms, with and without error correction, and emphasize the role of model staleness induced by the asynchronous setup we consider.

The standard way to scale SGD to multiple computing nodes is via *data-parallelism*: a set of worker machines have access to the n training samples through a distributed filesystem. Workers process samples concurrently: each node receives a copy of the parameter vector

$\mathbf{x}_t$, and computes stochastic gradients locally. Then, they send their gradients to a parameter server (PS). Upon receiving a gradient from a worker, the PS updates the model by producing a new iterate $\mathbf{x}_{t+1}$.
Due to asynchrony, a computing node may use a *stale* version of the parameter vector: a worker may compute the gradient of $f(\mathbf{x}_{\tau_t})$, $\tau_t \leq t$. We call τ_t the *staleness of a gradient update*. As stated more formally in Section 6.4, in this work we assume **bounded staleness**, which is realistic in the setup we consider. Other works, e.g. that consider Byzantine attackers [109], drop this assumption.

Gradient sparsification. A variety of compression [117, 118], quantization [87, 84] and sparsification [90, 89] operators have been considered in the literature. Here we use sparsification, defined as follows:

Definition 1 *Given a vector $\mathbf{u} \in \mathbb{R}^d$, a parameter $1 \leq k \leq d$, the operator $\Phi_k(\mathbf{u}) : \mathbb{R}^d \to \mathbb{R}^d$ is defined as:*

$$(\Phi_k(\mathbf{u}))_i = \begin{cases} (\mathbf{u})_{\pi(i)}, & \text{if } i \leq k, \\ 0, & \text{otherwise} \end{cases}$$

where π is a permutation of the indices $\{1, \ldots, d\}$ such that $(|\mathbf{u}|)_{\pi(i)} \geq (|\mathbf{u}|)_{\pi(i+1)}, \forall i \in 1, \cdots, d$.

Essentially, $\Phi_k(\cdot)$ sorts vector elements by their magnitude, and keeps only the top-k. A key property of the operator we consider is called the k-contraction property [89], which we use in our convergence proofs.

Definition 2 *For a parameter $1 \leq k \leq d$, a k-contraction operator $\Phi_k(\mathbf{u}) : \mathbb{R}^d \to \mathbb{R}^d$ satisfies the following contraction property:*

$$\mathbb{E}\left\|\mathbf{u} - \Phi_k(\mathbf{u})\right\|^2 \leq \left(1 - \frac{k}{d}\right)\|\mathbf{u}\|^2.$$

Both the top-k operator we consider, and randomized variants, satisfy the k-contraction property [90, 89]. Next, we state a Lemma that we will use for our convergence rate results.

Lemma 6.3.1 *Given a vector $\mathbf{u} \in \mathbb{R}^d$, a parameter $1 \leq k \leq d$, and the top-k operator $\Phi_k(\mathbf{u}) : \mathbb{R}^d \to \mathbb{R}^d$ introduced in Definition 1, we have that:*

$$\|\Phi_k(\mathbf{u})\|^2 \geq \frac{k}{d}\|\mathbf{u}\|^2.$$

The proof of Lemma 6.3.1 uses the k-contraction property in Definition 2, as shown in Section 6.5.1.

Memory and memory-less sparsified asynchronous SGD. We define two variants of sparsified SGD: the first uses sparsified stochastic gradient updates directly, whereas the second uses an error correction technique which accumulates information suppressed by

sparsification. Since we consider an asynchronous, *mini-batch* version of SGD, additional specifications are in order.

Definition 3 *Given n training samples, let ξ_t be a set of indices sampled uniformly at random from $\{1, \cdots, n\}$, with cardinality $|\xi_t|$. Let τ_t be the bounded* staleness *induced by the asynchronous setup, with respect to the current iterate t. That is, $t - S \leq \tau_t \leq t$. A stale, mini-batch stochastic gradient is defined as:*

$$\mathbf{g}\left(\mathbf{x}_{\tau_t}, \xi_t\right) = \frac{1}{|\xi_t|} \sum_{i \in \xi_t} \nabla f\left(\mathbf{x}_{\tau_t}, i\right).$$

6.3.1 Memory-less sparsified SGD

Given the operator $\Phi_k(\cdot)$, the memory-less, asynchronous sparsified SGD algorithm amounts to the following:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \Phi_k\left(\mathbf{g}\left(\mathbf{x}_{\tau_t}, \xi_t\right)\right),$$

where $\{\eta_t\}_{t \geq 0}$ denotes a sequence of learning rates.

6.3.2 Sparsified SGD with memory

Given the operator $\Phi_k(\cdot)$, the asynchronous sparsified SGD with memory algorithm is defined as:

$$\begin{aligned}
\mathbf{x}_{t+1} &= \mathbf{x}_t - \eta_t \Phi_k\left(\mathbf{m}_t + \mathbf{g}\left(\mathbf{x}_{\tau_t}, \xi_t\right)\right), \\
\mathbf{m}_{t+1} &= \mathbf{m}_t + \mathbf{g}\left(\mathbf{x}_{\tau_t}, \xi_t\right) - \Phi_k\left(\mathbf{m}_t + \mathbf{g}\left(\mathbf{x}_{\tau_t}, \xi_t\right)\right),
\end{aligned}$$

where $\{\eta_t\}_{t \geq 0}$ denotes a sequence of learning rates, and $\mathbf{m}_t$ represents the memory vector that accumulates the elements of the stochastic gradient that have been suppressed by the operator $\Phi_k(\cdot)$.

6.4 Ergodic convergence

In this work, we focus on the memory-less variant of SGD, and we study its convergence properties. The convergence of sparsified SGD with memory has been studied for both strongly convex [89, 90] and non-convex objectives [90], but only in the synchronous case. Nevertheless, in our empirical study, we compare both variants, and verify that the one with memory considerably benefits from error correction, as expected [89]. Before proceeding with the statement of the main theorem, we formalize our assumptions.

Assumption 1 *$f(\mathbf{x})$ is continuously differentiable and bounded below:*

$$\inf_x f(\mathbf{x}) > -\infty.$$

Assumption 2 *$\nabla f(\mathbf{x})$ is L-Lipschitz smooth:*

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

Assumption 3 *The variance of the (mini-batch) stochastic gradients is bounded:*

$$\mathbb{E}\left[\|\mathbf{g}(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_t)\|^2\right] \leq \sigma^2,$$

where $\sigma^2 > 0$ is a constant.

Assumption 4 *Distributed workers might use stale models to compute gradients $\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t)$. We assume bounded staleness, that is: $t - S \leq \tau_t \leq t$. In other words, the model staleness τ_t satisfies the inequality $t - \tau_t \leq S$. We call $S \geq 0$ the maximum delay.*

Assumption 5 *Let the **expected cosine distance** be:*

$$\frac{\mathbb{E}\left[\langle \Phi_k(\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t)), \nabla f(\mathbf{x}_t)\rangle\right]}{\mathbb{E}\left[\|\Phi_k(\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t))\| \, \|\nabla f(\mathbf{x}_t)\|\right]} = \mu_t \geq \mu.$$

We assume, similarly to previous work [112], that the constant $\mu > 0$ measures the discrepancy between the sparsified stochastic gradient and the full gradient.

Theorem 6.4.1 *Let Assumptions 1–5 hold. Consider the memory-less sparsified SGD defined in Section 6.3.1, which uses the $\Phi_k(\cdot)$ top-k operator for a given $1 \leq k \leq d$. Then, for an appropriately defined learning rate $\eta_t = \frac{\rho\mu}{L\sqrt{t+1}}$ and for $\Lambda = \left(f(\mathbf{x}_0) - \inf_{\mathbf{x}} f(\mathbf{x})\right)$, it holds that:*

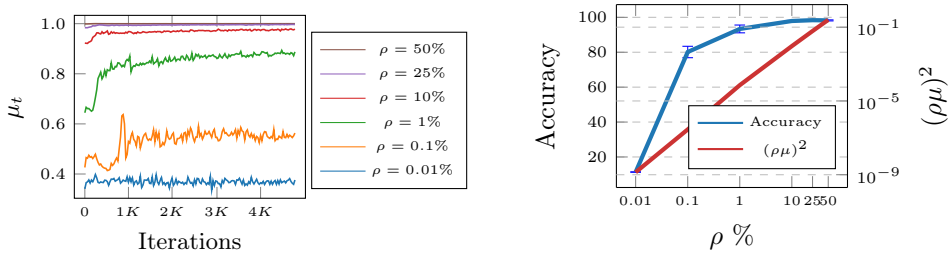
$$\min_{0 \leq t \leq T} \mathbb{E}\left[\|\nabla f(\mathbf{x}_t)\|^2\right] \leq \frac{\left(\sum_{t=0}^{T-1}\left(\frac{\eta_t^2 L}{2}\sigma^2\right)\right) + \Lambda + C}{\sum_{t=0}^{T-1}\left(\eta_t \rho\mu - \frac{\eta_t^2 L}{2}\right)}.$$

Corollary 6.4.1 *Let the conditions of Theorem 6.4.1 hold. Then for an appropriately defined constant learning rate $\eta_t = \eta = \frac{\rho\mu}{L\sqrt{T}}$, we have that:*

$$\min_{0 \leq t \leq T} \mathbb{E}\left[\|\nabla f(\mathbf{x}_t)\|^2\right] \leq \left(\frac{\sigma^2}{2} + \frac{(\Lambda + C)L}{(\rho\mu)^2}\right)\frac{1}{\sqrt{T} - \frac{1}{2}}.$$

Asymptotically, the convergence rate of memory-less sparsified SGD behaves as $\mathcal{O}(\frac{1}{\sqrt{T}})$, which matches the best known results for non-convex SGD [119], and for non-convex asynchronous SGD [107]. Note that considering a constant learning rate intuitively makes sense. When gradients are not heavily sparsified, i.e., ρ is large, we can afford a large learning rate. Similarly, when stale, sparse stochastic, and full gradients are similar, i.e., when μ is large, we can again set a large learning rate.

It is more difficult to quantify the role of the constant terms in Corollary 6.4.1, especially those involving sparsification. While it is evident that aggressive sparsification (extremely small ρ) could harm convergence, the exact role of the second constant term heavily depends on the initialization and the geometry of the loss function, which we do not address



(a) Expected cosine similarity μ_t (b) Test accuracy and $(\rho\mu)^2$

Figure 6.1 – Empirical results in support to Assumption 5. Experiments for ϕSGD with LeNet on Mnist, using a range of possible sparsification coefficients ρ.

in this work. We thus resort to a numerical study to clarify these questions, but introduce a proxy for measuring convergence rate. Instead of imposing a target test accuracy, and use the number of training iterations to measure convergence rate (which we found to be extremely noisy), we fix an iteration budget, and measure the test accuracy once training concludes.

Remarks. A careful assessment of Assumption 5 is in order. We assume that a sparse version of a stochastic gradient computed with respect to a stale model, does not diverge too much from the true, full gradient[1]. We measure this coherency trough a positive constant $\mu > 0$. However, it is plausible to question the validity of such assumption, especially in a situation where either the sparsification is too aggressive, or the maximum delay is too high.

We study the limits of our assumption empirically, and report our findings in Figure 6.1. The evolution of the expected cosine similarity μ_t defined in Assumption 5, reported here as a function of algorithmic progress, is in line with our assumption. Clearly, aggressive sparsification negatively impacts gradient coherency, as shown in Figure 6.1a. Moreover, as expected from Theorem 6.4.1, convergence rate measured through the proxy of test accuracy, also increases with $(\rho\mu)^2$. When sparsification is too aggressive, $(\rho\mu)^2$ is too small, which harms convergence.

6.5 Proof of Theorem 6.4.1

In this section we build all the useful tools to formalize the proof of Theorem 6.4.1. In Section 6.5.1 we prove the k-contraction lemma, in Section 6.5.2 we restate assumptions for simplicity, we derive useful facts in Section 6.5.3, derive a tighter bounding term for the sum of magnitudes of stale gradients in Section 6.5.4, and finally derive the full proof of the convergence theorem in Section 6.5.5.

[1] A similar remark, albeit without sparsification, has been made in [112].

6.5.1 Proof of Lemma 6.3.1

Given a vector $\mathbf{u} \in \mathbb{R}^d$, a parameter $1 \leq k \leq d$, and the top-k operator $\Phi_k(\mathbf{u}) : \mathbb{R}^d \to \mathbb{R}^d$ defined in Definition 1, we have that:

$$\|\Phi_k(\mathbf{u})\|^2 \geq \frac{k}{d}\|\mathbf{u}\|^2$$

In fact we can write $\|\mathbf{u}\|^2$ as follows:

$$\begin{aligned} \|\Phi_k(\mathbf{u})\|^2 &= \|\mathbf{u}\|^2 - \|\mathbf{u} - \Phi_k\mathbf{u}\|^2 \\ &\geq \|\mathbf{u}\|^2 - \left(1 - \frac{k}{d}\right)\|\mathbf{u}\|^2 \\ &\geq \frac{k}{d}\|\mathbf{u}\|^2 \end{aligned} \tag{6.1}$$

Where the inequality is obtained by simply applying the k-contraction property.

6.5.2 Recap of Assumptions

We start by rewriting for simplicity Assumption 1 to Assumption 5:

1. $f(\mathbf{x})$ is continuously differentiable and bounded below: $\inf_x f(\mathbf{x}) > -\infty$
2. $\nabla f(\mathbf{x})$ is L-Lipschitz smooth:

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

3. The variance of the (mini-batch) stochastic gradients is bounded:

$$\mathbb{E}\left[\|\mathbf{g}(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_t)\|^2\right] \leq \sigma^2,$$

   where $\sigma^2 > 0$ is a constant.

4. The staleness is bounded, that is: $t - S \leq \tau_t \leq t$, $S \geq 0$. In other words, the model staleness τ_t satisfies the inequality $t - \tau_t \leq S$. We call $S \geq 0$ the maximum delay.

5. The cosine distance between sparse, stale and stochastic gradient and the full one is lower bounded

$$\frac{\mathbb{E}\left[\langle \Phi_k(\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t)), \nabla f(\mathbf{x}_t)\rangle\right]}{\mathbb{E}\left[\|\Phi_k(\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t))\| \, \|\nabla f(\mathbf{x}_t)\|\right]} = \mu_t \geq \mu$$

Notice moreover that ξ_t is statistically independent from $\{\mathbf{x}_0, \cdots, \mathbf{x}_t\}$.

6.5.3 Useful facts

Starting from Assumption 2 we can write that:

$$f(\mathbf{x}) \leq f(\mathbf{y}) + \langle \mathbf{x} - \mathbf{y}, \nabla f(\mathbf{y})\rangle + \frac{L}{2}\|\mathbf{x} - \mathbf{y}\|^2 \quad \forall \mathbf{x}, \mathbf{y}$$

Trivially we can rewrite this inequality by using as arguments the two vectors $\mathbf{x}_t, \mathbf{x}_{t+1}$:

$$\begin{aligned} f(\mathbf{x}_{t+1}) &\leq f(\mathbf{x}_t) + \langle \mathbf{x}_{t+1} - \mathbf{x}_t, \nabla f(\mathbf{x}_t) \rangle + \\ &+ \frac{L}{2} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 \\ &= f(\mathbf{x}_t) - \eta_t \langle \Phi_k [\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t)], \nabla f(\mathbf{x}_t) \rangle \\ &+ \frac{\eta_t^2 L}{2} \|\Phi_k [\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t)]\|^2 \end{aligned} \tag{6.2}$$

where the last equality is due to $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \Phi_k(\mathbf{g}(\mathbf{x}_{\tau_t}, \xi_t))$. Notice that even if $\mathbf{x}_t$ as well as ξ_t, τ_t and consequently $f(\mathbf{x}_t)$ and $g(\mathbf{x}_t)$ are random processes, due to the geometric constraints imposed on the cost function, the above inequality holds with probability 1.

The second useful quantity we derive is a bound for squared magnitude of $\mathbf{g}(\mathbf{x}_t, \xi_t)$. We start with Assumption 3.

Before proceeding, we introduce the following notation: Ω is the set of ALL random variables (i.e. $\Omega = \{\xi_0, \cdots, \xi_t, \mathbf{x}_0, \cdots, \mathbf{x}_t, \tau_0, \cdots, \tau_t\}$), furthermore, we indicate with $\sim \xi_t$ the set difference between Ω and ξ_t.

We write:

$$\begin{aligned} &\mathbb{E}_\Omega \left[\|\mathbf{g}(\mathbf{x}_t, \xi_t) - \nabla f(\mathbf{x}_t)\|^2 \right] \\ &= \mathbb{E}_\Omega \left[\|\mathbf{g}(\mathbf{x}_t, \xi_t) - \mathbb{E}_{\xi_t}(\mathbf{g}(\mathbf{x}_t, \xi_t))\|^2 \right] \\ &= \mathbb{E}_{\sim \xi_t} \left[\mathbb{E}_{\xi_t} \left[\|\mathbf{g}(\mathbf{x}_t, \xi_t) - \mathbb{E}_{\xi_t}(\mathbf{g}(\mathbf{x}_t, \xi_t))\|^2 \right] \right] \\ &= \mathbb{E}_{\sim \xi_t} \left[\mathbb{E}_{\xi_t} [\|\mathbf{g}(\mathbf{x}_t, \xi_t)\|] - \|\mathbb{E}_{\xi_t}(\mathbf{g}(\mathbf{x}_t, \xi_t))\|^2 \right] \\ &= \mathbb{E}_\Omega [\|\mathbf{g}(\mathbf{x}_t, \xi_t)\|] - \mathbb{E}_\Omega [\|\nabla f(\mathbf{x}_t)\|] \leq \sigma^2 \end{aligned}$$

from which:

$$\mathbb{E}_\Omega [\|\mathbf{g}(\mathbf{x}_t, \xi_t)\|] \leq \mathbb{E}_\Omega [\|\nabla f(\mathbf{x}_t)\|] + \sigma^2 \tag{6.3}$$

6.5.4 Bounding magnitudes of delayed gradients

Differently from [112], we derive a tighter bound for the following term:

$$\sum_{t=0}^{T-1} \eta_t^2 \mathbb{E} \left[\|\nabla f(\mathbf{x}_{\tau_t})\|^2 \right] \tag{6.4}$$

Indeed, thanks to the fact that η_t is a decreasing sequence, and using the law of total expectation:

$$\begin{aligned}
&\sum_{t=0}^{T-1} \eta_t^2 \mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_{\tau_t}\right)\right\|^2\right] \\
&= \sum_{t=0}^{T-1} \eta_t^2 \sum_{l=t-S}^{t} Pr(\tau_t = l) \mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_l\right)\right\|^2\right] \\
&\leq \sum_{t=0}^{T-1} \sum_{l=t-S}^{t} \eta_l^2 Pr(\tau_t = l) \mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_l\right)\right\|^2\right]
\end{aligned}$$

Before proceeding, it is useful to introduce a new random quantity, the delay D, distributed according to some probability density function $Pr(D = i) = \pi_i$. Notice that the true relationship between $Pr(\tau_t)$ and $Pr(D)$ is:

$$Pr(\tau_t = l) = \frac{\pi_{t-l}}{\sum_{i=0}^{min(t,S)} \pi_i}.$$

Since $t - S \leq \tau_t \leq t$, obviously the delay variable D has support boundend in $[0, S]$. Moreover, to reduce clutter, we define: $\psi_l = \eta_l^2 \mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_l\right)\right\|^2\right]$.

Now, we can continue our derivation as:

$$
\begin{aligned}
&\sum_{t=0}^{T-1}\sum_{l=t-S}^{t}\frac{\pi_{t-l}}{\sum_{i=0}^{min(t,S)}\pi_i}\psi_l = \\
&\frac{\pi_S\psi_{-S+0}+\pi_{S-1}\psi_{-S+1}+\cdots+\pi_1\psi_{-1}+\pi_0\psi_0}{\pi_0}+ \\
&\frac{\pi_S\psi_{-S+1}+\pi_{S-1}\psi_{-S+2}+\cdots+\pi_1\psi_0+\pi_0\psi_1}{\pi_0+\pi_1}+ \\
&\frac{\pi_S\psi_{-S+2}+\pi_{S-1}\psi_{-S+3}+\cdots+\pi_1\psi_1+\pi_0\psi_2}{\pi_0+\pi_1+\pi_2}+ \\
&\frac{\pi_S\psi_{-S+3}+\pi_{S-1}\psi_{-S+4}+\cdots+\pi_1\psi_2+\pi_0\psi_3}{\pi_0+\pi_1+\pi_2+\pi_3}+ \\
&\cdots \\
&\frac{\pi_S\psi_{T-S}+\cdots\cdots\cdots\quad+\pi_1\psi_{T-1}+\pi_0\psi_T}{1}\leq \\
&\pi_S\psi_{-S+0}+\pi_{S-1}\psi_{-S+1}+\cdots+\pi_1\psi_{-1}+\pi_0\psi_0+C_0+ \\
&\pi_S\psi_{-S+1}+\pi_{S-1}\psi_{-S+2}+\cdots+\pi_1\psi_0+\pi_0\psi_1+C_1+ \\
&\pi_S\psi_{-S+2}+\pi_{S-1}\psi_{-S+3}+\cdots+\pi_1\psi_1+\pi_0\psi_2+C_2+ \\
&\pi_S\psi_{-S+3}+\pi_{S-1}\psi_{-S+4}+\cdots+\pi_1\psi_2+\pi_0\psi_3+C_3+ \\
&\cdots \\
&\pi_S\psi_{T-S}+\cdots\cdots\cdots\quad+\pi_1\psi_{T-1}+\pi_0\psi_T+0 \\
&\leq\sum_{t=0}^{T-1}\psi_t+C \\
&=\sum_{t=0}^{T-1}\eta_t^2\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_t\right)\right\|^2\right]+C
\end{aligned}
$$

where C is a suitable, finite, constant. We thus proved a strict bound on the sum of magnitudes of delayed gradients as

$$
\sum_{t=0}^{T-1}\eta_t^2\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_{\tau_t}\right)\right\|^2\right]\leq\sum_{t=0}^{T-1}\eta_t^2\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_t\right)\right\|^2\right]+C.
$$

6.5.5 Derivation of the theorem

We start the derivation from Equation (6.2). We rearrange the inequality to bound the increment of cost function at time instant t as:

$$
\begin{aligned}
f\left(\mathbf{x}_{t+1}\right)-f\left(\mathbf{x}_t\right)\leq&-\eta_t\left\langle\Phi_k\left[\mathbf{g}\left(\mathbf{x}_{\tau_t},\xi_t\right)\right],\nabla f\left(\mathbf{x}_t\right)\right\rangle \\
&+\frac{\eta_t^2L}{2}\left\|\Phi_k\left[\mathbf{g}\left(\mathbf{x}_{\tau_t},\xi_t\right)\right]\right\|^2
\end{aligned}
$$

Written in this form, we are still dealing with random quantities. We are interested in taking the expectation of above inequality with respect to **all** random processes. Then:

$$\begin{aligned}&\mathbb{E}_{\Omega}\left[f\left(\mathbf{x}_{t+1}\right)-f\left(\mathbf{x}_{t}\right)\right]\\&\leq\mathbb{E}_{\Omega}\left[-\eta_{t}\left\langle\Phi_{k}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right],\nabla f\left(\mathbf{x}_{t}\right)\right\rangle+\frac{\eta_{t}^{2}L}{2}\left\|\Phi_{k}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right]\right\|^{2}\right]\end{aligned}$$

The strategy to continue the proof is to find an upper bound for the expectation term $\mathbb{E}_{\Omega}\left[\left\|\Phi_{k}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right]\right\|^{2}\right]$ and a lower bound for the expectation term $\mathbb{E}_{\Omega}\left[\left\langle\Phi_{k}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right],\nabla f\left(\mathbf{x}_{t}\right)\right\rangle\right]$. We start with the upper bound as:

$$\begin{aligned}&\mathbb{E}_{\Omega}\left[\left\|\Phi_{k}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right]\right\|^{2}\right]\leq\mathbb{E}_{\Omega}\left[\left\|\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right\|^{2}\right]\\&\leq\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_{\tau_{t}}\right)\right\|^{2}\right]+\sigma^{2},\end{aligned}$$

where the first inequality is a trivial consequence of sparsification and the second is the application of Equation (6.3).

As anticipated, we aim at lower bounding the term: $\mathbb{E}_{\Omega}\left[\left\langle\Phi_{k}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right],\nabla f\left(\mathbf{x}_{t}\right)\right\rangle\right]$. We start with Assumption 5 and write:

$$\begin{aligned}&\mathbb{E}_{\Omega}\left[\left\langle\Phi_{k}\left(\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right),\nabla f\left(\mathbf{x}_{t}\right)\right\rangle\right]\\&\geq\mu\mathbb{E}_{\Omega}\left[\left\|\Phi_{k}\left(\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right)\right\|\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\\&=\mu\mathbb{E}_{\sim\xi_{t}}\left[\mathbb{E}_{\xi_{t}}\left[\left\|\Phi_{k}\left(\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right)\right\|\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\right]\\&=\mu\mathbb{E}_{\sim\xi_{t}}\left[\mathbb{E}_{\xi_{t}}\left[\left\|\Phi_{k}\left(\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right)\right\|\right]\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\end{aligned}$$

We focus on the term $\mathbb{E}_{\xi_{t}}\left[\left\|\Phi_{k}\left(\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right)\right\|\right]$ and, thanks to the k-contraction property and the inequality of the norm of expected values, we can write:

$$\begin{aligned}&\mathbb{E}_{\Omega}\left[\left\langle\Phi_{k}\left(\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right),\nabla f\left(\mathbf{x}_{t}\right)\right\rangle\right]\\&\geq\mu\rho\mathbb{E}_{\sim\xi_{t}}\left[\mathbb{E}_{\xi_{t}}\left[\left\|\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right\|\right]\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\\&\geq\mu\rho\mathbb{E}_{\sim\xi_{t}}\left[\left\|\mathbb{E}_{\xi_{t}}\left[\mathbf{g}\left(\mathbf{x}_{\tau_{t}},\xi_{t}\right)\right]\right\|\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\\&=\mu\rho\mathbb{E}_{\sim\xi_{t}}\left[\left\|\nabla f\left(\mathbf{x}_{\tau_{t}}\right)\right\|\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\end{aligned}$$

Before proceeding, we reasonably assume that:

$$\mathbb{E}_{\sim\xi_{t}}\left[\left\|\nabla f\left(\mathbf{x}_{\tau_{t}}\right)\right\|\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|\right]\geq\mathbb{E}_{\sim\xi_{t}}\left[\left\|\nabla f\left(\mathbf{x}_{t}\right)\right\|^{2}\right],\tag{6.5}$$

since stale versions of the gradient should be larger, in magnitude, than recent versions.

Combining everything together we rewrite our initial inequality as:

$$\Delta_t = \mathbb{E}_\Omega \left[f\left(\mathbf{x}_{t+1}\right) - f\left(\mathbf{x}_t\right) \right] \leq -\eta_t \rho \mu \mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \frac{\eta_t^2 L}{2} \left(\mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_{\tau_t}\right) \|^2 \right] + \sigma^2 \right).$$

To derive a convergence bound it is necessary to sum all the increments over t from 0 to T:

$$\begin{aligned}
&\sum_{t=0}^{T-1} \Delta_t \leq \sum_{t=0}^{T-1} \left(-\eta_t \rho \mu \mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \right. \\
&\left. \frac{\eta_t^2 L}{2} \left(\mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_{\tau_t}\right) \|^2 \right] + \sigma^2 \right) \right) \\
&\leq \left(\sum_{t=0}^{T-1} -\eta_t \rho \mu \mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \right. \\
&\left. \frac{\eta_t^2 L}{2} \left(\mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \sigma^2 \right) \right) + C = \\
&\left(\sum_{t=0}^{T-1} \left(-\eta_t \rho \mu + \frac{\eta_t^2 L}{2} \right) \mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \frac{\eta_t^2 L}{2} \sigma^2 \right) + C,
\end{aligned}$$

where for the second inequality we used the result of Section 6.5.4.

We further manipulate the result by noticing that: $\sum_{t=0}^{T-1} \Delta_t = \mathbb{E}_\Omega \left[f(\mathbf{x}_{t+1}) - f(\mathbf{x}_0) \right]$. Moreover since $\Lambda = \mathbb{E}_\Omega \left[f\left(\mathbf{x}_0\right) \right] - \inf_{\mathbf{x}} f\left(\mathbf{x}\right)$, it is easy to show that $\Lambda \geq \mathbb{E}_\Omega \left[f(\mathbf{x}_0) - f(\mathbf{x}_{t+1}) \right] = -\sum_{t=0}^{T-1} \Delta_t$. We combine the bounds together as:

$$-\Lambda \leq \sum_{t=0}^{T-1} \Delta_t \leq \left(\sum_{t=0}^{T-1} \left(-\eta_t \rho \mu + \frac{\eta_t^2 L}{2} \right) \mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \frac{\eta_t^2 L}{2} \sigma^2 \right) + C.$$

Then:

$$-\Lambda - C \leq \left(\sum_{t=0}^{T-1} \left(-\eta_t \rho \mu + \frac{\eta_t^2 L}{2} \right) \mathbb{E}_{\sim \xi_t} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] + \frac{\eta_t^2 L}{2} \sigma^2 \right),$$

and finally:

$$\left(\sum_{t=0}^{T-1} \left(\eta_t \rho \mu - \frac{\eta_t^2 L}{2} \right) \mathbb{E} \left[\| \nabla f\left(\mathbf{x}_t\right) \|^2 \right] \right) \leq \left(\sum_{t=0}^{T-1} \left(\frac{\eta_t^2 L}{2} \sigma^2 \right) \right) + \Lambda + C.$$

We conclude that:

$$\min_{0\leq t\leq T}\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_t\right)\right\|^2\right]\leq\frac{\left(\sum_{t=0}^{T-1}\left(\frac{\eta_t^2L}{2}\sigma^2\right)\right)+\Lambda+C}{\sum_{t=0}^{T-1}\left(\eta_t\rho\mu-\frac{\eta_t^2L}{2}\right)}.$$

We then proceed by prooving the simple corollary 6.4.1 of Theorem 6.4.1. By choosing a suitable constant learning rate $\eta_t=\eta=\frac{\rho\mu}{L\sqrt{T}}$ we can rewrite the inequality as

$$\min_{0\leq t\leq T}\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_t\right)\right\|^2\right]\leq\frac{\left(T\left(\frac{\eta^2L}{2}\sigma^2\right)\right)+\Lambda^{'}}{T\left(\eta\rho\mu-\frac{\eta^2L}{2}\right)}.$$

where for simplicity we have defined $\Lambda^{'}=\Lambda+C$.
We then notice that $\eta\rho\mu=\frac{(\rho\mu)^2}{L\sqrt{T}}$ and that $\frac{\eta^2L}{2}=\frac{(\rho\mu)^2}{2LT}$, consequently we can rewrite the upper bound as

$$\frac{\frac{(\rho\mu)^2\sigma^2}{2L}+\Lambda^{'}}{\frac{(\rho\mu)^2\sqrt{T}}{L}-\frac{(\rho\mu)^2}{2L}}=\frac{\frac{(\rho\mu)^2\sigma^2}{2L}+\Lambda^{'}}{\frac{(\rho\mu)^2}{L}\left(\sqrt{T}-\frac{1}{2}\right)}.$$

Since we are intersted in $\mathcal{O}$ convergence rate, we can safely assume that $\left(\sqrt{T}-\frac{1}{2}\right)\simeq\sqrt{T}$ and that thus

$$\begin{aligned}\min_{0\leq t\leq T}\mathbb{E}\left[\left\|\nabla f\left(\mathbf{x}_t\right)\right\|^2\right]&\lesssim\frac{\frac{(\rho\mu)^2\sigma^2}{2L}+\Lambda^{'}}{\frac{(\rho\mu)^2}{L}\sqrt{T}}\\&=\left(\frac{\sigma^2}{2}+\frac{\Lambda^{'}L}{(\rho\mu)^2}\right)\frac{1}{\sqrt{T}}\end{aligned}$$

6.6 Experiments

While the benefits of sparsification have been extensively validated in the literature [86, 87, 83, 88, 84], such works focus on communication costs in a synchronous setup, with the exception of the work in [89], which illustrates a simple experiment in a multi-core asynchronous setup. Instead, our experiments focus on verifying that: 1) the effects of staleness are negligible; 2) sparsification does not harm convergence rates, using test accuracy as a proxy; 3) the benefits of the memory mechanism applied to sparsified SGD. We consider several worker delay distributions, and we compare the performance of the three SGD variants: standard SGD, and sparsified SGD with and without memory. We also investigate the effects of scaling-out the system, by going up to 128 workers.
For our experimental campaign, we have built a custom simulator that plugs into existing machine learning libraries to leverage automatic differentiation and the vast availability of models, but abstracts away the complications of a real distributed setting. With this

setup, it is easy to compare a variety of stochastic optimization algorithms on realistic and complex loss functions. More details about our simulator are given in Paragraph 6.6.1.0.3.

6.6.1 Experimental setup

6.6.1.0.1 SGD variants

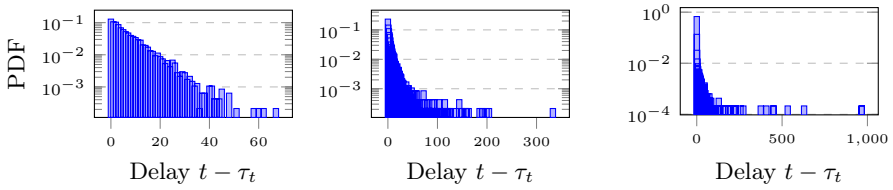
We compare sparsified SGD without (ϕSGD) and with memory (ϕMemSGD) to "vanilla" asynchronous SGD (ASGD). For all algorithms, and for all scenarios, we perform a grid search to find the best learning rate. When relevant, Figures report standard deviation, obtained by repeating our experiments 5 times. Note that for direct comparisons on individual experiments to be fair, we make sure to use the same algorithmic initialization for SGD (e.g., we use the same initial model parameters), and the same simulation seed.

6.6.1.0.2 Parameters

We configure the system architecture as follows: we consider a "parameter server" setup, whereby a given number of worker machines are connected to a master by a simple network model, we do not simulate network congestion, we impose fair bandwidth sharing, and we do not account for routing overheads.
In our simulations, both computation and communication costs can be modeled according to a variety of distributions. In this work we use uniformly distributed computation times with a small support, that are indicative of an homogeneous system. Instead of directly controlling the staleness of gradient updates, as done in other studies [109, 112], we indirectly induce staleness by imposing synthetic network delays, which we generate according to an exponential distribution with rate λ (the inverse of the mean). In particular, each worker samples a value for λ from a log-normal distribution with mean 0 and variance σ^2. Figure 6.2 shows the resulting delay distribution for the entire training period in a simulation with 8 workers, using different values of σ^2. As we increase σ^2, the maximum delay experienced by the workers increases, up to very large values. In addition, the mass of the distribution shifts towards lower delays; indeed, for higher values of σ^2, the majority of workers have small delays and only few workers experience very large delays. This is confirmed by the average staleness $\bar{\tau}$, which decreases as σ^2 increases. Notice that the interplay between communication delay and staleness is subtle: we provide a comprehensive description of the staleness generation process in Paragraph 6.6.1.0.3, with illustrations that help understanding the shape of the τ_t distribution.

6.6.1.0.3 The simulator and a note on staleness

In this Section we provide additional details to clarify the simulator structure, the definition of staleness and its generating process in our empirical validation. Figure 6.3 is an abstract representation of the distributed architecture we consider in this work. It consists of a time-line for each machine in the system: one for the *parameter server* (PS), and one for each worker W_i ($i = 3$ in the Figure).



(a) $\sigma^2 = 0.1$, $\bar{\tau} = 6.99$ (b) $\sigma^2 = 1$, $\bar{\tau} = 6.97$ (c) $\sigma^2 = 3$, $\bar{\tau} = 4.72$

Figure 6.2 – Delay distributions of a simulation run with LeNet on Mnist, in a distributed setting with 8 workers. For each worker we generate a network delay according to an exponential distribution with rate λ. We sample λ from a log-normal distribution with mean 0 and variance σ^2. For each configuration, we also report the resulting average staleness $\bar{\tau}$.

In our system model, the PS accepts contributions from workers and updates the current model iterate according to the variant of the SGD algorithm it implements. We use the notation $\mathbf{x}_n = U(\mathbf{x}_{n-1}, \nabla(\mathbf{x}_k))$ to indicate that the n_{th} model version at PS is obtained by updating the $n-1_{th}$ version using a gradient computed with the k_{th} version, according to the rules of the generic update algorithm $U(\cdot)$. In this case the staleness of the model update is equal to $n-1-k$. It is assumed that as soon as the PS update the parameters the worker immediately receives the updated model, indicated with $W_i \leftarrow \mathbf{x}_n$. Each worker is going to transmit many different updates ($N_{updates}$), each of which will reach the PS after a random delay. We define the sequence of delays for the i_{th} worker as $\{t^{(i)}_{\text{COMM},r}\}_{r=0}^{N_{updates}}$. The random generation process will be shortly after explained.
We are then ready to explain the example depicted in Figure 6.3:

1. At time instant $t_0 = 0$ all workers and PS are initialized with model $\mathbf{x}_0$.

2. At $t_1 = t^{(1)}_{\text{COMM},0}$ the PS receives the first gradient computed by W_1 using model $\mathbf{x}_0$. The PS updates the model ($\mathbf{x}_1 = U(\mathbf{x}_0, \nabla(\mathbf{x}_0))$) and send the update to W_1. In this case the staleness of the update is 0

3. At $t_2 = t^{(2)}_{\text{COMM},0}$ the PS receives the first gradient computed by W_2 using model $\mathbf{x}_0$. The PS updates the model ($\mathbf{x}_2 = U(\mathbf{x}_1, \nabla(\mathbf{x}_0))$) and send the update to W_2. In this case the staleness is $1 - 0 = 1$.

4. At $t_3 = t^{(3)}_{\text{COMM},0}$ the PS receives the first gradient computed by W_3 using model $\mathbf{x}_0$. The PS updates the model ($\mathbf{x}_3 = U(\mathbf{x}_2, \nabla(\mathbf{x}_0))$) and send the update to W_3. In this case the staleness is $3 - 1 = 2$.

5. At $t_4 = t_1 + t^{(1)}_{\text{COMM},1}$ the PS receives the second gradient computed by W_1 using model $\mathbf{x}_1$. The PS updates the model ($\mathbf{x}_4 = U(\mathbf{x}_3, \nabla(\mathbf{x}_1))$) and send the update to W_1. In this case the staleness is $2 - 0 = 2$.

6. At $t_5 = t_2 + t^{(2)}_{\text{COMM},1}$ the PS receives the second gradient computed by W_2 using model $\mathbf{x}_2$. The PS updates the model ($\mathbf{x}_5 = U(\mathbf{x}_4, \nabla(\mathbf{x}_2))$) and send the update to W_2. In this case the staleness is $4 - 2 = 2$.

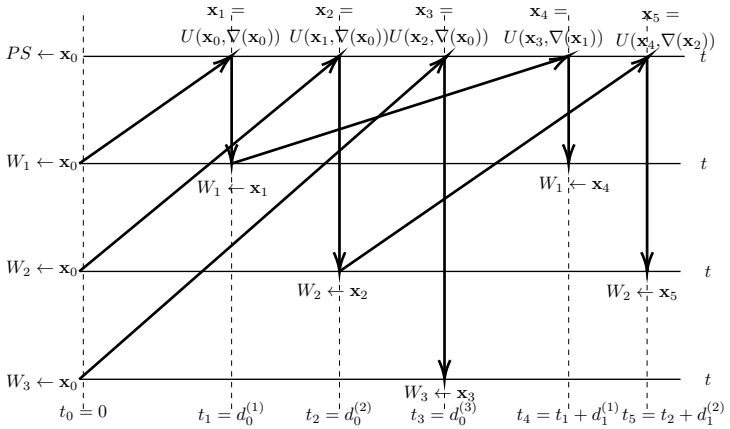


Figure 6.3 – Illustration of the distributed system operation. Example with one PS and 3 workers.

7. ...

Each communication delay is generated according to an exponential probability distribution with rate λ. To simulate network hetereogenity each worker has its own (fixed throughout the simulation) rate λ_i. These rates are extracted independently for each worker according to a lognormal distribution, i.e. $\ln(\lambda_i) \sim \mathcal{N}(0, \sigma^2)$ where σ^2 is a user defined parameter that can be used to change statistical configurations of the system. We use Figure 6.2 to illustrate the delay distribution for an entire simulation, that is, from the first model iterate, until the end of the training phase. Every time the PS receives a contribution from a worker, it increments the count in the bin corresponding to the staleness of the stochastic gradient message. In the Figure we vary σ^2 to obtain different delay distributions. Using a larger variance induces very large delay values (long tail in the distribution). However, if we observe the average staleness $\bar{\tau}$, we notice it decreases, because the mass of the distribution is concentrated on the left, which in practice means the majority of workers experience small delays.

From an implementation point of view, our work is divided in two tasks: the simulation of the delays and the simulation of the distributed architecture. The delays simulation is simply a generation of a sequence of random variables according to a given distribution, while to simulate the distributed system it is sufficient allocate in memory $N_{worker} + 1$ versions of the model (workers and PS) and update them according to the order of arrival of messages. It is important to notice that the simulator is transparent to the underling mechanisms for computing gradients and updates. In this work, we use the PyTorch-1.4 library to describe models, and how they are trained. The interface between the high and low level layers allows exchanging model iterates, and gradients. Note that we use automatic differentiation from PyTorch-1.4 to compute gradients.

Overall, the above software design allows to: 1) easily introduce new models to study, as they can be readily imported from legacy PyTorch-1.4 code; 2) easily implement variants of stochastic optimization algorithms; 3) experiment with a variety of system configurations, including heterogeneity, various communication patterns, and various staleness

distributions.

6.6.1.0.4 Models and datasets.

We consider a classification task, where we train two convolutional neural network (CNN) variants of increasing model complexity, to gain insight on the role of sparsification for large deep network models. First, we study the behavior of LENET, using the MNIST dataset, then we move on to RESNET-56, using the CIFAR10 dataset. The model parameter and gradients dimensionality are approximately $d \in \{60K, 600K\}$ for LENET and RESNET-56, respectively. We use a training mini-batch size of 64 for MNIST and 128 for RESNET-56 and a testing mini-batch size of 128 samples. For each experimental setting, we run 5 simulations. The number of workers has been selected as the best in the set $\{1, 2, 4, 8, 16, 32, 64, 128\}$, $t_{\text{COMM}} \sim Exp\,(\lambda)$, $\ln(\lambda) \sim \mathcal{N}(0, \sigma^2)$, $\sigma^2 = \{0.1, 1, 3\}$.

η	ϕMEMSGD	ϕMEMSGD	ASGD
LENET	0.01	0.01	0.01
RESNET-56	0.1	0.1	0.1
RESNET-56 - 32 workers	0.005	0.005	0.005

Table 6.1 – Learning rate parameters.

momentum	ϕMEMSGD	ϕMEMSGD	ASGD
LENET	0.5	0.5	0.5
RESNET-56	0.9	0.9	0.9

Table 6.2 – Momentum parameters.

All model parameters we used, including learning rates and momentum, are specified in Table 6.1 and Table 6.2. Additional details are as follows.

- Training epochs = 5 (LENET), 161 (RESNET-56)
- Training mini-batch size = 64 (LENET), 128 (RESNET-56)
- Testing mini-batch size = 64 (LENET), 128 (RESNET-56)

The CNN models are implemented in PYTORCH-1.X.

- LENET: architecture, $d = 61706$
- RESNET-56: architecture, $d = 590426$

6.6.2 Comparative analysis

We compare ϕSGD and ϕMEMSGD with ASGD by measuring the test accuracy reached after a fixed number of epochs, which we set to 5 for LENET on MNIST and 161 for RESNET-56 on CIFAR10. We consider three scenarios, with 8 workers: each has a different delay distribution, given by the parameter σ^2, as shown in Figure 6.2. For sparsified methods, we use the best sparsification coefficient ρ. We discuss how ρ can be tuned in Section 6.6.3.

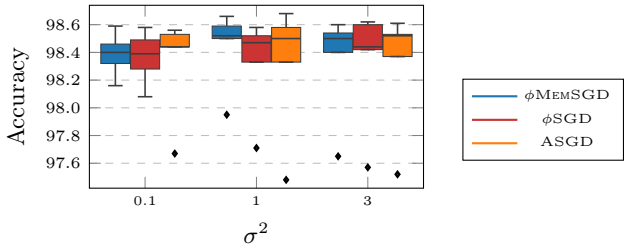


Figure 6.4 – Comparison of test accuracy of LeNet on Mnist, for three different asynchronous settings with 8 workers. In each setting we sample the exponential rates λ from a log-normal distribution with mean 0 and variance σ^2. For sparsified methods, the best ρ has been taken.

ϕMemSGD	ϕSGD	ASGD
87.05 ± 0.53	86.21 ± 1.06	85.90 ± 1.01

Table 6.3 – Comparison of test accuracy of ResNet-56 on Cifar10, with $\sigma^2 = 0.1$.

Figure 6.4 illustrates results obtained using a LeNet architecture with the Mnist dataset, while Table 6.3 reports the results obtained with ResNet-56 on Cifar10, fixing $\sigma^2 = 0.1$. Clearly, for both simple and deep models, the effects of sparsification on test accuracy are negligible. Given a reasonable choice of sparsification ρ, all variants achieve similar test accuracy using the same number of training iterations. Moreover, sparsified methods consistently achieve comparable performance to the non-sparse method irrespectively of the delay distribution. This confirms the result in Corollary 6.4.1, which indicates asymptotically vanishing effects of staleness on convergence (indeed, the term τ does not appear in the bound). We also observe that, as expected, the memory-based variant of sparsified SGD has an edge on the memory-less method, because it achieves better performance for lower values of ρ. This results clarifies the impact of memory-based error correction as a method to recover lost information due to aggressive sparsification.

Finally, note that we explicitly do not compare the methods using wall-clock times, as we are not interested in measuring the well-known benefits of sparsification in terms of reduced communication costs.

6.6.3 Tuning gradient sparsification

Using theorem 6.4.1 alone, it can be difficult to understand how ρ can be tuned. Next, we focus on the LeNet architecture using the Mnist, trained for 5 epochs, to understand how this affects the accuracy of sparsified SGD.

The results in Figure 6.5 show the impact of different values of the sparsification coefficient ρ on test accuracy. We notice a stark difference between ϕMemSGD and ϕSGD: the latter is much more sensitive to appropriate choices of ρ, and requires larger coefficients to achieve a reasonable test accuracy. For ϕMemSGD, instead, aggressive sparsification doesn't penalize performance noticeably, thanks to the memory mechanism. Then, even aggressive sparsification can be viable, as the cost per iteration (in terms of transmission times) decreases drastically, compared to standard SGD. Also, note that the top-k operator

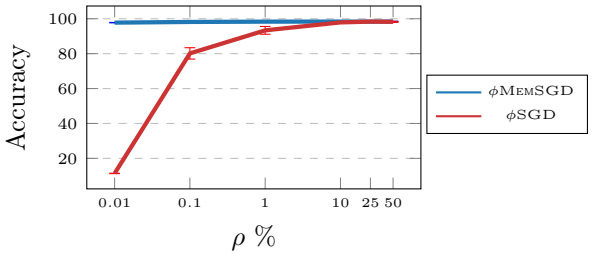


Figure 6.5 – Detailed study to understand how to tune the sparsification ρ , for LeNet on Mnist. Test accuracy as a function of ρ, in a system with 8 workers and $\sigma^2 = 0.1$

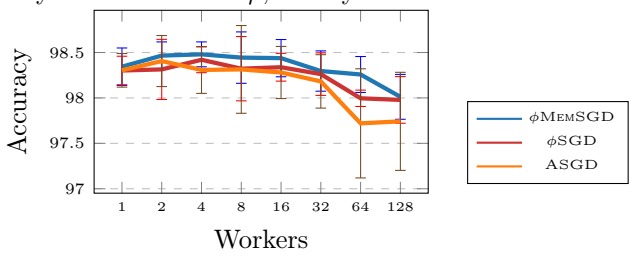


Figure 6.6 – Comparison of test accuracy, as a function of the number of workers. Results for LeNet on Mnist.

can be executed efficiently on GPUs [120], so that computational costs per iteration are equivalent to standard SGD.

6.6.4 Scalability

We now investigate how the three SGD variants scale with an increasing number of workers. As shown in Figure 6.6, all variants of SGD incur a slight drop in performance with more workers. Indeed, with more workers, the delay distribution in the system changes according to Figure 6.2, due to an increase in the probability of picking large delays. We note again that applying sparsification does not harm the performance of SGD, in that both ϕMemSGD and ϕSGD reach a comparable test accuracy with respect to ASGD. This is valid also for a deep convolutional network: for example, we run ResNet-56 on Cifar10 with 32 workers and we obtained a test accuracy of 85.30 ± 1.24 for ASGD and 86.01 ± 0.49 for ϕMemSGD with a sparsification coefficient $\rho = 1\%$.
Our results reinforce the message that sparsified SGD should be preferred over vanilla SGD, and this carries over to large scale scenarios in which, otherwise, excessively large message sizes could entail network congestion and jeopardize algorithmic efficiency.

6.7 Conclusion

In this work we focused on the role of sparsification methods applied to distributed stochastic optimization of non-convex loss functions, typically found in many modern machine learning settings.
For the first time, we provided a simple and concise analysis of the joint effects of asynchronicity and sparsification on mini-batch SGD, and showed that it converges asymptot-

ically as $\mathcal{O}\left(1/\sqrt{T}\right)$. Intuitively, top-$k$ sparsification restricts the path taken by model iterates in the optimization landscape to follow the principal components of a stochastic gradient update, as also noticed in [90].

We complemented our theoretical results with a thorough empirical campaign. Our experiments covered both variants of sparsified SGD, with and without memory, and compared them to standard SGD. We used a simple system simulator, which allowed us to explore scenarios with different delay distributions, as wel as an increasing number of workers.

Our results substantiated the theoretical findings of this work: the effects of staleness vanish asymptotically, and the impact of sparsification is negligible on convergence rate and test accuracy. We also studied how to appropriately chose the sparsification factor, and concluded that the memory mechanism applied to sparsified SGD allows to substantially sparsify gradients, save on communication costs, while obtaining comparable performance to standard SGD.

Our future plan is to establish a connection between gradient sparsification and recent studies showing that the landscape of the loss surface of deep models is typically sparse [121, 102, 122]. In light of such works, [123] suggests that sparsification can be directly applied to model parameters, albeit training requires multiple stages. Gradient sparsification could then be studied as a mechanism to favor model compression at training time.

7

Isotropic SGD: Practical Bayesian Posterior Sampling

Despite mathematical elegance and some promising results restricted to simple models, standard stochastic gradient (SG) methods for MCMC-based Bayesian posterior sampling [16, 100, 124, 17, 101] fall short in easily dealing with the complexity of the loss landscape of deep models [102, 103], for which stochastic optimization poses serious challenges [125, 126]. Existing methods are often unpractical, as they require ad-hoc, sophisticated vanishing learning rate schedules, and hyper-parameter tuning.

In general, SG-MCMC algorithms inject random noise to SG descent algorithms: the covariance of such noise and the learning rate are tightly related to the assumptions on the loss landscape, which together with the SG noise, determine their sampling properties [101]. Current SG-MCMC algorithms applied to popular complex models such as Deep Nets, cannot satisfy the simplifying assumptions on loss landscapes and on the behavior of the SG noise covariances, while operating with practical requirements, such as non-vanishing learning rates and ease of use. A recent work [127] argues for fixed step sizes, but settles for variational approximations of simple quadratic losses. In a generic Bayesian deep learning setting, none of the existing implementations of SG-MCMC methods converge to the true posterior without learning rate annealing.

While we are not the first to highlight these issues [102, 103], we believe that studying the role of noise in SG-MCMC algorithms has not received enough attention, and a deeper understanding is truly desirable, as it can clarify how various methods compare. We adapt the unified notation framework of [101] to understand the role of noise magnitude from a practitioner point of view. This framework suggests us to propose a simple, yet unexplored, SG-MCMC algorithm relying on fewer tunable parameters and less restrictive assumptions. Our goal is to study under which algorithmic conditions we can relax the assumptions while still having theoretical guarantees.

In this chapter, we make use of a mathematical framework [128] that emphasizes the role of SG noise covariance and learning rate on the behavior of SG-MCMC algorithms (Sec. 7.1). As a result, the equivalence between learning rate annealing and the injection of noise with extremely large variance becomes clear. This intuition, together with the technical condi-

tions described by Theorem 1, allows us to propose a novel, practical SG-MCMC algorithm (Sec. 7.4) that produces (approximate) posterior samples with a fixed, analytically derived, learning rate. In the considered algorithm, a non annealed learning rate is possible thanks to the choice of making the SG noise isotropic, a finite identity covariance matrix, instead of the usual, approaching to infinity one. The proposed SG-MCMC method is a valid, theoretically sound, and simple alternative to popular techniques, that have shortcomings when it comes to the assumptions they rely on [74].

We furthermore address a concern that has emerged only very recently in the stochastic optimization community for deep complex models: the SG noise is not Gaussianly distributed but follows an heavy tailed distribution [129, 130]. As carefully explained in Sec. 7.4, we incorporate this knowledge in a simple way in our Bayesian sampling scheme and show that this incorporation helps us improve performance.

We evaluate SG-MCMC algorithms (Sec. 7.6) through an extensive experimental campaign, where we compare our approach to a number of alternatives, including Monte Carlo Dropout (MCD) [74] and Stochastic Weighted Averaging Gaussian (SWAG) [126], which have been successfully applied to the Bayesian deep learning setting. Our results indicate that our approach is competitive to the state-of-the-art, in terms of accuracy and uncertainty, and its analytically derived (constant) learning rate make it a worthy unexplored alternative in the panorama of SG-MCMC methods.

7.1 MCMC Through the Lenses of Langevin Dynamics

Consider a data-set of $m-$dimensional observations $\mathcal{D} = \{\boldsymbol{U}_i\}_{i=1}^N$ and a statistical model defined through a likelihood function $p(\mathcal{D}|\boldsymbol{\theta})$ parameterized through a d-dimensional set of parameters $\boldsymbol{\theta}$. Given a prior $p(\boldsymbol{\theta})$ the posterior over the parameters is obtained by means of Bayes theorem as:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\mathcal{D})}, \tag{7.1}$$

where $p(\mathcal{D})$ is also known as the model evidence, defined as the integral $p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})\, p(\boldsymbol{\theta}) d\boldsymbol{\theta}$. The posterior distribution is necessary in order to obtain predictive distributions for new test observations $\boldsymbol{U}_*$: $p(\boldsymbol{U}_*|\mathcal{D}) = \int p(\boldsymbol{U}_*|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}$. We focus on Monte Carlo methods to obtain an estimate of this predictive distribution, by averaging over N_{MC} samples obtained from the posterior over $\boldsymbol{\theta}$, that is $\boldsymbol{\theta}^{(i)} \sim p(\boldsymbol{\theta}|\mathcal{D})$:

$$p(\boldsymbol{U}_*|\mathcal{D}) \approx \frac{1}{N_{\text{MC}}} \sum_{i=1}^{N_{\text{MC}}} p(\boldsymbol{U}_*|\boldsymbol{\theta}^{(i)}). \tag{7.2}$$

Since Eq. (7.1) is analytically intractable [2], unless the prior is conjugate to the likelihood, we work with an unnormalized version of the logarithm of the posterior density, and express

the negative logarithm of the joint distribution of the data-set $\mathcal{D}$ and parameters $\boldsymbol{\theta}$ as:

$$-f(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{U}_i|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{7.3}$$

For computational efficiency, we use a mini-batch stochastic gradient $\boldsymbol{g}(\boldsymbol{\theta})$, which guarantees that the estimated gradient is an unbiased estimate of the true gradient $\nabla f(\boldsymbol{\theta})$, and we assume that the randomness due to the mini-batch introduces a Gaussian noise:[1] $\boldsymbol{g}(\boldsymbol{\theta}) \sim N(\nabla f(\boldsymbol{\theta}), 2\boldsymbol{B}(\boldsymbol{\theta}))$, where the matrix $\boldsymbol{B}(\boldsymbol{\theta})$ denotes the SG noise covariance, which depends on the parametric model, the data distribution and the mini-batch size.

Mini-batch gradient approximation Starting from the gradient of the logarithm of the posterior density:

$$-\nabla f(\boldsymbol{\theta}) = \sum_{i=1}^{N} \nabla \log p(\boldsymbol{U}_i|\boldsymbol{\theta}) + \nabla \log p(\boldsymbol{\theta}),$$

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 $\boldsymbol{g}(\boldsymbol{\theta})$ is computed as

$$-\boldsymbol{g}(\boldsymbol{\theta}) = \frac{N}{N_b} \sum_{i=1}^{N_b} \nabla \log p(\boldsymbol{U}_i|\boldsymbol{\theta}) + \nabla \log p(\boldsymbol{\theta}),$$

By simple calculations it is possible to show that the estimation is unbiased ($E(\boldsymbol{g}(\boldsymbol{\theta})) = \nabla f(\boldsymbol{\theta})$). The estimation error covariance is defined to be $E\left[(\boldsymbol{g}(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta}))\,(\boldsymbol{g}(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta}))^{\top}\right] = 2\boldsymbol{B}(\boldsymbol{\theta})$.

If the minibatch size is large enough, invoking the central limit theorem, we can state that the minibatch gradient is normally distributed:

$$\boldsymbol{g}(\boldsymbol{\theta}) \sim N(\nabla f(\boldsymbol{\theta}), 2\boldsymbol{B}(\boldsymbol{\theta})).$$

A survey of algorithms to sample from the posterior using SG methods can be found in [101]. As shown in the literature [131, 127], 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 define a unified notation to compare many existing algorithms in light of the role played by their noise components.

Stochastic gradient descent (SGD) can be studied through the following stochastic differential equation (SDE) [132, 133, 134], when the learning rate η is small enough:

$$d\boldsymbol{z}_t = \boldsymbol{s}(\boldsymbol{z}_t)dt + \sqrt{2\eta \boldsymbol{D}(\boldsymbol{z}_t)}d\boldsymbol{W}_t. \tag{7.4}$$

Here we use a generic form of the SDE, with variable $\boldsymbol{z}$ instead of $\boldsymbol{\theta}$, that accommodates SGD variants, with and without momentum. It is typical SG-MCMC practice [16, 100, 124, 127]

[1] This is an ordinary assumption in the literature. In light of recent work [129, 130], we will address the case for an α-stable SG noise distribution in Section 7.4.1.

to allow the stochastic process induced by Eq. (7.4) to go through an initial adaptation phase where the learning rate is annealed, followed by fixing the learning rate to a small value, to ensure the process reaches and maintains a stationary distribution thereafter.

Definition 1 *A distribution $\rho(\boldsymbol{z}) \propto \exp(-\phi(\boldsymbol{z}))$ is said to be a **stationary** distribution for the* SDE *of the form* (7.4)*, if and only if it satisfies the following Fokker-Planck equation (*FPE*):*

$$0 = \mathrm{Tr}\left\{\nabla\left[-\boldsymbol{s}(\boldsymbol{z})^\top \rho(\boldsymbol{z}) + \eta\nabla^\top\left(\boldsymbol{D}\left(\boldsymbol{z}\right)\rho(\boldsymbol{z})\right)\right]\right\}. \tag{7.5}$$

Note that, the operator $\nabla^\top$ applied to matrix $\boldsymbol{D}(\boldsymbol{z})$ produces a row vector whose elements are the divergences of the $\boldsymbol{D}(\boldsymbol{z})$ columns [17].

In general, the stationary distribution does not converge to the desired posterior distribution, i.e., $\phi(\boldsymbol{z}) \neq f(\boldsymbol{z})$, as shown by [125]. Additionally, given an initial condition for $\boldsymbol{z}_t$, its distribution is going to converge to $\rho(\boldsymbol{z})$ only for $t \to \infty$.

Next, we revisit known approaches to Bayesian posterior sampling, and interpret them as variants of an SGD process, using the FPE formalism. In what follows, we use n to indicate discrete time, and t for continuous time.

Gradient methods without momentum. The generalized update rule of SGD, described as a discrete-time stochastic process, writes as:

$$\boldsymbol{\delta\theta}_n = -\eta\boldsymbol{P}(\boldsymbol{\theta}_{n-1})(\boldsymbol{g}(\boldsymbol{\theta}_{n-1}) + \boldsymbol{w}_n), \tag{7.6}$$

where $\boldsymbol{P}(\boldsymbol{\theta}_{n-1})$ is a user-defined preconditioning matrix, and $\boldsymbol{w}_n$ is a noise term, distributed as $\boldsymbol{w}_n \sim N(\boldsymbol{0}, 2\boldsymbol{C}(\boldsymbol{\theta}_n))$, with a user-defined covariance matrix $\boldsymbol{C}(\boldsymbol{\theta}_n)$. Then, the corresponding continuous-time SDE is [132]:

$$d\boldsymbol{\theta}_t = -\boldsymbol{P}(\boldsymbol{\theta}_t)\nabla f(\boldsymbol{\theta}_t)dt + \sqrt{2\eta\boldsymbol{P}(\boldsymbol{\theta}_t)^2\boldsymbol{\Sigma}(\boldsymbol{\theta}_t)}d\boldsymbol{W}_t. \tag{7.7}$$

The derivation is as follows. Since $\boldsymbol{g}(\boldsymbol{\theta}_{n-1}) \sim N(\nabla f(\boldsymbol{\theta}_{n-1}), 2\boldsymbol{B}(\boldsymbol{\theta}_{n-1}))$ we can rewrite the Equation (7.6) as:

$$\boldsymbol{\delta\theta}_n = -\eta\boldsymbol{P}(\boldsymbol{\theta}_{n-1})(\nabla f(\boldsymbol{\theta}_{n-1}) + \boldsymbol{w}_n^{'}),$$

where $\boldsymbol{w}_n^{'} \sim N(0, 2\boldsymbol{\Sigma}(\boldsymbol{\theta}_{n-1}))$. If we separate deterministic and random component we can equivalently write:

$$\boldsymbol{\delta\theta}_n = -\eta\boldsymbol{P}(\boldsymbol{\theta}_{n-1})\nabla f(\boldsymbol{\theta}_{n-1}) + \eta\boldsymbol{P}(\boldsymbol{\theta}_{n-1})\boldsymbol{w}_n^{'} = -\eta\boldsymbol{P}(\boldsymbol{\theta}_{n-1})\nabla f(\boldsymbol{\theta}_{n-1}) + \sqrt{2\eta\boldsymbol{P}^2(\boldsymbol{\theta}_{n-1})\boldsymbol{\Sigma}(\boldsymbol{\theta}_{n-1})}\boldsymbol{v}_n$$

where $\boldsymbol{v}_n \sim N(0, \eta E)$. When η is small enough ($\eta \to dt$) we can interpret the above equation as the discrete time simulation of the following SDE [132]:

$$d\boldsymbol{\theta}_t = -\boldsymbol{P}(\boldsymbol{\theta}_t)\nabla f(\boldsymbol{\theta}_t)dt + \sqrt{2\eta\boldsymbol{P}(\boldsymbol{\theta}_t)^2\boldsymbol{\Sigma}(\boldsymbol{\theta}_t)}d\boldsymbol{W}_t,$$

where $d\boldsymbol{W}_t$ is a $d-$dimensional Brownian motion.
We denote by $\boldsymbol{C}(\boldsymbol{\theta})$ the covariance of the *injected noise* and by $\boldsymbol{\Sigma}(\boldsymbol{\theta}_t) = \boldsymbol{B}(\boldsymbol{\theta}_t) + \boldsymbol{C}(\boldsymbol{\theta}_t)$ the *composite noise* covariance, which combines the SG and the injected noise.
We define the stationary distribution of the SDE in Equation (7.7) as $\rho(\boldsymbol{\theta}) \propto \exp(-\phi(\boldsymbol{\theta}))$, noting that when $\boldsymbol{C} = \mathbf{0}$, the potential $\phi(\boldsymbol{\theta})$ differs from the desired posterior $f(\boldsymbol{\theta})$ [125]. 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* (7.7) *and define the stationary distribution* $\rho(\boldsymbol{\theta}) \propto \exp(-\phi(\boldsymbol{\theta}))$. *If* $\nabla^\top \left(\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\right) = \mathbf{0}^\top$ *and* $\eta \boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}$, *then* $\phi(\boldsymbol{\theta}) = f(\boldsymbol{\theta})$.

The proof of Theorem 1 is in Appendix B.0.1.
SGLD [16] is a simple approach to satisfy Theorem 1; it uses no preconditioning, $\boldsymbol{P}(\boldsymbol{\theta}) = E$, and sets the injected noise covariance to $\boldsymbol{C}(\boldsymbol{\theta}) = \eta^{-1}E$. In the limit for $\eta \to 0$, it holds that $\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \boldsymbol{B}(\boldsymbol{\theta}) + \eta^{-1}E \simeq \eta^{-1}E$. Then, $\nabla^\top \left(\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\right) = \eta \nabla^\top E = \mathbf{0}^\top$, and $\eta \boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}$. While SGLD succeeds in (asymptotically) generating samples from the true posterior, its mixing rate is unnecessarily slow, due to the extremely small learning rate [100].
An extension to SGLD is Stochastic Gradient Fisher Scoring (SGFS) [100], 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, $\boldsymbol{P}(\boldsymbol{\theta}) \propto \boldsymbol{B}(\boldsymbol{\theta})^{-1}$. Generally, however, $\boldsymbol{B}(\boldsymbol{\theta})$ is ill conditioned: many of its eigenvalues are almost zero [125], and computing $\boldsymbol{B}(\boldsymbol{\theta})^{-1}$ is problematic. Moreover, using a non-vanishing learning rate, the conditions of Theorem 1 are met only if, at convergence, $\nabla^\top(\boldsymbol{B}(\boldsymbol{\theta})^{-1}) = \mathbf{0}^\top$, which would be trivially true if $\boldsymbol{B}(\boldsymbol{\theta})$ was constant. However, recent work [102, 103] suggests that this condition is difficult to justify.
The Stochastic Gradient Riemannian Langevin Dynamics (SGRLD) algorithm [124] extends SGFS to the setting in which $\nabla^\top(\boldsymbol{B}(\boldsymbol{\theta})^{-1}) \neq \mathbf{0}^\top$. The process dynamics are adjusted by adding the term $\nabla^\top(\boldsymbol{B}(\boldsymbol{\theta})^{-1})$ which, however, cannot be easily estimated, restricting SGRLD to cases where it can be computed analytically.
The work in [135] considers a regularized diagonal preconditioning matrix $\boldsymbol{P}(\boldsymbol{\theta})$, derived from the SG noise. However, in the sampling phase, the method neglects the term $\nabla^\top \boldsymbol{P}(\boldsymbol{\theta})$ and the regularization term is a user-defined parameter, which is not trivial to tune.
The approach in [127] investigates constant-rate SGD (with no injected noise), and determines analytically the learning rate and preconditioning that minimize 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 $\boldsymbol{B}(\boldsymbol{\theta})$ 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 geometry.

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

$$\begin{cases} \boldsymbol{\delta\theta}_n = \eta \boldsymbol{P}(\boldsymbol{\theta}_{n-1}) \boldsymbol{M}^{-1} \boldsymbol{r}_{n-1} \\ \boldsymbol{\delta r}_n = -\eta \boldsymbol{A}(\boldsymbol{\theta}_{n-1}) \boldsymbol{M}^{-1} \boldsymbol{r}_{n-1} - \eta \boldsymbol{P}(\boldsymbol{\theta}_{n-1})(\boldsymbol{g}(\boldsymbol{\theta}_{n-1}) + \boldsymbol{w}_n), \end{cases} \tag{7.8}$$

where $\boldsymbol{P}(\boldsymbol{\theta}_{n-1})$ is a preconditioning matrix, $\boldsymbol{M}$ is the mass matrix and $\boldsymbol{A}(\boldsymbol{\theta}_{n-1})$ is the friction matrix [17, 136]. Similarly to the first order counterpart, the noise term is distributed as $\boldsymbol{w}_n \sim N(\boldsymbol{0}, 2\boldsymbol{C}(\boldsymbol{\theta}_n)))$. Then, the SDE to describe continuous-time system dynamics is:

$$\begin{cases} d\boldsymbol{\theta}_t = \boldsymbol{P}(\boldsymbol{\theta}_t) \boldsymbol{M}^{-1} \boldsymbol{r}_t dt \\ d\boldsymbol{r}_t = -(A(\boldsymbol{\theta}_t) \boldsymbol{M}^{-1} r_t + \boldsymbol{P}(\boldsymbol{\theta}_t) \nabla f(\boldsymbol{\theta}_t)) dt + \sqrt{2\eta \boldsymbol{P}(\boldsymbol{\theta}_t)^2 \boldsymbol{\Sigma}(\boldsymbol{\theta}_t)} d\boldsymbol{W}_t. \end{cases} \tag{7.9}$$

where $\boldsymbol{P}(\boldsymbol{\theta}_t)^2 = \boldsymbol{P}(\boldsymbol{\theta}_t)\boldsymbol{P}(\boldsymbol{\theta}_t)$, and $\boldsymbol{P}(\boldsymbol{\theta}_t)$ is symmetric.
The derivation is as follows. Similarly to the case without momentum, we rewrite the second equation in 7.8 as

$$\boldsymbol{\delta r}_n = -\eta \boldsymbol{A}(\boldsymbol{\theta}_{n-1}) \boldsymbol{M}^{-1} \boldsymbol{r}_{n-1} - \eta \boldsymbol{P}(\boldsymbol{\theta}_{n-1})(\boldsymbol{g}(\boldsymbol{\theta}_{n-1}) + \boldsymbol{w}_n) = -\eta \boldsymbol{A}(\boldsymbol{\theta}_{n-1}) \boldsymbol{M}^{-1} \boldsymbol{r}_{n-1} - \eta \boldsymbol{P}(\boldsymbol{\theta}_{n-1}) \nabla f(\boldsymbol{\theta}_{n-1}) + \sqrt{2\eta \boldsymbol{P}^2(\boldsymbol{\theta}_{n-1}) \boldsymbol{\Sigma}(\boldsymbol{\theta}_{n-1})} \boldsymbol{v}_n$$

where again $\boldsymbol{v}_n \sim N(0, \eta E)$. If we define the super-variable $\boldsymbol{z} = [\boldsymbol{\theta}, \boldsymbol{r}]^\top$, we can rewrite the system as:

$$\boldsymbol{\delta z}_n = -\eta \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}_{n-1}) \\ \boldsymbol{P}(\boldsymbol{\theta}_{n-1}) & \boldsymbol{A}(\boldsymbol{\theta}_{n-1}) \end{bmatrix} \boldsymbol{s}(\boldsymbol{z}_{n-1}) + \sqrt{2\eta \boldsymbol{D}(\boldsymbol{z}_{n-1})} \boldsymbol{\nu}_n$$

where $\boldsymbol{s}(\boldsymbol{z}) = \begin{bmatrix} \nabla f(\boldsymbol{\theta}) \\ \boldsymbol{M}^{-1} \boldsymbol{r} \end{bmatrix}$, $\boldsymbol{D}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P}(\boldsymbol{\theta})^2 \boldsymbol{\Sigma}(\boldsymbol{\theta}) \end{bmatrix}$ and $\boldsymbol{\nu}_n \sim N(0, \sqrt{\eta} E)$.
As the learning rate goes to zero ($\eta \to dt$), similarly to the previous case, we can interpret the above difference equation as a discretization of the following FPE

$$d\boldsymbol{z}_t = -\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}_t) \\ \boldsymbol{P}(\boldsymbol{\theta}_t) & \boldsymbol{A}(\boldsymbol{\theta}_t) \end{bmatrix} \boldsymbol{s}(\boldsymbol{z}_t) + \sqrt{2\eta \boldsymbol{D}(\boldsymbol{z}_t)} d\boldsymbol{W}_t,$$

which corresponds to Equation (7.9)
The theorem hereafter describes the conditions for which the SDE converges to the true posterior distribution.

Theorem 2 *Consider dynamics of the form* (7.9) *and define the stationary distribution for* $\boldsymbol{\theta}_t$ *as* $\rho(\boldsymbol{\theta}) \propto \exp(-\phi(\boldsymbol{\theta}))$. *If* $\nabla^\top \boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{0}^\top$ *and* $\boldsymbol{A}(\boldsymbol{\theta}) = \eta \boldsymbol{P}(\boldsymbol{\theta})^2 \boldsymbol{\Sigma}(\boldsymbol{\theta})$, *then* $\phi(\boldsymbol{\theta}) = f(\boldsymbol{\theta})$.

The proof of Theorem 2 is in Appendix B.0.2.
In the naive case, where $\boldsymbol{P}(\boldsymbol{\theta}) = E, \boldsymbol{A}(\boldsymbol{\theta}) = \boldsymbol{0}, \boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{0}$, the conditions in Theorem 2

are not satisfied and the stationary distribution does not correspond to the true posterior [17]. To generate samples from the true posterior, it is sufficient to set $\boldsymbol{P}(\boldsymbol{\theta}) = E, \boldsymbol{A}(\boldsymbol{\theta}) = \eta\boldsymbol{B}(\boldsymbol{\theta}), \boldsymbol{C}(\boldsymbol{\theta}) = \mathbf{0}$ (as in Eq. (9) in [17]).

Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) [17] suggests that estimating $\boldsymbol{B}(\boldsymbol{\theta})$ can be costly. Hence, the injected noise $\boldsymbol{C}(\boldsymbol{\theta})$ is chosen such that $\boldsymbol{C}(\boldsymbol{\theta}) = \eta^{-1}\boldsymbol{A}(\boldsymbol{\theta})$, where $\boldsymbol{A}(\boldsymbol{\theta})$ is user-defined. When $\eta \to 0$, the following approximation holds: $\boldsymbol{\Sigma}(\boldsymbol{\theta}) \simeq \boldsymbol{C}(\boldsymbol{\theta})$. It is then trivial to check that conditions in Theorem 2 hold without the need for explicitly estimating $\boldsymbol{B}(\boldsymbol{\theta})$. A further practical reason to avoid setting $\boldsymbol{A}(\boldsymbol{\theta}) = \eta\boldsymbol{B}(\boldsymbol{\theta})$ is that the computational cost for the operation $\boldsymbol{A}(\boldsymbol{\theta}_{n-1})\boldsymbol{M}^{-1}\boldsymbol{r}_{n-1}$ has $\mathcal{O}(D^2)$ complexity, whereas if $\boldsymbol{C}(\boldsymbol{\theta})$ 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 [101]), which considers a generic, space-varying preconditioning matrix $\boldsymbol{P}(\boldsymbol{\theta})$ derived from information geometric arguments [137]. SGRHMC suggests to set $\boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{G}(\boldsymbol{\theta})^{-\frac{1}{2}}$, where $\boldsymbol{G}(\boldsymbol{\theta})$ is the Fisher Information matrix. To meet the requirement $\nabla^\top \boldsymbol{P}(\boldsymbol{\theta}) = \mathbf{0}^\top$, it includes a correction term, $-\nabla^\top \boldsymbol{P}(\boldsymbol{\theta})$. The injected noise is set to $\boldsymbol{C}(\boldsymbol{\theta}) = \eta^{-1}E - \boldsymbol{B}(\boldsymbol{\theta})$, consequently $\boldsymbol{\Sigma} = \eta^{-1}E$, and the friction matrix is set to $\boldsymbol{A}(\boldsymbol{\theta}) = \boldsymbol{P}(\boldsymbol{\theta})^2$. With all these choices, Theorem 2 is satisfied. While appealing, the main drawbacks of this method are the need for an analytical expression of $\nabla^\top \boldsymbol{P}(\boldsymbol{\theta})$, and the assumption for $\boldsymbol{B}(\boldsymbol{\theta})$ to be known.

Recently, the work in [138] suggests to use a cyclical learning rate schedule to better explore the loss landscape and sample more efficiently. While the idea is appealing, it introduces further hyper-parameters to tune, which is opposite to our quest for a simple, easy to use method.

From a practical standpoint, momentum-based methods suffer from the complexity and fragility of hyper-parameter tuning, including the learning rate schedule and those that govern the simulation of a second-order Langevin dynamics. The method we propose in this work can be applied to momentum-based algorithms as well; then, it can be viewed as an extension to the work in [139], albeit addressing more complex loss landscapes. However, we leave this avenue of research for future work.

7.2 Bayesian Posterior Sampling by Layer-wise Isotropization

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, observe that $\boldsymbol{P}(\boldsymbol{\theta}), \boldsymbol{\Sigma}(\boldsymbol{\theta})$ are instrumental to determine the convergence properties of SG methods to the true posterior. We consider the constructive approach of

Algorithm 1: Idealized posterior sampling

SAMPLE $(\boldsymbol{\theta}_0, \boldsymbol{B}(\boldsymbol{\theta}), \boldsymbol{\Lambda})$:
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$ ▷ Initialize $\boldsymbol{\theta}_t$
loop
  $\boldsymbol{g} = \frac{\nabla \tilde{f}(\boldsymbol{\theta})}{N}$ ▷ Compute SG
  $\boldsymbol{C}(\boldsymbol{\theta})^{\frac{1}{2}} \leftarrow (\boldsymbol{\Sigma} - \boldsymbol{B}(\boldsymbol{\theta}))^{\frac{1}{2}}$
  $\boldsymbol{n} \sim N(0, \boldsymbol{I})$
  $\boldsymbol{w} \leftarrow \boldsymbol{C}(\boldsymbol{\theta})^{\frac{1}{2}} \boldsymbol{n}$
  $\delta\boldsymbol{\theta} \leftarrow (N\boldsymbol{\Sigma})^{-1}(\boldsymbol{g} + \sqrt{2}\boldsymbol{w})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \delta\boldsymbol{\theta}$ ▷ Update $\boldsymbol{\theta}$

Table 7.1 – Idealized posterior sampling

designing $\eta\boldsymbol{P}(\boldsymbol{\theta})$ to be a constant, diagonal matrix, constrained to be layer-wise uniform:

$$\eta\boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{\Lambda}^{-1} = \mathrm{diag}([\underbrace{\lambda^{(1)}, \dots, \lambda^{(1)}}_{\text{layer } 1}, \dots, \underbrace{\lambda^{(N_l)}, \dots \lambda^{(N_l)}}_{\text{layer } N_l}])^{-1}. \tag{7.10}$$

Notice that we treat biases and weights of the layers as unique groups of parameters. By properly setting parameters $\lambda^{(p)}$, we achieve the simultaneous result of a non-vanishing learning rate and a well-conditioned preconditioning matrix. This implies a layer-wise learning rate $\eta^{(p)} = \frac{1}{\lambda^{(p)}}$ for the p-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 7.2.1 *Given the dynamics of Eq.* (7.7) *and the stationary distribution* $\rho(\boldsymbol{\theta}) \propto \exp(-\phi(\boldsymbol{\theta}))$*, if* $\eta\boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{\Lambda}^{-1}$ *as in Eq.* (7.10)*, and* $\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{\Lambda} - \boldsymbol{B}(\boldsymbol{\theta}) \succ \boldsymbol{0} \, \forall\boldsymbol{\theta}$*, i.e. it is positive definite, then* $\phi(\boldsymbol{\theta}) = f(\boldsymbol{\theta})$*.*

The requirement $\boldsymbol{C}(\boldsymbol{\theta}) \succ \boldsymbol{0} \quad \forall\boldsymbol{\theta}$, ensures that the injected noise covariance is valid. The composite noise matrix is equal to $\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \boldsymbol{\Lambda}$. Since $\nabla^\top \boldsymbol{\Sigma}(\boldsymbol{\theta}) = \nabla^\top \boldsymbol{\Lambda} = \boldsymbol{0}$ and $\eta\boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{\Lambda}^{-1}$ by construction, then Theorem 1 is satisfied.
If the above conditions hold, it is simple to show that matrices $\boldsymbol{P}(\boldsymbol{\theta})$ and $\boldsymbol{C}(\boldsymbol{\theta})$ satisfy Theorem 1. Then, we say that the composite noise covariance $\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \boldsymbol{C}(\boldsymbol{\theta}) + \boldsymbol{B}(\boldsymbol{\theta}) = \mathrm{diag}\left(\left[\lambda^{(1)}, \dots, \lambda^{(1)}, \dots, \lambda^{(N_l)}, \dots \lambda^{(N_l)}\right]\right)$ is *isotropic* within model layers. We set $\boldsymbol{\Lambda}$ to be, among all valid matrices satisfying $\boldsymbol{\Lambda} - \boldsymbol{B}(\boldsymbol{\theta}) \succ \boldsymbol{0}$, the smallest, i.e., the one with the smallest λ's. Indeed, larger $\boldsymbol{\Lambda}$ induces a small learning rate, thus unnecessarily reducing sampling speed.

7.3 An ideal method.

Now, let's consider an ideal case, in which we assume the SG noise covariance $\boldsymbol{B}(\boldsymbol{\theta})$ and $\boldsymbol{\Lambda}$ to be known in advance. The procedure described in Algorithm 7.1 illustrates a naive SG method that uses the *injected noise* covariance $\boldsymbol{C}(\boldsymbol{\theta})$ to sample from the true posterior.
This deceivingly 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 $f(\boldsymbol{\theta})$,

we compute the (mini-batch) gradient of $\frac{f(\boldsymbol{\theta})}{N}$, similarly to the notation used in [127], that we indicate with $\frac{\nabla \tilde{f}(\boldsymbol{\theta})}{N}$. However, such method cannot be used in practice as $\boldsymbol{B}(\boldsymbol{\theta})$ and $\boldsymbol{\Lambda}$ are unknown. Also, the algorithm is costly, as it requires computing $(\boldsymbol{\Sigma} - \boldsymbol{B}(\boldsymbol{\theta}))^{\frac{1}{2}}$, which requires $\mathcal{O}(d^3)$ operations, and $\boldsymbol{C}(\boldsymbol{\theta})^{\frac{1}{2}}$, 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 [127] suggests to explore a related preconditioning, but does not develop the idea.

7.4 A practical method: Isotropic SGD.

To make the idealized 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 $\boldsymbol{B}(\boldsymbol{\theta})$ can be approximated with a diagonal matrix, i.e., $\boldsymbol{B}(\boldsymbol{\theta}) = \mathrm{diag}(\boldsymbol{b}(\boldsymbol{\theta}))$. Thus, the noise components are independent [139, 100].*

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 [139, 140]. Hence, combining with Assumption 1, $\boldsymbol{b}(\boldsymbol{\theta}) \simeq \frac{\mathrm{E}[\boldsymbol{g}(\boldsymbol{\theta}) \odot \boldsymbol{g}(\boldsymbol{\theta})]}{2}$ (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 [141]): $\beta^{(p)} = \max_{j \in I_p} \mathbf{b}_j(\boldsymbol{\theta})$, where I_p is the set of indexes of parameters belonging to p_{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) $\boldsymbol{\Lambda} = \left[\lambda^{(1)}, \dots, \lambda^{(1)}, \dots, \lambda^{(N_l)}, \dots \lambda^{(N_l)}\right]$ satisfying Corollary 7.2.1 can be obtained as $\lambda^{(p)} = \beta^{(p)}$. The proof is reported hereafter.
By the assumptions the matrix $\boldsymbol{B}(\boldsymbol{\theta})$ is diagonal, and consequently $\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{\Lambda} - \boldsymbol{B}(\boldsymbol{\theta})$ is diagonal as well. The preconditioner $\boldsymbol{\Lambda}$ must be chosen to satisfy the positive semi-definite constraint, i.e. $\boldsymbol{C}(\boldsymbol{\theta})_{ii} \geq 0 \quad \forall i, \forall \boldsymbol{\theta}$. Equivalently, we must satisfy $\lambda^{(p)} - \mathbf{b}_j(\boldsymbol{\theta}) \geq 0 \quad \forall j \in I_p, \forall p, \forall \boldsymbol{\theta}$, where I_p is the set of indexes of parameters belonging to p_{th} layer. By

Algorithm 2: I-SGD: practical sampling

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

Table 7.2 – Practical sampling

Assumption 3, i.e. $\beta^{(p)} = \max_{k \in I_p} \mathbf{b}_k(\boldsymbol{\theta})$, to satisfy the positive semi-definite requirement in all cases the minimum valid set of $\lambda^{(p)}$ is determined as $\lambda^{(p)} = \beta^{(p)}$.

Since we cannot assume $\boldsymbol{B}(\boldsymbol{\theta})$ to be known, in what follows we discuss two approaches to estimate its components. A simple method to estimate $\boldsymbol{B}(\boldsymbol{\theta})$ is as follows (see Section 7.4.1): we compute $\lambda^{(p)} = \max_{j \in I_p} \mathbf{b}_j(\boldsymbol{\theta}) = \frac{1}{2}\max_j(\boldsymbol{g}_j(\boldsymbol{\theta})^{(p)})^2$,where $\boldsymbol{g}(\boldsymbol{\theta})^{(p)}$ is the portion of stochastic gradient corresponding to the p-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 [129, 130] (see Section 7.4.1 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: $w \sim p(w)$, where $\int_{-\infty}^{+\infty} \exp(j2\pi wt)\, p(w)dw = \exp(-|ct|^\alpha)$, with $\alpha \in (0, 2]$, where α, c can vary across different parameters. When $\alpha = 2$, $p(w)$ becomes Gaussian, but for $\alpha < 2$, its variance goes to infinity, thus estimating the SG noise covariance is problematic.

Prior works [142] 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 minimize the l_2-distance between $p(w) = (2\pi\tilde{\sigma}^2)^{-\frac{1}{2}} \exp\left(-\frac{w^2}{2\tilde{\sigma}^2}\right)$ and $q(w) = \int_{-\infty}^{+\infty} \exp(-j2\pi wt)\exp(-|ct|^\alpha)dt$ (see Section 7.4.1 for details). We estimate the parameters of the α-stable distribution by extending [143] to space varying settings, and derive the equivalent variances $\tilde{\sigma}^2$ that minimize the l_2 distance. Then, the $\lambda^{(p)}$ are computed as $\frac{1}{2}\max_j(\tilde{\boldsymbol{\sigma}}_j(\boldsymbol{\theta})^{(p)})^2$.

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

The practical implementation of I-SGD is shown in Algorithm 7.2.

The computational cost of I-SGD is as follows. Similarly to [17], we define the cost of computing a gradient mini-batch as $C_{\boldsymbol{g}}(N_b, d)$, where, as in the previous Chapter, d is the dimensionality of the parameter vector. Then (see Section 7.4.1), 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, SGFS and their variants: it scales as $\mathcal{O}(N_{\text{MC}}d)$, where N_{MC} is the number of posterior samples.

7.4.1 Estimation of $\lambda^{(p)}$

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).

7.4.1.1 The case of Gaussian noise.

We here give additional details on the estimation of $\lambda^{(p)}$. The simple and naive estimation described in the thesis is the following: $\lambda^{(p)} = \max_{j \in I_p}(\boldsymbol{g}_j(\boldsymbol{\theta})^{(p)})^2$. For the Gaussian SG noise case we found however the following (safe) looser estimation of the maximum noise covariance to be more stable: $\lambda^{(p)} = \sum_{j \in I_p} b_j(\boldsymbol{\theta}) = \frac{||\boldsymbol{g}(\boldsymbol{\theta})^{(p)}||^2}{2}$. From a practical point of view, we found the following filtering procedure to be useful and robust:

$$\lambda^{(p)} \leftarrow \mu\lambda^{(p)} + (1-\mu)\frac{||\boldsymbol{g}^{(p)}(\boldsymbol{\theta})||^2}{2} \tag{7.11}$$

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 $\boldsymbol{B}(\boldsymbol{\theta})$. 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 $\sum_p \lambda^{(p)}$.

7.4.1.2 The case of Heavy Tailed Noise.

As anticipated, we challenge the assumption that SG noise is Gaussianly distributed ([129, 130]) and consider instead heavy tailed distributions. In particular, the hypothesis is that the noise follows and α-*stable* distribution, i.e.

$$w \sim p(w) = \mathcal{F}^{-1}\left(\exp(-|ct|^{\alpha})\right) \tag{7.12}$$

where $\alpha \in [0, 2]$. Notice that except for particular cases, $p(w)$ 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 [142], 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-$\boldsymbol{\alpha}$**, 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 minimize the l_2-distance between $p(x)$ and $q(x)$, where $p(x) = \sqrt{2\pi\sigma^2}\exp\left(-\frac{x^2}{2\sigma^2}\right)$ and $q(x) = \mathcal{F}^{-1}\left(\exp(-|ct|^\alpha)\right)$. As stated above, in general no closed form exists for $q(x)$. Thanks to Parseval's equality, however, we can compute the distance in the frequency domain between the two distributions, i.e.

$$C = \int_{-\infty}^{+\infty} |p(x) - q(x)|^2 dx = \int_{-\infty}^{+\infty} |p(t) - q(t)|^2 dt \tag{7.13}$$

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

$$C_{eq} = \int_{-\infty}^{+\infty} |p(t)|^2 dt - 2\int_{-\infty}^{+\infty} p(t)q(t)dt = \int_{-\infty}^{+\infty} \exp(-\sigma^2 t^2)dt - 2\int_{-\infty}^{+\infty} \exp(-\frac{\sigma^2 t^2}{2})\exp(-|ct|^\alpha)dt$$

$$\frac{\sqrt{\pi}}{\sigma} - \frac{2}{\sigma}\int_{-\infty}^{+\infty} \exp(-\frac{\tau^2}{2})\exp(-|\frac{c}{\sigma}\tau|^\alpha)d\tau = \frac{\sqrt{\pi}}{\sigma} - \frac{2\sqrt{2\pi}}{\sigma}\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}}\exp(-\frac{\tau^2}{2})\exp(-|\frac{c}{\sigma}\tau|^\alpha)d\tau =$$

$$\frac{1}{\sigma}\left(\sqrt{\pi} - \sqrt{2\pi}E_{T\sim N(0,1)}[\exp\left(-|\frac{c}{\sigma}T|^\alpha\right)]\right). \tag{7.14}$$

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

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

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: $\tilde{\sigma}^2 = 2c^2$. The parameters α, c are estimated (extending the results of [129, 143] to space varying settings) as described below. Given a

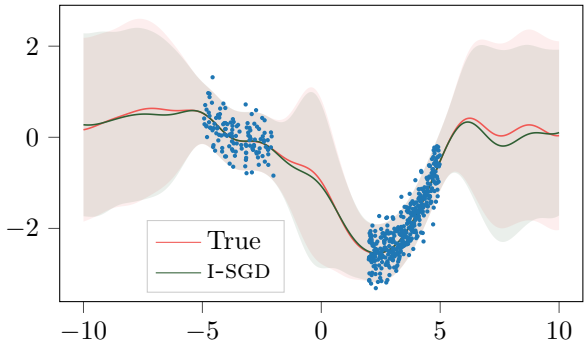


Figure 7.1 – True and I-SGD predictive posterior distributions on a simple example.

sequence of $N = N_1 \times N_2$ samples $w[n]$ from an alpha-stable distribution, it is possible to estimate α, c using

$$\frac{1}{\hat{\alpha}} = \frac{1}{\log(N_1)} \left(\frac{1}{N_2} \sum_{i=0}^{N_2-1} \log \left| \sum_{j=0}^{N_1-1} w[iN_1 + j] \right| - \frac{1}{N} \sum_{i=0}^{N-1} \log |w[i]| \right) \tag{7.15}$$

$$\hat{c} = \exp\left(\frac{1}{N} \sum_{i=0}^{N-1} \log |w[i]| - \left(\frac{1}{\hat{\alpha}} - 1\right) \gamma\right) \tag{7.16}$$

where $\gamma = 0.5772156649015329\ldots$ 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.

7.4.2 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 $p(\mathbf{w}) = \mathcal{N}(0, I_D)$; 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 Figure 7.1 we show the predictive posterior distribution (estimated using Eq. (7.2)) 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.

7.5 Assumptions and convergence to the true posterior

Our theory shows that the ideal version of I-SGD (Corollary 7.2.1 holds, and $\boldsymbol{B}(\boldsymbol{\theta})$ is known) converges to the true posterior with a constant learning rate. This is not the case for existing work. Even when $\boldsymbol{B}(\boldsymbol{\theta})$ is assumed to be known, SGFS requires the correction term $\nabla^{\top}\boldsymbol{B}(\boldsymbol{\theta})^{-1} = 0$. Also, both SGRLD and SGRHMC require computing $\nabla^{\top}\boldsymbol{B}(\boldsymbol{\theta})^{-1}$, for which an estimation procedure is elusive. The method in [139] needs a *constant*, diagonal $\boldsymbol{B}(\boldsymbol{\theta})$, 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 $\boldsymbol{B}(\boldsymbol{\theta})$ 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 require 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 trade-offs 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 7.6.1, 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.

7.6 Experimental Results

We first study I-SGD using standard UCI data-sets [144] and a fully connected Multi Layer Perceptron. Then, we focus on classification and use a CNN (Convolutional Neural Network) on MNIST [145]. We compare I-SGD (with the Gaussian approximation to the estimated α-stable distribution) to SGHMC [17], SGLD [16] and to alternative approaches to approximate Bayesian inference, including MCD [74], and SWAG [126], the variational SGD approach (V-SGD [127]). The code has been written in PYTHON.

7.6.1 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 it 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 contain unknown constants that are either impossible of 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 well 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 another one in terms of test metrics, while being poorer in terms of goodness of the posterior representation.

The two problems should be orthogonal and independent of each 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.

7.6.2 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. (7.2). 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-$\boldsymbol{\alpha}$**. 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-$\boldsymbol{\alpha}$ sum**. This variant is again equal to the **I-SGD-$\boldsymbol{\alpha}$** 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 [77] and learning rate 0.01.
- **I-SGD-$\boldsymbol{\alpha}$ Ad**. As **I-SGD-$\boldsymbol{\alpha}$** but again using as staring 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 [139] 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 [17].
- **SGLD**. We use the annealing procedure described in the original paper [16], 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 [16] and [100].
- **V-SGD C**. It is the implementation of the variational scheme described in [127]. 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

Table 7.3 – 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*

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.

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. 7.3,7.4 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 7.2). For this first set of experiments it seems that the presence of momentum helps SGHMC converging faster.

7.6.3 Classification tasks, with deeper models.

In this set of experiments we use a LeNet-5 (Convolutional Neural Network) on the Mnist dataset [145] to perform classification.
We compare different methods in terms of

Table 7.4 – 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

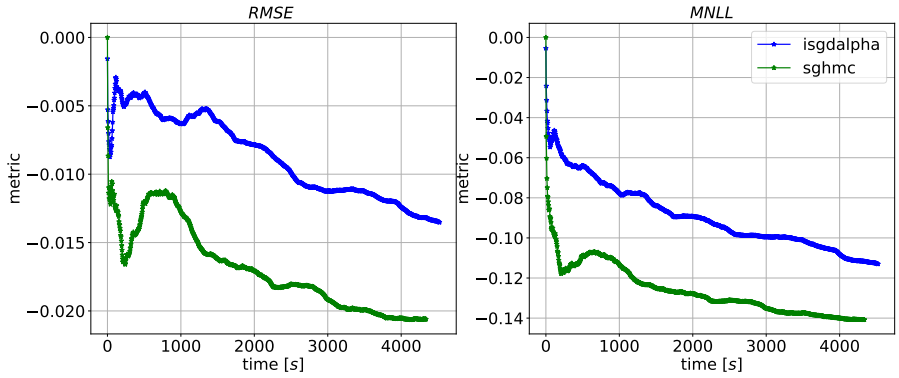


Figure 7.2 – Convergence speed of SGHMC and I-SGD for the WINE dataset. To improve readability the metrics are shifted vertically with respect to the value of the SGHMC method at time instant 0.

- 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. The entropy of the output of an N_{cl} classes classifier, represented by the vector $\mathbf{p}$, is defined as $-\sum_{i=1}^{N_{cl}} p_i \log p_i$ (so the lower, the better).
- 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. 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.

Table 7.5 – Results for classification on MNIST data-set.

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

- **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 7.5. 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 spectacularly. The I-SGD variants perform similarly to SGLD and in general they outperform SGHMC. An honest comment is that 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 performance 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.

7.7 Conclusion

SG methods allowed Bayesian posterior sampling algorithms, such as MCMC, to regain relevance in an age when data-sets have reached extremely large sizes. However, despite mathematical elegance and promising results, standard approaches from the literature are restricted to simple models. The sampling properties of these algorithms are determined by simplifying assumptions on the loss landscape, which do not hold for deep networks. SG-MCMC algorithms require vanishing learning rates, which force practitioners to develop creative annealing schedules that are often model specific and difficult to justify.

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 behavior of SG-MCMC algorithms. We presented a practical variant of the SGD algorithm, which uses a constant learning rate, and an additional noise to perform Bayesian posterior sampling. Our proposal is derived from an ideal method, which guarantees that samples are generated from the true posterior. When the noise terms are empirically estimated, our method automatically determines the learning rate, and it offers a very good approximation to the posterior, as demonstrated by our experimental campaign.

8
Conclusion and Perspectives

The work presented in this thesis is meant to improve robustness and scalability of machine learning models, with applications to the airline industry. We conclude this thesis by summarising the principal themes and contributions presented in the preceding chapters, with particular emphasis on their significance in the context of complementary work in this direction of research. This is followed by a brief outlook on possible avenues for future work where we indicate how one might go about achieving these objectives.

8.1 Themes and Contributions

In this thesis, we primarily investigated the following themes related to machine learning and airline industry:

- **Uncertainty in ticket price evolution [Chapter 3]**

  Airline companies use convoluted dynamic pricing strategies to maximize their revenues. This strong price instability causes concerns among travellers, which do not know how to face the uncertainty arising therefrom. In Chapter 3 we proposed three practical approaches to help travellers in dealing with uncertainty in ticket price evolution. Using the Value at Risk concept, we provide information on the smallest amount of money the user would pay in addition, with a certain probability, if he waits α days to buy the ticket. The evaluation of the quantile regression models shows generally calibrated predictions and good conditional density estimations. The regret approach, instead, provides directly a suggestion about whether to buy or wait. We evaluate it computing the amount of money that is lost, in relative terms, when relying on the output provided by the model instead of the hindsight decision, which shows that the proposed approach makes the user save money. The last approach estimates the expected profit of the option of buying the ticket in α days with today's price. Compared to the actual profit, we show that the proposed pricing model is fair. While the first and third approaches provide useful money information, the second one has the advantages of lending a practical suggestion about whether to buy the ticket or wait.

- **Model performance monitoring [Chapter 4]**

  Accurate travel products price forecasting is a highly desired feature that allows customers to take informed decisions about purchases, and companies to build and offer attractive tour packages. Thanks to machine learning, it is now relatively cheap to develop highly accurate statistical models for price time-series forecasting. However, once models are deployed in production, it is their monitoring, maintenance and improvement which carry most of the costs and difficulties over time. In Chapter 4 we introduced a data-driven framework to continuously monitor and maintain deployed time-series forecasting models' performance, to guarantee stable performance of travel products price forecasting models. Under a supervised learning approach, we predict the errors of time-series forecasting models over time, and use this predicted performance measure to achieve both model monitoring and maintenance. The experimental evaluation shows accurate forecasting error predictions, which proves the framework's reliability in carrying out model monitoring and allows for dynamic model selection.

- **Sparsified asynchronous SGD [Chapter 6]**

  The latest advances in machine learning, together with the increasing availability of massive datasets, have given rise to the research field of large-scale machine learning, where deep learning models with billions of parameters are trained on very big datasets. Large-scale machine learning relies on distributed optimization, whereby several machines contribute to the training process of a statistical model, using Stochastic Gradient Descent as optimization method. In Chapter 6 we studied the performance of asynchronous, distributed settings, when applying sparsification, a technique used to reduce communication overheads. In particular, for the first time in an asynchronous, non-convex setting, we theoretically prove that, in presence of staleness, sparsification does not harm SGD performance: the ergodic convergence rate matches the known result of standard SGD, that is $\mathcal{O}(1/\sqrt{T})$. The empirical study that we carried out to complement our theory confirms that the effects of sparsification on the convergence rate are negligible, when compared to "vanilla" SGD, even in the challenging scenario of an asynchronous, distributed system. These results carry over when the system scales out, which is a truly desirable property.

- **Bayesian posterior sampling [Chapter 7]**

  Latest advances in machine learning research have led to the use of such models in an ever increasing number of applications, including ones previously requiring human expertise. The adoption of machine learning models to sensitive applications has hence sparked a resurgence of interest in probabilistic modelling, which is inherently intended to capture the uncertainty or lack of knowledge a model might have about a learned task, through the use of Bayesian inference. Although conceptually simple, applying Bayesian inference poses a number of computational challenges, due to high-dimension operations and non-analytical equations. Several stochastic gradient MCMC algorithms have been proposed to carry out approximate Bayesian inference.

Despite mathematical elegance and some promising results restricted to simple models, however, most of these works fall short in easily dealing with the complexity of the loss landscape of deep models, for which stochastic optimization poses serious challenges. In Chapter 7 we introduced a novel, practical approach to posterior sampling, which makes the SG noise isotropic using a fixed learning rate that we determine analytically, and that requires weaker assumptions than existing algorithms. Our extensive experimental validations indicate that our proposal is competitive with the state-of-the-art on SG-MCMC, while being much more practical to use.

8.2 Future work

Beyond the extensive discussion featured in this thesis, the themes explored in this body of work not only motivate immediate extensions for improvement, but also set the foundations for broader long-term objectives. In this section, we expand upon the directions for future work which we believe to be particularly pertinent to ongoing developments in both the practical and theoretical aspects of machine learning discussed in this thesis.

8.2.1 Uncertainty in ticket price evolution

Finding the best moment to buy. In Section 3.2 we proposed three approaches to deal with uncertainty in ticket price evolution. All of them consider two options: buying the ticket at time t or at time $t + \alpha$. However, this kind of application turns out to be limited. If we consider a normal user which searches for a flight ticket, usually he is interested in knowing which is the best moment to buy the ticket, considering the whole period before the departure. Thus, the use-cases we discussed in this work represent just particular cases of the more general application, which acts as a decision-making tool and suggests the best moment to buy a ticket.
This setting is similar to the well-known secretary problem [146] and it can be solved using the optimal stopping theory [147], which is concerned with the problem of choosing a time to take a particular action, in order to maximise an expected reward or minimise an expected cost. The approach we described in Section 3.2.5 can be therefore improved by applying such theory.
Quantile regression approach. In Section 3.2.5 we introduced a strategy to help travellers in dealing with uncertainty in ticket price evolution which provides a suggestion about whether to buy at time t or wait α days more, using the quantile function of $\Delta = p_{t+\alpha} - p_t$ (Table 3.3). However, we saw that this strategy has some limitations. Firstly, it significantly depends on the quantile τ, a parameter that represents the user's risk appetite, thus its generalization is reduced. Secondly, even fixing τ, an important role is played by the distribution of Δ and we have seen an example where by following the model's suggestion, we end up taking the risk of loosing a large amount of money, to save a small amount.
A solution to this problem is to introduce some thresholds on the Δ values. In this way we could define what are the significant gains and losses, which are crucial in the decision

about whether to buy or wait. This problem is related to prospect theory [148], which assumes that losses and gains are valued differently and thus individuals make decisions based on perceived gains instead of perceived losses.

8.2.2 Stochastic optimization

Sparsified asynchronous SGD. In Section 6.4 we provided the convergence analysis of SGD when the joint effects of sparsification and asynchrony are taken into account and we limited the study to the memory-less variant of SGD. However, in our empirical study we compared also the memory variant and we verified that this considerably benefits from the error correction technique. Thus, the first extension of this work is to study the convergence of sparsified SGD with memory, in asynchronous settings.
Beside this, it would be interesting to investigate the connection between gradient sparsification and recent studies showing that the landscape of the loss surface of deep models is typically sparse [121, 102, 122]. In light of such works, [123] suggests that sparsification can be directly applied to model parameters, albeit training requires multiple stages. Gradient sparsification could then be studied as a mechanism to favor model compression at training time.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[2] C. M. Bishop, *Pattern recognition and machine learning*, 1st ed. Springer, 2006.

[3] O. Etzioni, R. Tuchinda, C. A. Knoblock, and A. Yates, "To buy or not to buy: Mining airfare data to minimize ticket purchase price," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 119–128.

[4] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in neural information processing systems*, 2015, pp. 2503–2511.

[5] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," *IEEE transactions on cybernetics*, 2019.

[6] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[7] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[8] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting Distributed Synchronous SGD," in *International Conference on Learning Representations Workshop Track*, 2016.

[9] J. Zhang, C. D. Sa, I. Mitliagkas, and C. Re, "Parallel SGD: When does averaging help?" *arXiv*, vol. abs/1606.07365, 2016.

[10] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 693–701.

[11] C. M. De Sa, C. Zhang, K. Olukotun, and C. Re, "Taming the wild: A unified analysis of hogwild-style algorithms," in *Advances in neural information processing systems*, 2015, pp. 2674–2682.

[12] S. U. Stich, "Local SGD converges fast and communicates little," in *Proceedings of the 7th International Conference on Learning Representations*, ser. ICLR'19, 2019.

[13] H. Yu, S. Yang, and S. Zhu, "Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *AAAI 2019*, 2019.

[14] E. Awad, S. Levine, M. Kleiman-Weiner, S. Dsouza, J. B. Tenenbaum, A. Shariff, J.-F. Bonnefon, and I. Rahwan, "Blaming humans in autonomous vehicle accidents: Shared responsibility across levels of automation," *arXiv preprint arXiv:1803.07170*, 2018.

[15] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, pp. 1573–1576, 2016.

[16] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th international conference on machine learning, ICML*, 2011, pp. 681–688.

[17] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient hamiltonian monte carlo," in *International conference on machine learning*, 2014, pp. 1683–1691.

[18] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven, "Bayesian sampling using stochastic gradient thermostats," in *Advances in neural information processing systems*, 2014, pp. 3203–3211.

[19] E. Thomas, A. G. Ferrer, B. Lardeux, M. Boudia, C. Haas-Frangii, and R. A. Agost, "Cascaded machine learning model for efficient hotel recommendations from air travel bookings," in *Proceedings of Proceedings of the 12th ACM Conference on Recommender Systems, ACM RecSys Workshop on Recommenders in Tourism (RecTour 2019 vol 2435)*. CEUR Workshop Proceedings Copenhagen, 2019, pp. 9–16.

[20] N. Bondoux, A. Q. Nguyen, T. Fiig, and R. Acuna-Agost, "Reinforcement learning applied to airline revenue management," *Journal of Revenue and Pricing Management*, pp. 1–17, 2020.

[21] Y. Chen, J. Cao, S. Feng, and Y. Tan, "An ensemble learning based approach for building airfare forecast service," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 964–969.

[22] T. Wohlfarth, S. Clemencon, F. Roueff, and X. Casellato, "A data-mining approach to travel price forecasting," in *2011 10th International Conference on Machine Learning and Applications and Workshops*, vol. 1, 2011, pp. 84–89.

[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[24] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[25] A. Shafaei, M. Schmidt, and J. Little, "Does your model know the digit 6 is not a cat? a less biased evaluation of "outlier" detectors," *ArXiv*, vol. abs/1809.04729, 2018.

[26] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[27] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," *arXiv preprint arXiv:1703.02910*, 2017.

[28] T. Chen, J. Navrátil, V. Iyengar, and K. Shanmugam, "Confidence scoring using whitebox meta-models with linear classifier probes," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1467–1475.

[29] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[30] D. Lopez-Paz, "From dependence to causation," *arXiv preprint arXiv:1607.03300*, 2016.

[31] N. Tagasovska and D. Lopez-Paz, "Single-model uncertainties for deep learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 6417–6428.

[32] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[33] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, 2015.

[34] T. Gneiting, F. Balabdaoui, and A. E. Raftery, "Probabilistic forecasts, calibration and sharpness," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 69, no. 2, pp. 243–268, 2007.

[35] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[36] C. M. Bishop, "Mixture density networks," 1994.

[37] S. Choi, K. Lee, S. Lim, and S. Oh, "Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6915–6922.

[38] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in neural information processing systems*, 2015, pp. 3483–3491.

[39] M. P. Holmes, A. G. Gray, and C. L. Isbell, "Fast nonparametric conditional density estimation," in *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, ser. UAI'07. Arlington, Virginia, USA: AUAI Press, 2007, p. 175–182.

[40] B. L. Trippe and R. E. Turner, "Conditional density estimation with bayesian normalising flows," *arXiv preprint arXiv:1802.04908*, 2018.

[41] R. Koenker and G. Bassett Jr, "Regression quantiles," *Econometrica: journal of the Econometric Society*, pp. 33–50, 1978.

[42] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola, "Nonparametric quantile estimation," *Journal of machine learning research*, vol. 7, no. Jul, pp. 1231–1264, 2006.

[43] Y. Liu and Y. Wu, "Stepwise multiple quantile regression estimation using non-crossing constraints," *Statistics and its Interface*, vol. 2, no. 3, pp. 299–310, 2009.

[44] Y. Liu and Y. Wu, "Simultaneous multiple non-crossing quantile regression estimation using kernel constraints," *Journal of nonparametric statistics*, vol. 23, no. 2, pp. 415–437, 2011.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] R. Koenker, S. Portnoy, P. T. Ng, A. Zeileis, P. Grosjean, and B. D. Ripley, "Package 'quantreg'," 2012.

[47] J. C. Hull and A. D. White, "Value at risk when daily changes in market variables are not normally distributed," *The Journal of Derivatives*, vol. 5, no. 3, pp. 9–19, 1998.

[48] R. F. Engle and S. Manganelli, "Caviar," *Journal of Business & Economic Statistics*, vol. 22, no. 4, pp. 367–381, 2004.

[49] J. W. Taylor, "A quantile regression approach to estimating the distribution of multiperiod returns," *The Journal of Derivatives*, vol. 7, no. 1, pp. 64–78, 1999.

[50] P. J. Schoemaker, *Experiments on decisions under risk: The expected utility hypothesis*. Springer Science & Business Media, 2013.

[51] W. Feller, "An introduction to probability theory and its applications," *1957*.

[52] A. O'Sullivan and S. M. Sheffrin, *Economics : Principles in Action*. Pearson Prentice Hall, 2003.

[53] C. Ré, F. Niu, P. Gudipati, and C. Srisuwananukorn, "Overton: A data system for monitoring and improving machine-learned products," *arXiv preprint arXiv:1909.05372*, 2019.

[54] International Air Transport Association, "World air transport statistics," *World air transport statistics 2019*, 2019.

[55] M. Aiolfi and A. Timmermann, "Persistence in forecasting performance and conditional combination strategies," *Journal of Econometrics*, vol. 135, no. 1-2, pp. 31–53, 2006.

[56] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[57] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1387–1395.

[58] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 793–804.

[59] J. A. Ferreira, R. H. Loschi, and M. A. Costa, "Detecting changes in time series: A product partition model with across-cluster correlation," *Signal Process.*, vol. 96, pp. 212–227, 2014.

[60] S. Liu, M. Yamada, N. Collier, and M. Sugiyama, "Change-point detection in time-series data by relative density-ratio estimation," *Neural Networks*, vol. 43, pp. 72–83, 2013.

[61] A. Saadallah, F. Priebe, and K. Morik, "A drift-based dynamic ensemble members selection using clustering for time series forecasting," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2019, pp. 678–694.

[62] A. Saadallah, L. Moreira-Matias, R. Sousa, J. Khiari, E. Jenelius, and J. Gama, "Bright—drift-aware demand predictions for taxi networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 2, pp. 234–245, 2020.

[63] E.-J. Wagenmakers, P. Grünwald, and M. Steyvers, "Accumulative prediction error and the selection of time series models," *Journal of Mathematical Psychology*, vol. 50, no. 2, pp. 149 – 166, 2006, special Issue on Model Selection: Theoretical Developments and Applications.

[64] A. R. Ali, B. Gabrys, and M. Budka, "Cross-domain meta-learning for time-series forecasting," *Procedia Computer Science*, vol. 126, pp. 9–18, 2018.

[65] T. S. Talagala, R. J. Hyndman, G. Athanasopoulos *et al.*, "Meta-learning how to forecast time series," *Monash Econometrics and Business Statistics Working Papers 6/18, Monash University, Department of Econometrics and Business Statistics*, 2018.

[66] V. Cerqueira, L. Torgo, F. Pinto, and C. Soares, "Arbitrated ensemble for time series forecasting," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2017, pp. 478–494.

[67] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[68] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: Results, findings, conclusion and way forward," *International Journal of Forecasting*, vol. 34, no. 4, pp. 802 – 808, 2018.

[69] Z. Chen and Y. Yang, "Assessing forecast accuracy measures," *Preprint Series*, vol. 2010, pp. 2004–10, 2004.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[72] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[73] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[74] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[75] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[76] M. Titsias, "Variational learning of inducing variables in sparse gaussian processes," in *Artificial Intelligence and Statistics*, 2009, pp. 567–574.

[77] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[78] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[79] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *Journal of the American statistical association*, vol. 106, no. 496, pp. 1513–1527, 2011.

[80] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1223–1231.

[81] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, 2014, pp. 583–598.

[82] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, 2014, pp. 571–582.

[83] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNN s," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[84] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, " QSGD : Communication-efficient SGD via gradient quantization and encoding," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 1709–1720.

[85] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1509–1519.

[86] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv*, vol. abs/1704.05021, 2017.

[87] N. Dryden , T. Moon , S. A. Jacobs , and B. V. Essen, "Communication quantization for data-parallel training of deep neural networks," in *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, Nov 2016, pp. 1–8.

[88] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[89] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 4447–4458.

[90] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 5973–5983.

[91] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *International Conference on Learning Representations*, 2018.

[92] P. Goyal, P. Dollar, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training i mage net in 1 hour," *arXiv*, vol. abs/1706.02677, 2017.

[93] Y. You, I. Gitman, and B. Ginsburg, "Scaling sgd batch size to 32k for i mage n et training," *arXiv, abs/1708.03888*, vol. 6, 2017.

[94] M. Krzywinski and N. Altman, "Importance of being uncertain," 2013.

[95] B. de Finetti, *Theory of Probability*, ser. First of two volumes translated from *Teoria Delle probabilità*, published 1970. The second volume appeared under the same title in 1975. Wiley, New York, 1974.

[96] Z. Ghahramani, "Bayesian non-parametrics and the probabilistic approach to modelling," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1984, p. 20110553, 2013.

[97] D. J. MacKay, "Bayesian interpolation," *Neural computation*, vol. 4, no. 3, pp. 415–447, 1992.

[98] C. E. Rasmussen and Z. Ghahramani, "Occam's razor," in *Advances in neural information processing systems*, 2001, pp. 294–300.

[99] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine learning*, vol. 37, no. 2, pp. 183–233, 1999.

[100] S. Ahn, A. Korattikara, and M. Welling, "Bayesian posterior sampling via stochastic gradient fisher scoring," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, pp. 1771–1778.

[101] Y.-A. Ma, T. Chen, and E. Fox, "A complete recipe for stochastic gradient mcmc," in *Advances in Neural Information Processing Systems*, 2015, pp. 2917–2925.

[102] F. Draxler, K. Veschgini, M. Salmhofer, and F. A. Hamprecht, "Essentially no barriers in neural network energy landscape," *arXiv preprint arXiv:1803.00885*, 2018.

[103] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns," in *Advances in Neural Information Processing Systems*, 2018, pp. 8789–8798.

[104] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[105] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 1223–1231.

[106] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: ACM, 2017, pp. 463–478.

[107] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2737–2745.

[108] J. Liu and S. J. Wright, "Asynchronous stochastic coordinate descent: Parallelism and convergence properties," *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 351–376, 2015.

[109] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, "Asynchronous byzantine machine learning (the case of sgd)," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholm, Sweden: PMLR, 10–15 Jul 2018, pp. 1145–1154.

[110] T. Lin, S. U. Stich, and M. Jaggi, "Don't use large mini-batches, use local SGD ," *arXiv, abs/1808.07217*, 2018.

[111] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD ," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, A. Storkey and F. Perez-Cruz, Eds. PMLR, 2018, pp. 803–812.

[112] W. Dai, Y. Zhou, N. Dong, H. Zhang, and E. P. Xing, "Toward understanding the impact of staleness in distributed machine learning," in *Proceedings of the 7th International Conference on Learning Representations*, ser. ICLR'19, 2019.

[113] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 1299–1309.

[114] E. Moulines and F. R. Bach, "Non-asymptotic analysis of stochastic approximation algorithms for machine learning," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 451–459.

[115] I. Mitliagkas, C. Zhang, S. Hadjis, and C. Re, "Asynchrony begets momentum, with an application to deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 997–1004.

[116] S. U. Stich and S. P. Karimireddy, "The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication," 2019.

[117] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "Compression by the signs: distributed learning is a two-way street," in *ICLR*, 2018.

[118] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," in *Proceedings of Machine Learning Research*, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 560–569.

[119] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[120] A. Shanbhag, H. Pirk, and S. Madden, "Efficient top-k query processing on massively parallel hardware," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1557–1570.

[121] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou, "Empirical analysis of the hessian of over-parametrized neural networks," *arXiv preprint arXiv:1706.04454*, 2017.

[122] R. Karakida, S. Akaho, and S.-i. Amari, "Universal statistics of fisher information in deep neural networks: mean field approach," *arXiv preprint arXiv:1806.01316*, 2018.

[123] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks." in *ICLR*, 2019.

[124] S. Patterson and Y. W. Teh, "Stochastic gradient riemannian langevin dynamics on the probability simplex," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3102–3110.

[125] P. Chaudhari and S. 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.

[126] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson, "A simple baseline for bayesian uncertainty in deep learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 132–13 143.

[127] S. Mandt, M. D. Hoffman, and D. M. Blei, "Stochastic gradient descent as approximate bayesian inference," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4873–4907, 2017.

[128] G. Franzese, "Contributions to Efficient Machine Learning," Ph.D. dissertation, Politecnico di Torino, 2021.

[129] U. Şimşekli, M. Gürbüzbalaban, T. H. Nguyen, G. Richard, and L. Sagun, "On the heavy-tailed theory of stochastic gradient descent for deep neural networks," *arXiv preprint arXiv:1912.00018*, 2019.

[130] U. Şimşekli, L. Sagun, and M. 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.

[131] C. Chen, D. Carlson, Z. Gan, C. Li, and L. Carin, "Bridging the gap between stochastic gradient mcmc and stochastic optimization," in *Artificial Intelligence and Statistics*, 2016, pp. 1051–1060.

[132] C. W. Gardiner, *Handbook of stochastic methods for physics, chemistry and the natural sciences*, 3rd ed., ser. Springer Series in Synergetics. Springer-Verlag, 2004, vol. 13.

[133] H. Kushner and G. G. Yin, *Stochastic approximation and recursive algorithms and applications*. Springer Science & Business Media, 2003, vol. 35.

[134] L. Ljung, G. Pflug, and H. Walk, *Stochastic Approximation and Optimization of Random Systems*. CHE: Birkhauser Verlag, 1992.

[135] C. Li, C. Chen, D. Carlson, and L. Carin, "Preconditioned stochastic gradient langevin dynamics for deep neural networks," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[136] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, p. 2, 2011.

[137] M. Girolami and B. Calderhead, "Riemann manifold langevin and hamiltonian monte carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 2, pp. 123–214, 2011.

[138] R. Zhang, C. Li, J. Zhang, C. Chen, and A. G. Wilson, "Cyclical stochastic gradient mcmc for bayesian deep learning," in *International Conference on Learning Representations, ICLR*, 2020.

[139] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter, "Bayesian optimization with robust bayesian neural networks," in *Advances in neural information processing systems*, 2016, pp. 4134–4142.

[140] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox, "On the information bottleneck theory of deep learning," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, no. 12, p. 124020, 2019.

[141] Z. Zhu, J. Wu, B. Yu, L. Wu, and J. Ma, "The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects," in *International Conference on Machine Learning*, 2019, pp. 7654–7663.

[142] U. ŞimŠekli, "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.

[143] J. Levy Vehel, A. Philippe, and C. Robet, "Explicit and combined estimators for stable distributions parameters," 11 2018.

[144] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[145] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[146] J. Neil Bearden, "A new secretary problem with rank-based selection and cardinal payoffs," *Journal of Mathematical Psychology*, vol. 50, no. 1, pp. 58 – 59, 2006.

[147] B. Brown, "Great expectations: The theory of optimal stopping," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 4, pp. 610–610, 1972.

[148] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," in *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, 2013, pp. 99–127.

Appendix A
Additional regret results

In Section 3.2.5 we introduced two approaches to help travellers in dealing with uncertainty in ticket price evolution, both providing a suggestion about whether to buy at time t or wait α days more. We put in this section the regret results obtained using the strategy in Table 3.3, using SQR as quantile regression model. To simplify the procedure, we select the quantile τ that gives the minimum regret. Results show that the two machine learning models have similar performance, with RF providing slightly smaller expected regret.

	$\alpha = 1$	$\alpha = 7$	$\alpha = 30$
RF	**0.0087**	**0.0076**	**0.0301**
SQR	0.0088	0.0094	0.0325
Now	0.0088	0.0133	0.0447
Later	0.0177	0.0118	0.0341
Random	0.0132	0.0125	0.0395

Table A.1 – Expected regret results for different values of lag α, time-based split.

	$\alpha = 1$	$\alpha = 7$	$\alpha = 30$
RF	**0.0092**	**0.0094**	**0.0185**
SQR	0.0111	0.0135	0.0190
Now	0.0111	0.0152	0.0437
Later	0.0205	0.0153	0.0361
Random	0.0158	0.0152	0.0398

Table A.2 – Expected regret results for different values of lag α, random split.

A.0.1 Optimal τ

Given that in the results shown above we used the best τ, it is worth to understand how this optimal τ varies with respect to α and DTD. In Figure A.1 we see that this has high values when dealing with $\alpha = 1$, while in the two remaining cases the behavior is different. For $\alpha = 7$ we can use very low quantiles, in case of large DTDs, and medium quantiles for the rest. The trend is similar with $\alpha = 30$, except that the quantiles are much larger when DTD is small. This tells us that when the decision about whether to buy or wait relates 1 or 30 days (up to 6 months before the departure) of lag, the verdict helps risk-adverse people, corresponding to bigger τ. Instead, when the problem is about whether to buy now or wait one week, the model is more suited for risk-tolerant people. These results are in line with what we have seen in Figure 3.11, in that 30 days

is a lag where prices vary a lot and make risk-adverse people to naively adopt the Now strategy, because of the future uncertainty. For seven days, instead, the model addresses people who are ready to wait one more week, but that doing this end up paying more. In this case, results show that using SQR it is possible to save money.

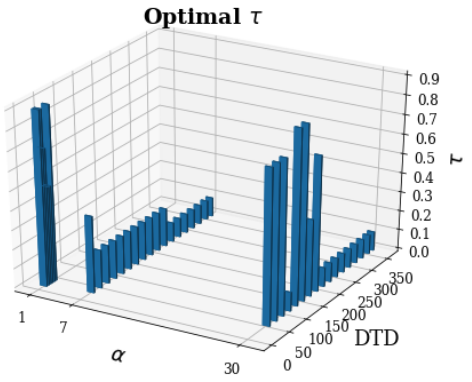


Figure A.1 – Optimal τ as function of α and DTD

Appendix B
Isotropic SGD: Proof of theorems

B.0.1 Proof of Theorem 1

The stationary distribution of the above SDE, $\rho(\boldsymbol{\theta}) \propto \exp(-\phi(\boldsymbol{\theta}))$, satisfies the following FPE:

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

that we rewrite as

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

The above equation is verified with $\nabla f(\boldsymbol{\theta}) = \nabla\phi(\boldsymbol{\theta})$ if

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

that proves Theorem 1.

B.0.2 Proof of Theorem 2

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

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

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

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

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

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

If $\nabla^\top \boldsymbol{P}(\boldsymbol{\theta}) = \boldsymbol{0}$, in fact:

$$\mathrm{Tr}\left(\nabla\left(\nabla^\top(\phi(\boldsymbol{z}))\rho(\boldsymbol{z})\begin{bmatrix}\boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{P}(\boldsymbol{\theta}) & \boldsymbol{0}\end{bmatrix}\right)\right) = \nabla^\top\left(\begin{bmatrix}\boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{P}(\boldsymbol{\theta}) & \boldsymbol{0}\end{bmatrix}\nabla(\phi(\boldsymbol{z}))\rho(\boldsymbol{z})\right) =$$
$$\nabla^\top\left(\begin{bmatrix}\boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{P}(\boldsymbol{\theta}) & \boldsymbol{0}\end{bmatrix}\right)\nabla(\phi(\boldsymbol{z}))\rho(\boldsymbol{z}) + \mathrm{Tr}\left(\begin{bmatrix}\boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{P}(\boldsymbol{\theta}) & \boldsymbol{0}\end{bmatrix}\nabla\left(\nabla^\top(\phi(\boldsymbol{z}))\rho(\boldsymbol{z})\right)\right) = 0,$$

since $\nabla^\top\begin{bmatrix}\boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{P}(\boldsymbol{\theta}) & \boldsymbol{0}\end{bmatrix} = \boldsymbol{0}$ and the second term is zero due to the fact that $\begin{bmatrix}\boldsymbol{0} & -\boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{P}(\boldsymbol{\theta}) & \boldsymbol{0}\end{bmatrix}$ is anti-symmetric while $\nabla\left(\nabla^\top(\phi(\boldsymbol{z}))\rho(\boldsymbol{z})\right)$ is symmetric.
Thus we can rewrite:

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

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