UNIVERSITÁ DEGLI STUDI DI MILANO-BICOCCA

Variational inference and semi-parametric methods for time-series probabilistic forecasting

by

Mattia Bolzoni

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

Dipartimento di Statistica e Metodi Quantitativi

March 2021

Declaration of Authorship

I, Mattia Bolzoni, declare that this thesis titled, 'Variational inference and semi-parametric methods for time-series probabilistic forecasting' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

"Better to be approximately right than exactly wrong."

Many people, approximately quoting Carveth Read

UNIVERSITÁ DEGLI STUDI DI MILANO-BICOCCA

Abstract

Dipartimento di Statistica e Metodi Quantitativi

Doctor of Philosophy

by Mattia Bolzoni

Probabilistic forecasting is a common task. The usual approach assumes a fixed structure for the outcome distribution, often called model, that depends on unseen quantities called parameters, then uses data to infer a reasonable distribution over these latent values.

The inference step is not always straightforward, because selecting a single value can lead to poor performances and overfitting, while handling a proper distribution with MCMC can be challenging. Variational Inference (VI) is emerging as a viable approximated optimisation based alternative, that models the target posterior with instrumental variables called variational parameters. However, VI usually imposes a parametric structure on the proposed posterior.

The thesis's first contribution is Hierarchical Variational Inference (HVI) a methodology that uses Neural Networks to create semi-parametric posterior approximations with the same minimum requirements as Metropolis-Hastings or Hamiltonian MCMC.

The second contribution is a Python package to conduct VI on time-series models for mean-covariance estimate, using HVI and standard VI techniques combined with Neural Networks. Results on econometric and financial data show a consistent improvement using VI, compared to point estimate, obtaining lower variance forecasting.

Acknowledgements

I want to thank many people, but conciseness requires to shorten the list.

I thank Matteo Pelagatti, for his never invasive guide and advice.

I thank the PhD colleagues, invaluable friends that taught and helped me a lot, in alphabetical order: Riccardo Brignone, Andrea Cappozzo, Francesco Denti, Luca Gonzato, Nickos Petrakis.

I thank the people that bore my bad character during these years, friends, relatives and companions.

Contents

List of Figures

List of Tables

Abbreviations

VI	**V**ariational **I**nference
ELBO	**E**vidence **L**ower **BO**und
MC	**M**onte **C**arlo methods
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MAP	**M**aximum-**A**-**P**osteriori
DCC	**D**ynamical **C**onditional **C**orrelation
LW	**L**edoit-**W**olf covariance estimator
SGD	**S**tochastic-**G**radient **D**escent
NN	**N**eural **N**etworks
MLP	**M**ulti **L**inear **P**rojector
LSTM	**L**ong-**S**hort-**T**erm **M**emory
AE	**A**uto**E**ncoder
VAE	**V**ariational**A**uto**E**ncoder
EM	**E**xpectation **M**aximization

Introduction

One of the main tasks of statistics is to forecast unknown quantities given the available information. The standard approach is to assume a fixed probabilistic structure, called model or learner, that depends on unobserved variables, called parameters. After collecting data about the target phenomenon it is possible to infer a distribution for this latent entities, and then use it to forecast future values. Many methodologies within the statistical learning literature exploit such scheme.

How to choose the abstract structure and how to infer the unseen parameters depends on the problem at hand. Statistics literature usually calls these two steps modelling and inference, while machine learning literature refers to them as learner choice and training phase.

The dimension of the parameters space, or model dimension, is a crucial element to take into account. A vast number of parameters adds extra flexibility, allowing the model to represent real-world complexity. Unfortunately, this flexibility makes them able to learn also spurious correlations or random patterns. Not only, but proper inference could become computational intensive for high-dimensional models.

In particular, one of the oldest approach (and probably, still the more common) is to select the single value for the parameters that better explains the observed data, providing the best forecast on these examples. Unfortunately, this straightforward technique could lead to estimates that adapt well to the current observations but not to future cases. Bayesian inference is more reliable but can be challenging, because it not only requires to formalise priors knowledge about the problem on the form of a probability measure over the parameters but the inferred distribution, called posterior, is often intractable. In these cases, the traditional way to handle the posterior is Markov Chain Monte Carlo (MCMC), an asymptotically exact sampling technique.

Variational Inference (VI), an optimisation based approximation method, is emerging as a viable alternative. VI is usually way faster than MCMC but does not guarantee exactness. Not only, adapting it to the problem at hand could require a lot of effort

[1]. Many methodologies lie in the middle of such tradeoff between exactness and computational time. Anyway, the underlying principles are usually the same: maximise an empirical score or build a procedure that converges to a satisfying solution, or combine both.

This paper introduces a novel VI variant, called Hierarchical Variational Inference (HVI) that allows for flexible inference with minimum effort (equivalent to apply standard Metropolis-Hastings) using a specific class of Neural Networks (NN). NN are flexible semi-parametric functions. Using backpropagation is possible to modify their parameters, adapting the function to optimise a given functional score. Applying them to the VI score (called Evidence Lower Bound, or ELBO) allows conducting the inference. This is the first contribution of the paper.

The second contribution is a Python package that uses NN and VI for time-series probabilistic forecasting.

NN functions are commonly used as forecasters, mapping the input regressors to the outcome probability distribution, depending on a high number of parameters that can lead to overfitting. Randomised approaches with Bayesian interpretations (as dropout [2]), proper Bayesian estimation [3], or approximations with VI techniques [4] can increase their performances and reduce the tendency to overfit data. These distributional approaches to parameters value lead to state-of-the-art models in many contexts as Natural Language Processing or Image recognition [2].

Neural networks have been applied to time series for long and have shown different results. However, single-value forecast is the typical case. Rare examples of multivariate probabilistic forecasting for NN exists. Even advanced methodologies, as [5], usually focus on univariate point prediction. A reason is that multivariate time-series forecast requires to consider interactions between the single variables and the different time-indexes, leading naturally to conditional probability functions with many parameters. The high dimension of the problem, combined with the naturally high dimension of networks forecasters makes it challenging to conduct inference for NN, as will be shown empirically in chapter 4.

Econometric techniques usually tackle this dimension issue with parsimonious models and reliable inference procedures. Although less common, Bayesian estimate has shown able to improve performances on multivariate time series [6],[7]. Still, it requires intensive computational effort.

This paper shows that combining VI with a mean-covariance likelihood, NN can improve the forecasting power of such time-series standard estimators in terms of bias-variance

tradeoff. The above claim is tested on both financial data and macroeconomics series, leading to consistent results.

The thesis is organised as follows: 1 introduce the theoretical framework behind the paper; 2 define HVI and compares it with standard MCMC on artificial data; 3 presents the Python package for probabilistic forecast on multivariate time-series; 4 concludes, applying the implemented models on a forecasting experiment and discussing the results; the appendix A collects mathematical constructions, notations and definitions used in the previous chapters.

Chapter 1

Inference and training, models and learners

This chapter introduces the methodologies applied in the remaining chapters, using the notations described in the appendix A. This tour will include few models examples 1.3, inference schemes and numerical techniques 1.4. The chapter is not a comprehensive discussion of such topics but wants to be a concise presentation of the underlying principle, with literature reference.

1.1 Probabilistic forecasting

One of the most common problems in statistics and machine learning is the estimate of the probability of future outcome of a target variables y.

A reasonable approach is to assign a probability, conditionally on available information x, to every possible outcome $p(y|x)$. Whenever y is continuous $p(y|x)$ become a distribution density. Such probabilistic forecasting has been intensely studied in literature and applied in practice.

Comparing the quality of different forecasters is not always straightforward. Using a score function S is the most common choice. Score functions [8] assign a numeric value $S(p(y|x), y)$ to the forecast, based on the observed outcome y and the predicted probability (or density).

Consider, for example, a forecaster f that predicts the most probable outcome of y, called $\hat{y} = f(x)$, trying to get as close as possible. This kind of forecast is often called single-value forecast. A common error function in this context is mean-square-error

(for continuous y) or cross-entropy (for categorical y). Under this standard metrics, single-value forecast can be seen as a simplified version of probabilistic forecasting [9].

1.2 Statistical learning

How to assign $p(y|x)$?

A standard approach in statistical learning assumes a fixed form for the conditional probability, L, that depends on unobservable parameters $\theta \in \Omega \subset \mathbb{R}^d$. The forecast distribution is then $p(y|x) = L(y, \theta, x)$.

Statistical literature calls L likelihood, while machine learning literature calls it learner function or model.

After observing data, it is possible to infer a reasonable distribution for the parameters, $p(\theta)$, and then use this a-posteriori random variable to forecast new observations with

$$p(\cdot|x) = \int_{\Omega} L(\cdot, \theta, x) dp(\theta) \tag{1.1}$$

.

This general approach is a common factor for many techniques in the statistical learning field. It can be virtually separated in two phases: deciding L and inferring $p(\theta)$ after observing data (sometimes called inference or training).

The two steps affect one another. In practice, the existence of a reliable inference procedure restricts the choice of L.

Not only, probabilistic theory can suggest different $p(\theta)$ with useful properties, but such distributions are often impossible to handle directly and could require numerical approximation.

The following section 1.3 briefly introduce the models from literature that will be used in the remaining chapters; follows 1.4 that describe the mentioned inference techniques and numerical approximation methods.

1.3 Learners and models

The data structure affects the choice of L, that must adapt to the problem at hand.

Consider $y \in \mathbb{R}^d$.

It can be reasonable to assume that the different observations are independent, conditionally on the set of regressors x_i, x_j (if $y_i, y_j, i \neq j$), and that L depends only on global parameters θ: $p(y_i|x_i) = L(y_i, \theta, x_i)$.

In such case, it is possible to consider the model:

$$\begin{aligned} F_1 : x_i, \theta \to \mu \in \mathbb{R}^d \\ F_2 : x_i, \theta \to V \in \mathcal{M}_d^+ \\ L(y_i, \theta, x_i) = \mathcal{N}_{\mu_i, V_i}(y_i) \end{aligned} \tag{1.2}$$

for any pair of functions F_1, F_2 mapping x_i, θ to a mean vector and a (postive definite) covariance matrix. For example, combining a neural network and the reparametrization A.4.1. This kind of model will be extensively used in the paper.

On the contrary, time-series observations are indexed by a time variable $t \to y_t$ and correlation between different times exists. This would suggest a conditional expression as:

$$p(y_t|x_t, y_{t-1}, x_{t-1}, \ldots, y_0, x_0) = L(y_t, x_t, y_{t-1}, x_{t-1}, \ldots, y_0, x_0, \theta)$$

unfortunately, such general assumption increase the complexity for every new observation and, thus, is rarely applied.

A standard solution is to assume the existence of state parameters z_t, and the Markov property (conditionally on both regressors and states). The likelihood function is then of the kind:

$$p(y|x, \theta) = L(y_t, x_t, z_t, x_{t-1}, z_{t-1}, \theta)$$

The reasons for the Markov hypothesis can be different. The simplest is that the system generating the observations has an internal state that is not observable, and the probability of evolving of this state is uniquely determined by the state itself and the regressors x. States can evolve trough time with their own dynamic, factorising L in:

$$L(y_t|x_t, z_t, z_{t-1}, x_{t-1}, \theta) = L(y_t, x_t, z_t) D(z_t|x_t, y_{t-1}, x_{t-1}, z_{t-1}, \theta) \tag{1.3}$$

with D a known conditional probability for the states value (sometimes called transition kernel).

The choice of D is crucial and can be stochastic or deterministic.

Examples of stochastic dynamic are stochastic volatility or time-varying parameters in Vector Autoregressive [6]. Autoregressive models (as GARCH, DCC or recurrent neural networks) are examples of deterministic transition.

For example, consider a univariate 0-mean GARCH with no regressors:

$$\begin{aligned} \theta &= \omega, \alpha, \beta \\ L(y_t, z_t, x_t, \theta) &= \mathcal{N}_{0,z_t} \\ z_t &= \omega + \alpha z_{t-1} + \beta y_{t-1}^2 \end{aligned} \tag{1.4}$$

where the third condition is equivalent to a deterministic transition function D for the state z_t. DCC uses a multivariate covariance matrix as z_t and a more evolved transition mechanicsm, but retain the deterministic nature. Deatils about DCC definition can be found in [10].

Differently, a naive stochastic volatility models could be:

$$\begin{aligned} \theta &= \omega, \alpha, \beta, h^2 \\ L(y_t, z_t, x_t, \theta) &= \mathcal{N}_{0,z_t}(y_t) \\ \log(z_t) &\sim \mathcal{N}_{\omega+\alpha\log(z_{t-1})+\beta\log(y_{t-1}^2),h^2} \end{aligned} \tag{1.5}$$

that strictly resemble the above GARCH, but imply a probabilistic transition of the state z_t, conditionally log-gaussian.

ARMA processes and TVP-VAR present the same duality, in the sense that both models the conditional mean with, respectively, a deterministic transition kernel and a stochastic one (e.g. [11] for the latter).

The latent nature of z_t would require, in general, a probabilistic dynamic. A deterministic observation-driven approach could lead to multiple steps incoherent forecast densities (see [12]); however, it only requires few parameters and is thus very popular.

Although more realistic assumptions and more solid theoretical basis, inference for full stochastic models is harder and could require particle filters [13], expectation maximization [14] or a computationally intensive MCMC [6]. On the other hand, the fast inference for deterministic transition models made them very popular.

In the machine learning field, recurrent neural networks are extensively used, particularly in the form of long-short-term-memory (LSTM [15]) networks. Such networks are similar to autoregressive methods, in the sense that there is an internal state z_t and its value is deterministic, conditionally on $\theta, x_t, z_{t-1}, y_{t-1}, x_{t-1}$.

There are differences in the inference process and the theoretical structure behind, but these two approaches share common principles.

In particular, recurrent network inference truncates the number of regressive steps to speed up the process, is estimated with stochastic gradient methods, and usually implements single-value forecast. Autoregressive models, instead, apply a more rigid class of update dynamics but specify a complete distribution for the target y_t.

The possibility to conduct effective inference affects the choice of L. The next section describe principles and a few common methodologies for learning $p(\theta)$ fixed L.

1.4 Inference and training

Fixed data and the model, inference has the goal to find a posterior distribution over the parameters that forecast well on future observations, using the expression 1.1.

How to select a similar distribution? First of all, it is necessary to decide which is the 'desired' $p(\theta)$. Then, it is often required to approximate or derive such distribution $p(\theta)$, because it is not explicitly available.

From now on, the first step (deciding the target) will be referred to as inference approach, or inference framework, while the second (deriving or approximating the target) as computation, or approximation.

The latter can be done with closed-form computation or using numerical methods, for example solving an optimisation problem (as maximum likelihood or VI) or applying a sequential algorithm with asymptotic guarantees (as MCMC or EM).

Unfortunately, find an optimum could end up in local optima and stopping an asymptotically right procedure can give a result far from the proper solution. The term approximation stresses the fact that numerical and computational constraints affect inference every time that closed form is not available.

Inferential approaches can be divided into single-point and proper distributions.

The first methods select a single value for the parameters, equivalent to a Dirac Delta $p(\theta|x) = \delta_{\hat{\theta}}(\theta)$. This choice is an approximation that cancels any uncertainty about θ, despite its latent nature. The main reason for this choice is that a single value is easy to handle.

On the contrary, obtaining a proper distribution is a more general approach. Still, it is more complicate, in the same way as a distribution is a more general object than a deterministic single value.

Here are presented two single-point inference procedures: maximum likelihood and penalised maximum likelihood, and two distribution approaches: Bayesian inference and

VI. Follows a brief discussion of hybrid techniques and other methods, but all the methodologies used in the rest of the paper will belong to these four cases.

1.4.1 Maximum likelihood

A reasonable choice for θ is the one that maximise the empirical value:

$$\tilde{\theta} = \arg\max_{\theta} L(Y, \theta) \tag{1.6}$$

This approach is prevalent both in statistical modelling and in machine learning, due to its simplicity. Whenever L is a conditional probability, it is called maximum likelihood estimator. Sometimes L is a general measure of the forecast, called score.

In particular, this estimator is not reliable whenever the dimension of the parameters is high wrt the data, because it can lead to overfitting or to forecast that does not generalise well. A simple explanation is that a global maximum taken in a parameters domain that is too large can lead to unfair estimate if data does not carry enough information about the parameters (or when a single θ that represent the real dependencies does not exist).

1.4.2 Maximum penalised-likelihood

When maximum-likelihood become unreliable, a regularisation term could be added to the empirical score:

$$\tilde{\theta} = \arg\max_{\theta} L(Y, \theta) + \pi(\theta) \tag{1.7}$$

This regularization function π is often called penalization (or shrinkage) term and forces the estimated quantity to stay within a-priori reasonable region of the whole parameters space. This can lead to better results.

The most famous application is probably the lasso (and the ridge) regression [16], where the penalty shrinks the regression coefficients of a linear model towards 0 (or close to), often leading to better generalisation power and good feature selection. It is extensively applied also in econometric [6].

The shrinkage term gives information about possible $\hat{\theta}$ and the optimisation technique can exploit the penalty characteristics to speed up the inference. For example, when the penalty term is a prior distribution over Ω, sampling particles from it can suggest good starting points for the optimisation.

However, the penalty term sometimes does not solve the maximum-likelihood estimator's problems because selecting a single-value for θ still cancels the uncertainty about such latent quantity. There are contexts when this uncertainty matters.

1.4.3 Bayesian inference

The Bayesian paradigm is the usual reference to infer a complete distribution. This approach focus on the posterior $p(\theta)$, defined as:

$$p(\theta) = \frac{L(Y,\theta)\pi(\theta)}{\int_\Omega L(Y,\theta)\pi(\theta)d\theta} \tag{1.8}$$

The prior $\pi(\theta)$ is a prior distribution over Ω and represents a priori knowledge about the parameters value. The denominator is called Bayes factor, $\mathcal{B}(Y)$, and it allows to compare different models.

There is a known relation between penalized maximum-likelihood and Bayesian posterior. Whenever the penalty term and the prior term coincides, the mode of the posterior density is the penalized maximum estimator. For this reason, the latter value is often called Maximum-a-posteriori (**MAP**).

Unfortunately, the posterior is often intractable, due to the unavailable closed-form computation of $\mathcal{B}(Y)$. The estimation of this integral using standard Monte Carlo techniques, as importance sampling, is often challenging, because of its variability.

MCMC and VI are respectively, the most common procedures to conduct intractable inference.

MCMC is the standard technique to approximate the posterior. This approach creates a chain of samples asymptotically distributed as the target distribution. It is possible to estimate posterior's moments or evaluate the predictive distribution, using these particles. Details about many variants of this methodology are available in [17].

The main difference between standard Monte Carlo and MCMC is that the first build independent samples with a fixed distribution, and then uses a weighted average of this particles to compute posterior moments; the second creates a sequence of dependent samples with the posterior as stationary distribution. Usually, the next sample is randomly selected based on the previous one.

Unfortunately, the asymptotically exact MCMC does not guarantee a good approximation on finite, real, cases. In particular, it is well known that MCMC could suffers on high-dimensional problems or with highly correlated components of θ [17].

This issues could be less severe for model-specific MCMC as Gibbs samplers. However, applying model-specific methods require time and mathematical skills and is error-prone, compared to a general approach as Metropolis-Hastings or Hamiltonian MCMC.

1.4.4 Variational inference

An optimisation-based alternative is Variational Inference (VI).

This technique introduces a family of distributions Q over Ω as admissible approximations of the posterior $p(\theta)$, then selects as proposed solution $q \in Q$ the one that minimises the Kullback-Leibler divergence wrt the real posterior $KL(q, p)$.

The KL divergence, defined as:

$$KL(p, q) = \mathbb{E}_q \left[\frac{\log(q)}{\log(p)} \right] \tag{1.9}$$

is not immediatly available (because p is not available), but substituting 1.8, it holds:

$$\begin{aligned} KL(p, q) &= \mathbb{E}_q \left[-\log(L(Y, \cdot)) - \log(\pi(\cdot)) + \log(\mathcal{B}(Y)) \right] + \mathbb{E}_q \left[\log(q) \right] \\ KL(p, q) - \log(\mathcal{B}(Y)) &= -\mathbb{E}_q \left[\log(L(Y, \cdot)) + \log(\pi(\cdot)) \right] + \mathbb{E}_q \left[\log(q) \right] \end{aligned} \tag{1.10}$$

Re-arranging the terms and defining the Evidence Lower Bound ELBO as minus the second term of the last equation, it holds:

$$-KL(p, q) + \log(\mathcal{B}(Y)) = ELBO(q) \tag{1.11}$$

this shows that minimising $KL(q, p)$ is equivalent to maximising the ELBO, because $\log(\mathcal{B}(Y))$ does not depend on q.

Not only, but the positiveness of KL ensure that the ELBO is a lower bound for the Bayes factor. This property justifies the name ELBO and suggests its usage for model comparison (unfortunately, there are complications [1]).

The proposed approximation of variational inference is then

$$\begin{aligned} q &= \arg\max_{q \in Q} ELBO(q) \\ q &= \arg\max_{q \in Q} \mathbb{E}_q[\log\left(L(Y, \cdot)\pi(\cdot)\right)] - \mathbb{E}_q[q(\cdot)] \end{aligned} \tag{1.12}$$

This optimisation based approach is not completely different from maximum penalised likelihood. In fact, under the Dirac Delta assumption of single-value inference, the term in the first expectation is equivalent to the MAP target function. The second term

is the entropy distribution of q. Without it, the optimisation will collapse on a point distribution: the MAP estimate.

The general schema above can also exploit functional different from the ELBO. Targeting the ELBO is called Variational Bayes.

Additional regularisation terms could be added to the ELBO to create a custom score. For example, during representation learning this is done to force required characteristics on the proposed variables. In chapter 4 the term VI will refer to optimising a global score, sometimes not reducible to a proper model ELBO.

The set of non-singular distributions is far more complicated than single point values. For this reason, the optimisation problem domain is restricted to a fixed subset Q of the possible distributions. This restriction introduces a bias in the inference, but it can be way faster than MCMC.

Several choices for Q are available, and different way to tackle the consequent optimisation problem are available. Here follows a brief overview of the most common.

Mean Field Variational Inference MFVI historically was the more common version. This technique assume Q is the set of distributions with independent components:

$$q(\theta_1, \dots, \theta_d) = \prod_{i=1}^{d} q_i(\theta_i) \quad \forall q \in Q \tag{1.13}$$

The optimisation is done component by component, with a technique called VI-Expectation Maximization (VI-EM). This solution works brilliantly in many contexts but fails when parameters are highly correlated; see [18] for specific examples.

Black-Box Variational Inference BBVI describe a general approach that is not model specific, proposed in [19].

The idea is to use stochastic optimisation to maximise the ELBO. The technique is general in the sense that it exploits information about Q, but only requires that the likelihood and the prior are smooth functions. This absence of requirements about model functions put this class of techniques on the same level of general MCMC schemes as Metropolis-hastings or Hamiltonian MCMC.

In particular, it considers a parametric $q_\lambda \in Q$. It requires to be able to sample particles from q_λ and uses a Monte Carlo estimate of the ELBO gradient wrt to λ to maximise this quantity.

The difficulty is to find Q with the above characteristics that is not too far from the desired target distribution.

Reparametrization trick methods belongs to the broader family of BBVI. The trick requires the existence of a sampling reparametrisation of Q, then the inference follows from the application of SGD (A.5.6) with general L, π functions.

A sampling reparametrization of Q is a pair ϕ, F, with $\phi : \lambda, \epsilon \to \theta$ a smooth function and F a fixed, known, distribution, s.t.:

$$\forall\lambda, \forall\epsilon \sim F \implies \phi(\lambda, \epsilon) \sim q_\lambda$$

. In practice, by generating random numbers (ϵ) with the fixed pdf F and applying the smooth function ϕ, it is possible to generate samples from $q_\lambda, \forall\lambda$. This enables a Monte Carlo estimate of the gradient:

$$E\left[\frac{1}{MC}\sum_{i=1}^{MC}\nabla_\lambda \log\left(\frac{L(\theta_i, Y)\pi(\theta_i)}{q(\theta_i)}\right)\right] = \tag{1.14}$$

$$E\left[\nabla ELBO(q, p)\right]$$

A common example is the sampling reparametrisation of the set of gaussian variables: $F = \mathcal{N}_{0,1}$, $\lambda = (\mu, \sigma) \in \mathbb{R}\bigoplus\mathbb{R}^+$ and $\phi(\epsilon, \lambda) = \mu + \sigma\epsilon$. The reparametrization can be easily extended to the multivariate case.

Variational Autoencoders (VAE [4]) use it to estimate a distribution over the latent observation-specific parameters z_t.

The general procedure is resumed in 1.

Algorithm 1: Reparametrization trick for variational inference

Result: q_λ, a distribution that approximates the model posterior

Hyperparameter: $\nu \in \mathbb{R}, I \in \mathbb{N}^+$

Initialize $\lambda = \lambda_0$;

while $E\hat{LBO}_t$ *does not converge* **do**

Sample $\epsilon_i, \ldots, \epsilon_I \sim p(\epsilon)$;

Compute $\theta_i = \phi_k(\epsilon_i), q_i = q_\lambda(\theta_i)$;

Compute $Ł_i = Ł(\theta_i), \pi_i = \pi(\theta_i)$;

Compute $E\hat{LBO}_t = \frac{1}{MC}\sum_{i=1}^{M} C\left[\log(L_i) + \log(\pi_i) - \log(q_i)\right]$;

Update $\lambda \leftarrow \lambda + \nu \nabla_\lambda E\hat{LBO}_t$ (gradient is available through automatic differentiation)

end

It is possible to use approximated sampling reparametrisation to deal with discrete distribution (see [20]), although not used in this paper.

1.4.5 MCMC vs VI

This section presents a short discussion of the differences between MCMC or VI, depending on the context.

The accuracy, the computational effort, the scalability and the flexibility of the methodology are crucial elements to decide which method to apply.

In particular, under the accuracy point of view, standard MCMC guarantees exact asymptotic inference. VI is intrinsically approximate and, unfortunately, there is not a general theory connecting KL to usual metrics on the parameters space. However, it can be difficult to detect how far MCMC are from stationary and sometimes reaching the ergodicity could require too much time.

On the other hand, VI is usually way faster than MCMC and could reduce the required time by magnitudes [1]. MCMC are widely recognised as slow and computational intensive [17]. This reason is behind the creation of versions of MCMC that sacrifice the exactness to speed up the inference [21][22].

Scalability is another precious characteristic in the big data era. For these reasons, scalable versions of both VI (as BBVI [19]) and MCMC [21][22], are frequently studied and proposed.

However, speed and accuracy are not the only relevant elements. In practice, most of the inference algorithm requires tuning over the specific problem to obtain satisfying results.

The amount of time required to tune the method to a specific model or data is another essential characteristic. How difficult is this process in terms of mathematical expertise, coding skills, and how error-prone it is, actively affects the usability of any inference technique.

This reason is behind the success of model agnostic techniques as Hamiltonian MCMC [23]. This procedure does not exploit well the model structure as a Gibbs sampler, but it is immediately applicable.

The next chapter 2 proposes a general VI methodology with the same minimum requirements of Hamiltonian MCMC (and standard Metropolis-Hastings).

1.4.6 Other methods

A plethora of different techniques have been proposed over the decades. Although single variants can better adapt to specific models, the general principles are usually the same: calibrate the forecast on the available data, while maximising the sharpness [8]. This lead to a variety of approaches with different balancing of these two characteristics.

In many cases it is not possible to identify a single value for θ that produce an effective forecast (sacrificing too much the sharpness) and also derive a full distribution $p(\theta)$ would be difficult (due to computationally intense calibration).

For example, on time-series models, the probabilistic dynamics of the state parameters z_t could create problems. Selecting the states that simply maximise the likelihood would lead to overfitting and biased inference.

For this reason, Expectation-Maximisation (EM) algorithm [24] is a common choice. EM treat differently global parameters θ and state parameters z_t. θ estimate uses Maximum likelihood (fixed an approximation for z_t), while states estimate z_t is approximated conditionally on θ.

z_t approximation often uses a set of weighted particles, leading to particle filtering [13]. In fact, particles could help to handle the problem, resulting in greater generalisation power.

Ensemble methods move even one step behind, averaging a significant number of randomised naive forecaster to obtain a suitable one [25]. This kind of methods has become a standard reference in many forecasting competitions.

However, how to balance the sharpness and calibration is still an open point [25] also for ensemble methods. In this context these terms are often referred as heterogeneity of the weak forecasters.

A detailed discussion of the general and specifics approaches is outside the scope of this paper that only apply point-wise inference, MCMC and VI.

Chapter 2

Hierarchical Variational Inference

Inference for Bayesian probabilistic models requires to handle the model posterior, which is often intractable. MCMC methods are the standard solution, allowing to sample particles with the exact ergodic distribution. However, the time required for convergence can be unreasonable.

Variational inference (VI) approximate the posterior with a simpler distribution, choosing it within a family of distributions Q. The proposed solution is the one that minimises a fixed divergence measure wrt the true posterior, as mentioned in 1.4.4. Unfortunately, VI techniques usually force a parametric structure on Q.

This chapter introduces a novel methodology that relaxes this constraint using a semi-parametric family of distributions. The technique uses a version of the reparametrisation trick, as described in 1.4.4.

The chapter is organized as follows: section 2.1 introduces the required semi-parametric family of distributions, section 2.1.1 resumes how to use this family to conduct variational inference, 2.2.3 defines two possible variants to the methodology to adapt to specific problems, then section 2.3 presents an experiment to evaluate HVI on artificial data and discuss the results, while section 2.4 concludes presenting possible future developments.

2.1 A semi-parametric family of functions

This section defines a methodology to create a semi-parametric set of distribution Q over $\mathbb{R}^d$ that allows using the reparametrisation trick to conduct inference with the same minimum requirements as standard Metropolis-Hastings.

A single value of the parameters λ, called variational parameters, uniquely identifies a distribution $q_\lambda \in Q$. It is possible to do sampling and probability density evaluation $\forall q_\lambda \in Q$. Not only, but the gradient of the density wrt the variational parameters, $\nabla_\lambda q_\lambda(\cdot)$, is also available in closed form. This possibility allows the usage of Stochastic Gradient Descent (SGD) to optimise specific functionals, as the ELBO, over this set of distribution, to conduct VI.

2.1.1 Single layer example

Sampling particles with a known distribution and transforming them with a neural network enables to obtain particles with a flexible distribution. This technique is widely used in generative models, with excellent success [26]. However, the density of such particles is unknown, making a direct estimate of the ELBO impossible.

HVI methodology imposes two requirements to the network to allow the density evaluation, enabling to conduct VI.

For example, consider a single layer neural network ϕ that transform an input $x \in \mathcal{R}^d$ into an output $\phi(x) \in \mathcal{R}^d$ following the expression:

$$\phi(x) = A_1 g(A_0 x + b_0) + b_1 \tag{2.1}$$

where A_i is a matrix that linearly projects x into a vector in $\mathbb{R}^d$, $b_i \in \mathbb{R}^d$ is a shift vector, and g is a non-linear activation function. If A_i is non singular and g is invertible, then the function $\phi(\cdot)$ is invertible too.

If we consider random samples $x_i \in \mathcal{R}^d, i = 1, \ldots, s$ with known closed form density $p_0(x_i)$, then it holds:

$$\begin{gathered} p(\phi(x_i)) = \frac{p_0(x_i)}{|A_1||A_0||\nabla g(A_0 x_i)|} \\ \log\left[p(\phi(x_i))\right] = \log\left[p_0(x_i)\right] - \log\left[|A_1|\right] - \log\left[|\nabla g(A_0 x_i)|\right] - \log\left[|A_0|\right] \end{gathered} \tag{2.2}$$

where ∇g is the gradient of the non-linear activation function, and $|A|$ refers to the determinant of A.

This shows that, if the projection matrix is not singular and the activation function is invertible (with known gradient), then it is possible to compute the density of the output in closed form.

Not only, selecting an arbitrary point y in the output space and inverting the previous transformations ($A_1 \to A_1^{-1}, g \to g^{-1}$) it is possible to compute the density $y = \phi(x)$ and subsequently $p(y)$, if needed.

2.1.2 Multiple layers extension

Consider now a k-layers network ϕ_k, obtained composing k layers with the same structure above. Applying the k transformations to the same points $x_i \in \mathbb{R}^d$ defined before and defining $x_{j,i} = \phi_j(\dots\phi_1(x_i)), j = 1\dots k$ the transformed values using the first j layers, it holds:

$$\phi_k(x_i) = A_k g(\dots g(A_1 g(A_0 x_i))) \tag{2.3}$$

the output $\phi_k(x)$ keeps the characteristic mentioned above, having:

$$\begin{gathered} p(\phi_k(x_i)) = \frac{p_0(x_i)}{|A_0|\prod_{j=1}^k |A_j||\nabla g(A_{j-1}x_{j,i})|} \\ \log\left[p(\phi_k(x_i))\right] = \log\left[p_0(x_i)\right] - \sum_{j=1}^k \log\left[|A_j|\right] - \sum_{j=1}^k \log\left[|\nabla g(A_{j-1}x_{j,i})|\right] - \log[|A_0|] \\ x_{j,i} = A_i g(x_{j,i-1}), \forall i = 1,\dots,k \\ x_{0,i} = x_i \end{gathered} \tag{2.4}$$

Adding extra layers increase the representation power of Q. Still, they could introduce numerical instability in the density or the gradient evaluation wrt the variational parameters, causing gradient vanishing or gradient exploding.

This issue is solved choosing self-normalizing (invertible) activation functions [27], as *selu*. Numerical experiments show that expressions like 2.4 keep sufficient stability in all the explored cases.

In particular, the density expression enters the ELBO expression only through $\log(p)$, that is additive in the different layers, avoiding any interaction in the gradient expression.

2.2 Variational inference using HVI

Constructing a set $q_\lambda \in Q$, as described before, enables the application of the reparametrization trick to solve the optimization problem:

$$\tilde{\lambda} = \max_{q_\lambda \in Q} F(q_\lambda) \tag{2.5}$$

where F is any bounded functional over Q.

Consider a Bayesian model with smooth prior and smooth likelihood, taking the ELBO 1.11 as score F is possible to run approximated Bayesian inference, as described in 1.4.4.

The procedure requires to initialize λ randomly then sample a batch $x_i \sim p_0$, apply the transformations to obtain $\theta_i = \phi_k(x_i)$, evaluate the posterior numerator on this particle and estimate the gradient of ELBO wrt λ, and update λ using SGD. Repeat the sampling-updating procedure until convergence of the ELBO estimate.

Note that the gradient evaluation is automatic with automatic differentiation frameworks as TensorFlow [28].

2.2.1 The reparametrization analogy

This methodology approximates the posterior by transforming in a (semi) parametric way input particles with a fixed distribution p_0. The semi-parametric transformation, ϕ_λ, is a hierarchy of invertible and differentiable transformations.

This approach is equivalent to look for a reparametrisation of the model $\theta' = \phi_\lambda^{-1}(\theta)$ with a posterior approximated by a fixed p_0.

Adding an internal parametric state to p_0 does not break the process if the density is still available in closed form and differentiable wrt such new parameters.

It is possible to exploits such hierarchical structure in many ways. For example, selecting a specific p_0 or shaping the transformation in different ways.

2.2.2 Representation power

Which posterior can be effectively approximated by HVI is a crucial question.

Theoretical results are not developed here for any specific class of networks. However, the empirical experiment in 2.3 tests it, at least on a particular context. Not only, but there are also several approximation theorems concerning the representation power of general neural networks.

The well-known approximation theorem [29] states that a neural network with at least an internal layer and sufficiently high number of nodes can approximate any regular function. The has been generalised to different activation functions [30]. This results, however, can not be applied in this case. First of all, because HVI maps $\mathcal{R}^d \to \mathcal{R}^d$ and not $\mathcal{R}^d \to \mathcal{R}$.

Secondary, the fixed number of nodes required by HVI 2.4 does not allow to apply the theorem [30] because it requires an indefinitely high number of nodes.

Recently, [31] proved that with an internal dimension $d+4$ and sufficient depth, a net can approximate any L^1 function. Again, the theorem is not applicable, because $d+4 > d$ and it applies to real-valued functions.

The recent result in [32], on the other hand, concerns interpolation of points with the same number of nodes at every internal layer. Still, this result is not applicable, because interpolation of a finite number of points is not enough. Not only, but the theorem would also require to be able to write the target mapping between X and Ω as a differential equation.

However, this does not answer any question about the possibility to learn a satisfying approximation.

The possibility to choose any base distribution p_0 enlarge the representable set of distribution.

To empirically test the representation power of the methodology, section 2.3.1 presents a simple experiment to test HVI effectiveness.

2.2.3 Adapting HVI to specific problems

It is possible to adapt HVI changing the hierarchy of reparametrisations, ϕ_λ or the fixed posterior approximation p_0.

This section presents two examples. The first variant deals with multi-modal distribution. The other introduces a constraint on λ, significantly reducing the dimension of the variational parameters, speeding up the inference process in high dimensions.

2.2.4 Mixture variant

Choosing a mixture distribution for p_0 is possible to sample θ with a natural mixture structure.

The invertibility of the transformation ϕ_k will keep different modes separated, although allowed to change their relative density and distance.

2.2.5 Dimensionality reduction variant

It is possible to reduce the dimensionality of the semi-parametric family by fixing one or many parameters in A. This reduction in the dimension of Q tackles the expressive

power but also ease the optimisation problem and reduce the variability while estimating the gradient of ELBO.

Taking for example $A_i = I_d + L_i$, with L_i low rank and I_d the identity matrix, or assuming A_i diagonal.

During the empirical experiment that follows **HVI-low-dim**, with $q < d$, will refer to the following choice:

$$L_i = \begin{bmatrix} l_{1,1} & \cdots & l_{1,q} & 0 & \cdots & 0 \\ \vdots & l_{r,c} & \vdots & \vdots & & \\ l_{q,1} & \cdots & l_{q,q} & 0 & & \\ 0 & \cdots & 0 & 0 & & \\ \vdots & & & & \ddots & \\ 0 & & & & & 0 \end{bmatrix} \qquad i < k$$

$$L_k = \begin{bmatrix} l_{1,1} & \cdots & l_{1,q} & 0 & & \cdots & 0 \\ & \ddots & \vdots & \vdots & & & \\ & & l_{q,q} & 0 & & & \\ \vdots & \cdots & l_{q+1,q} & l_{q+1,q+1} & & & \\ & & \vdots & 0 & \ddots & & \vdots \\ & & \vdots & \vdots & & \ddots & 0 \\ l_{d,1} & \cdots & l_{d,q} & 0 & \cdots & 0 & l_{d,d} \end{bmatrix} \qquad i = k \tag{2.6}$$

equivalent to consider a standard HVI for the first q dimensions and an identity transformation for the remaining $d - k$, for every layer except the last one. The remaining A_k transformation creates every coordinate as a linear combination of the first k components, plus a diagonal transformation that keeps the density non-singular.

The number of variational parameters drops from kd^2 to $(k-1)k^2 + (k+1)d = k^3 - k^2 + kd + d$, making the problem linear in d. In the experiment 2.3.2 $k = 5$ appear to adapt well, similarly to full HVI.

2.3 Empirical experiment

This section describes the empirical experiment that tests the efficacy of the proposed methodology wrt MCMC on artificial data. The goal is to compare the ability of HVI to estimate the posterior moments to standard MCMC as Metropolis-Hastings (RWMH) and Hamiltonian (HMCMC).

The test problem will be the classical multivariate Gaussian observations (T observations of N variables) with unknown mean and unknown covariance matrix, with conjugate prior (normal-Wishart k, μ, ν, W).

The motivations behind this choice are different. First of all, the problem solution is available in closed form. Secondary, the strong correlation between parameters is known to be challenging for both MCMC [17], and VI [18]. Third, the necessity to invert the precision matrix (numerically capricious operation) is often present in real contexts.

The evaluation metrics for every inference method will be the mean squared error obtained estimating the mean and the standard deviation of the posterior distribution. The comparison will also account for the computational time and the dimension of the parameter space.

The choice of competitors is due to their similarity. They do not exploit any information about the problem, as HVI. Either HVI, RWMH and HMCMC apply to any general posterior numerator smooth function and only requires a function that maps $\mathbb{R}^d$ to the parameters domain.

The details and results of the experiment are available in the next section 2.3.1, and the results in the following 2.3.2.

2.3.1 The experiment settings

This section describes the details of the inference problem with gaussian conditional likelihood.

In this context $Y_0 \in \mathbb{R}^N, \cdots, Y_T \in \mathbb{R}^N$ are observed. The model likelihood is conditionally Gaussian $Y \sim \mathcal{N}(m, \eta^{-1})$, where m and η are the latent parameters (the mean and the precision matrix, respectively). The dimension parameters N will take three different values $N = 5, 10, 15$. The precision matrix of the true data generator will vary $\eta = I_d, 1/3I_d, 1/5I_d$ corresponding to increasing noise in the data.

The prior π for m, η is the conjugate prior for this likelihood, a Normal-Wishart distribution, with parameters k, μ, ν, W. This experiments uses the specific values $k = N, \mu = 0_N, \nu = N, W = I_N/N$, so that the prior density expression is:

$$\pi(m, \eta) = \mathcal{W}_{W,\nu}(\eta^{-1})\mathcal{N}_{\mu,\eta^{-1}}(m) \tag{2.7}$$

The experiment steps and the details about the chosen problem parameters and estimation hyperparameters:

Algorithm 2: Multivariate Gaussian inference experiment
Result: A set of particles, for every MCMC and HVI method

1. Fix the problem dimension N, the parameters number will be $d = N(N+1)/2 + N$ (the first addend for the terms of the Cholesky decomposition of the precision matrix, while the second accounts for the vector means).
2. Fix the number of observations ($T = 50$ used here).
3. Generate a vector for the prior mean $m_0 \sim \mathcal{N}(0, I_d)$ and fix the true mean (wlog 0_d is applied).
4. Generate a matrix for the prior precision $\eta_0 \sim \mathcal{W}(d, I_d/d)$ and fix the true precision matrix $\eta = cI_d$ with c that controls the level of noise in the data (in this paper $c = 1, 1/3, 1/5$ is used).
5. Fix the prior uncertainty parameters $k = T, \nu = d$.
6. Generate data $y \sim \mathcal{N}(m, \eta^{-1})$.
7. Run the different MCMC methods to be evaluated, with initial point m_0, η_0 (the prior mode). Collects the samples from 100 parallel chains and $1400, 2800, 5000$ particles for dimensions $N = 5, 10, 15$ respectively.
8. Run the different HVI methods to be evaluated, with initial distribution centred on m_0, η_0 (the prior mode), with a reasonably small deviation around it. Collect the samples at every *SGD* step. The number of learning steps is fixed to $9000, \forall N$.

Concerning HVI, the non-linear layer will be *selu* and initial distribution p_0 uniform over the hypercube. The specific SGD variant will be Adam optimiser, being probably the most common choice. The number of layers will be fixed to 5, for any different N, to avoid un unfair adaptiveness compared to the competitors MCMC.

2.3.2 Results

The results are plotted in 2.1,2.2,2.3, for problems with $N = 5, 10, 15$ respectively. In every figure, the ordinate is the mean squared error of the estimate of the true posterior moments. The abscissa is the time required to obtain the sample. At every fixed

checkpoint in time, all the particles generated by every method are used to estimate the moments, except the first 10% that goes in the burn-in set, as commonly done in MCMC inference [33]. The same applies to HVI.

For every problem, two figures are plotted. The left one refers to the parameters mean, and the other to the standard deviation. Different rows of each figure refer to different level of η (inverse of noise in the data).

From above to below the same problem with different noise in the generating process is compared (precision $\eta = 1, 1/3, 1/5$).

VI better estimates the expected value of the parameters, on every dimension. The standard deviation estimate of HVI is slightly better for $N = 5$, similar for $N = 10$ and slightly worst for $N = 15$. VI difficulty in estimating the deviations is a well-known characteristic, caused by the ELBO expression [1]. Another peculiarity is the limited representative power of HVI that let the estimated error converge to a value greater than 0, while theoretically the MCMC converge exactly to 0.

Concerning different levels of η, results are similar across methodologies, although the overall precision in the estimate decreases, as expected.

2.4 Discussion and future developments

The proposed HVI methodology requires the same, minimum, effort of a standard Metropolis-Hastings to be applied, in terms of calculus. This fact makes it a viable and flexible alternative that can better approximate the posterior on short runtime, as shown in 2.3.1. In particular, HVI exhibits the known behaviour of VI methods: it underestimates parameters dispersion but better measures the posterior mean, compared to MCMC.

There are open issues still to be investigated, about the new methodology. For example, no theoretical results about the representation power of HVI have been developed yet. Not only, under which conditions HVI works better than MCMC would require other empirical investigations. If sharp precision is needed or enormous computational time is available, probably the right answer is MCMC. However, during prototyping and exploration, HVI could be a better choice. Secondly, MCMC and HVI could be combined. For example, using the latter for prototyping or evaluating a good initialisation point for MCMC, or developing specific hybrid strategies (similarly to [34]).

Chapter 4 applies HVI on real data.

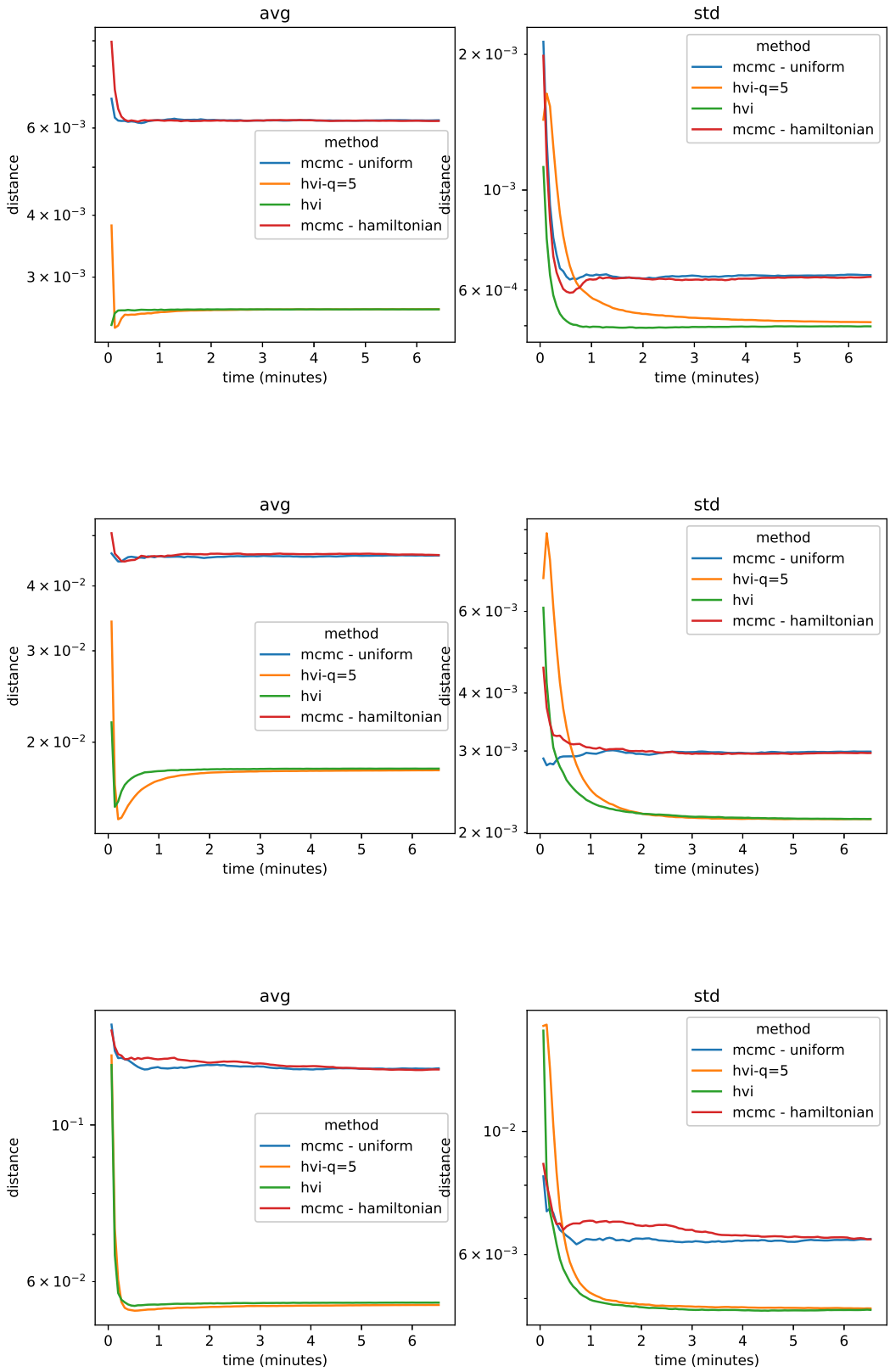

FIGURE 2.1: Plot of the estimation error for the experiments with dimension $N = 5$ and different precision values ($\eta = 1, 1/3, 1/5$). Abscissa represents the computational time, the ordinate the mean-square-error (in log scale). Left column refers to estimate of the expected value, right to the standard deviation of the parameters distribution.

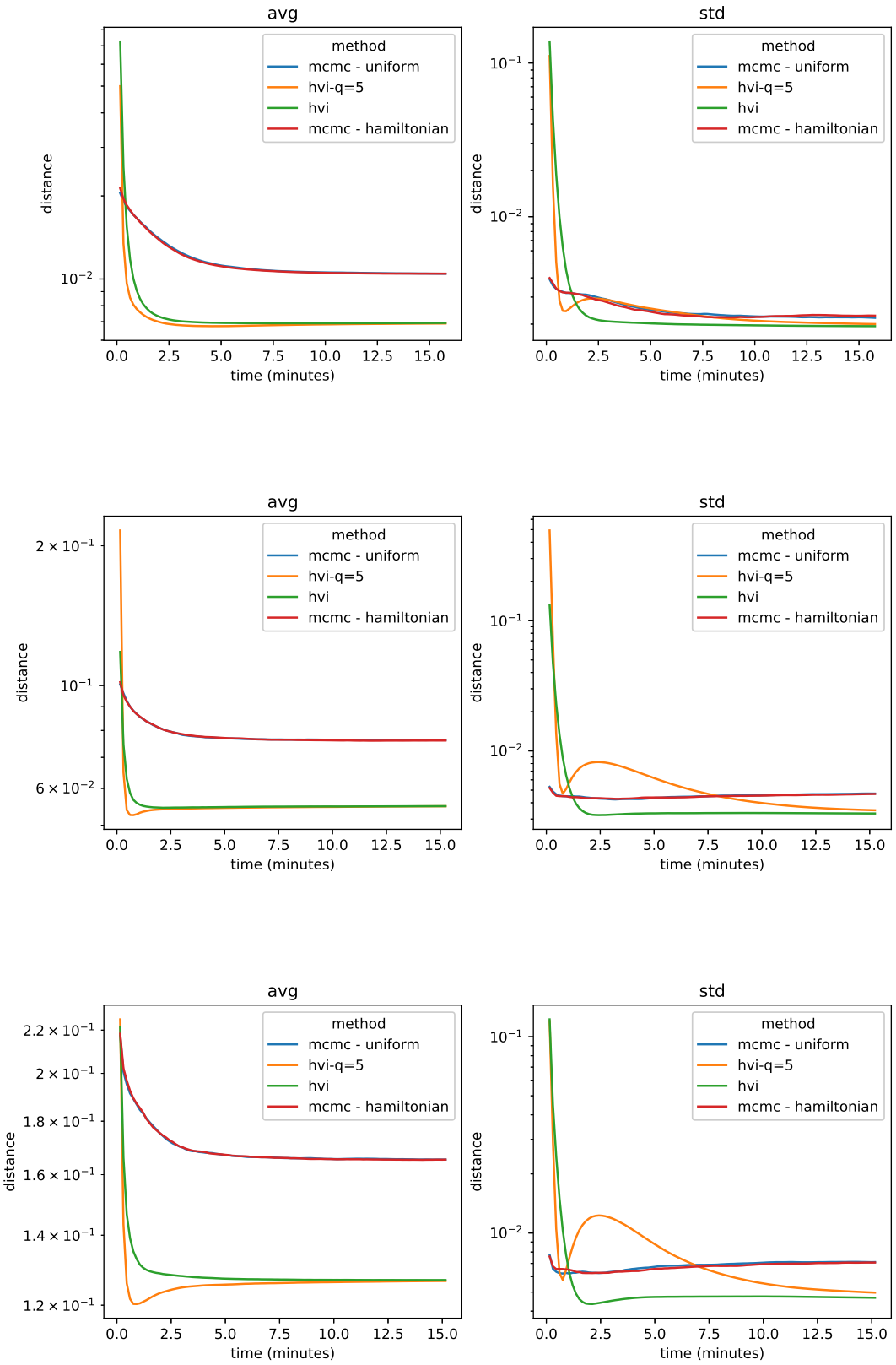

FIGURE 2.2: Plot of the estimation error for the experiments with dimension $N = 10$ and different precision values ($\eta = 1, 1/3, 1/5$). Abscissa represents the computational time, the ordinate the mean-square-error (in log scale). Left column refers to estimate of the expected value, right to the standard deviation of the parameters distribution.

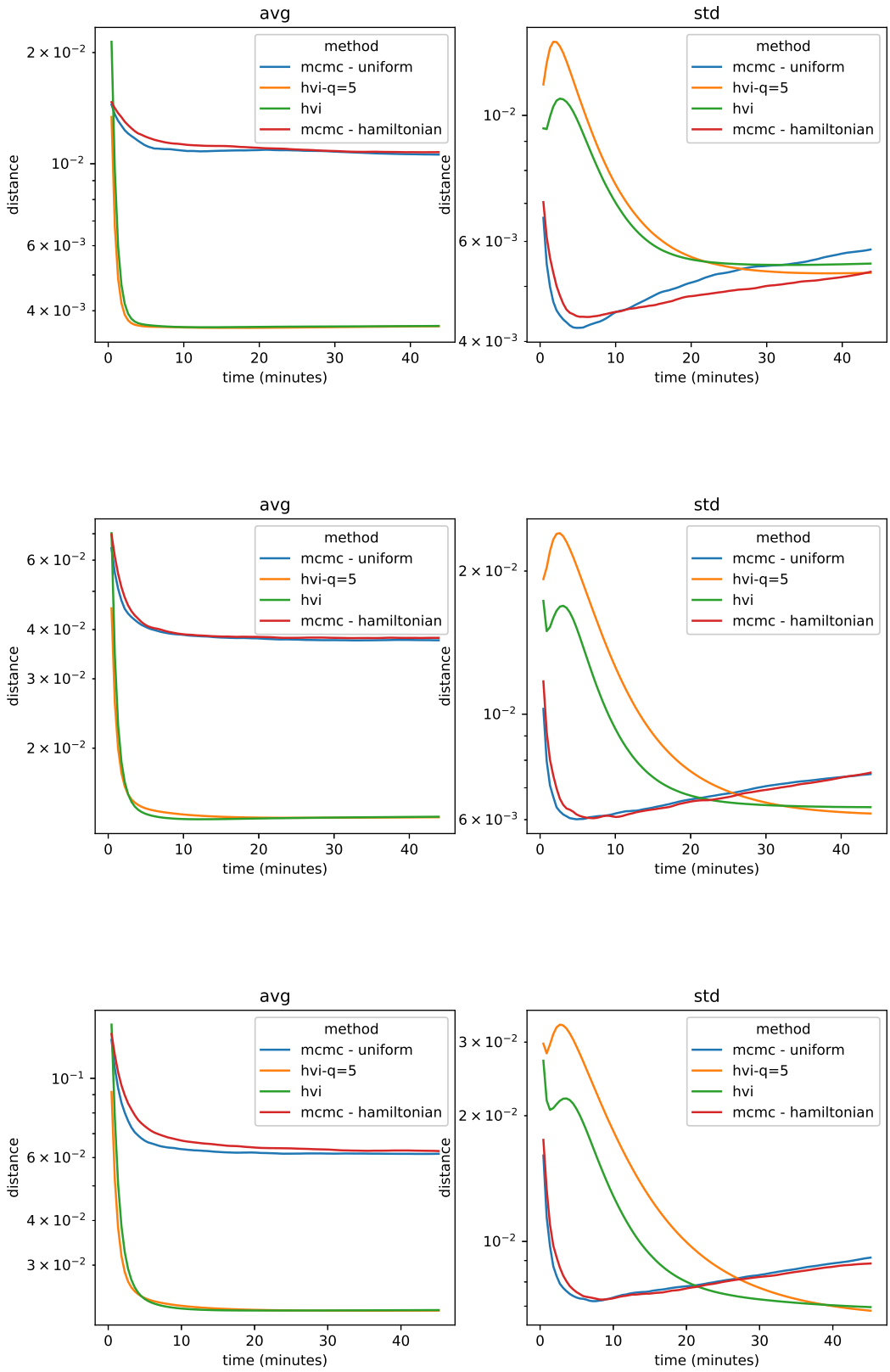

FIGURE 2.3: Plot of the estimation error for the experiments with dimension $N = 15$ and different precision values ($\eta = 1, 1/3, 1/5$). Abscissa represents the computational time, the ordinate the mean-square-error (in log scale). Left column refers to estimate of the expected value, right to the standard deviation of the parameters distribution.

Chapter 3

A Python package for multivariate time-series

This chapter describes the Python repository available at:

https://github.com/BolzMattia/ThesisCode.git.

The package implements different kinds of multivariate time-series density forecaster with semi-parametric VI.

This chapter presents the repository building a complete example of forecasting exercise: section 3.1 imports data and split train and test; 3.2 shows how to compute features, including standard mean-covariance estimators as DCC and Ledoit-Wolf; section 3.4.1 show examples of inference and forecasting for various semi-parametric models; 3.4.2 shows how to apply HVI to mean-covariance estimate; short conclusion and comments in 3.5.

The models list vary from standard DCC [10], standard static Ledoit-Wolf [35], a Bayesian estimate of covariance matrix using HVI and Ledoit-Wolf estimator, standard and Bayesian Multilinear Projector A.5.1, autoencoders (both standard A.5.2 and variationals A.5.2.1), and Long-Short-Term-Memory LSTM A.5.3.1.

3.1 Data wrapper

To store data and features the package uses class *features.wrapperDataFrame* a wrapper of the standard *pandas.DataFrame*.

This wrapper allows storing different features even with complex structure. For example, to read a .csv with this class and to split it between train and test set:

```
dsf = wrapperPandasCsv(r'fileName.csv')
dsf.split_train_test(sequential=True, test_size=0.3)
```

In this example, the first 70% of the observations go into the train set. A complete description of the parameters is available in the code comments, in the repository.

To access, for example, the time vector or the train observation or the test values, use the correspondent attributes:

```
dsf.indexes_train, dsf.indexes_test # Arrays with the time values
dsf.x_train,dsf.x_test # Dictionaries with the observations stored with name 'y'
```

3.2 Feature engineering

Every feature is stored with a name in the up-mentioned dictionaries and can have a complex internal structure.

For example, to estimate a DCC [10] model, and use the estimated mean and variance as features, it is enough to write the following instructions

```
import timeseries_transformation as T
import mean_covariance_models as M

# Adds the dcc states as features with the name 'dcc'
dsf.withColumn('dcc', *T.dcc(*dsf['y']))
# Adds another feature with the linearisation of 'dcc'
states_line = [M.mu_vcv_linearize(**states) for states in dsf['dcc']]
dsf.withColumn('dcc_line', *states_line)
```

The *withColumn* method that assign the results of the function *T.dcc* to the feature named *dcc*. The * is standard Python code that tells the interpreter to pass the pair of features named *y* (train and test values) as two different input.

The linearisation function *M.mu_vcv_linearize* apply the inverse of the reparametrisation A.4.1 to the DCC states estimate.

A complete list of feature transformations is available in the documentation.

3.3 Optimizer

Most models in this package rely on SGD to make the inference using the Adam optimizer. It is straightforward to extend the wrapper to use optimizers available in TensorFlow [28].

All the current available models will use the following:

```
import optimizers as O
learning_rate = 1e-4
beta_1 = 0.9
optimizer = O.optimizer_adam(learning_rate=learning_rate, beta_1=beta_1)
```

3.4 The abstract models

This section describes the general models and their subtypes, with coding examples. Different inference techniques are available for several learners. All learners use mean-covariance state variables with the correspondant Gaussian likelihood.

The first kind of models assumes a Markowitz property for the conditional distribution of the states z_t given regressors x_t and z_{t-1}. Amortized VI inference, with the reparametrisation trick, will take care of the training phase as described in 3.4.1.

Another approach uses HVI to estimate a static distribution on mean-covariance states, centred on a given initial estimate (as Ledoit-Wolf estimator, for example) 3.4.2.

3.4.1 Semi-parametric models

The model assumes Markowitz property conditional to the set of features and the previous state. The mean-covariance states z_t and the target variable distribution $p(y_t)$ will depend on a specific model function M and its global parameters θ:

$$\begin{aligned} z_t &= M(\theta, x_t, z_{t-1}) \\ p(y_t) &= \mathcal{N}_{z_t}(y_t) \end{aligned} \tag{3.1}$$

Whenever the inference procedure return a non-singular $p(\theta)$ the forecasting distribution is

$$p(y_t) = \int_\Omega \mathcal{N}_{z_t(\theta, z_{t-1})}(y_t) d\theta \tag{3.2}$$

the integral is approximated using standard MC sampling.

The single models differ in the choice of the mapping M. In the package, different choices are available. All of them shares the following structure: compose a Neural Networks mapping the input into $\mathbb{R}^d$ with the parametrisation in A.4.1.

M can contribute to the score function with an additional regularisation term. Consider, for example, an autoencoder loss, or any penalty over its internal parameters.

The scheme above is equivalent to amortized inference [36].

All the M choices will depend on an initialisation point, $z_{t,0}$. The class constructor takes such value as input. Specifically

$$z_{0,t} = r^{-1}(\mu_{0,t}, V_{0,t})$$

where r^{-1} is the inverse of the reparametrisation A.4.1, and μ_t, V_t general mean-covariance admissible states. The following example will use DCC or Ledoit-Wolf to estimate $\mu_{0,t}, V_{0,t}$.

The model class also accounts for a regularization penalty on z_t. The default penalty has the form of an empirical prior distribution on z_t (see A.6 for the definition).

3.4.1.1 Multilinear projector, MLP

MLP is the simplest type of neural network, as described in A.5.1.

The code required to create an instance of this predictor, run the learning process with the fit mehtod and predict the density on new data:

```
from learners import CholeskyMLP
feature_name = 'x'
target_name = 'y'
initialization_states_linearized = 'dcc_line'
hidden_layer_dim = 32
layers_number = 5
m = CholeskyMLP(dsf,
                #The feature names for the regressors, target and center
                'x', 'y', initialization_states_linearized,
                #Set to True for a gaussian posterior approximation
                #instead of a Dirac Delta, for the parameters
                gaussian_posterior=False,
                #The network hidden layers structure
                hidden_layers_dim=[hidden_layer_dim for i in range(layers_number)],
                #The hyperparameters of the optimization process
                init_scale=1e0, learning_rate=1e-4, beta_1=0.1)
m.fit(dsf, epochs=single_round_epochs, batch_size=batch_size, verbose=0)
states, theta, llkl, p0, log_p = m.density_forecast_multi_particles(
                                    dsf, on_test=True)
```

Setting the parameters *gaussian_posterior* to true will apply the standard gaussian reparametrisation trick to infer a diagonal gaussian posterior on the parameters.

3.4.1.2 Autoencoder

Autoencoders can be seen as regularised versions of the above MLP, as described in A.5.2.

The code to train and forecast such a neural network is:

```
from learners import CholeskyAutoEncoder

feature_name = 'x'
target_name = 'y'
initialization_states_linearized = 'dcc_line'

hidden_layer_dim = 32
encoder_layers_number = 5
encoder_layers = [hidden_layer_dim for i in range(encoder_layers_number)]
decoder_layers_number = 5
decoder_layers = [hidden_layer_dim for i in range(decoder_layers_number)]
forecaster_layers_number = 5
forecaster_layers = [hidden_layer_dim for i in range(forecaster_layers_number)]

m = CholeskyAutoEncoder(dsf,
        #The feature names for the regressors, target and center
        'x', 'y', initialization_states_linearized,
        # The net structure
        encoder_layers_dim=encoder_layers,
        encode_dim=encode_dim,
        decoder_layers_dim=decoder_layers,
        #Set to True to use a Variational AutoEncoder
        variational_reparametrization=True,
        #The hyperparameters of the optimization process
        init_scale=5e-1, learning_rate=2e-5)

m.fit(dsf, epochs=single_round_epochs, batch_size=batch_size, verbose=0)
states, theta, llkl, p0, log_p = m.density_forecast_multi_particles(
                                    dsf,on_test=True)
```

3.4.1.3 Variational Autoencoder

Setting the parameter *variational_reparametrization* equal to true will create a variational autoencoder A.5.2.1, [4].

3.4.1.4 LSTM

LSTM is a recurrent neural network described in A.5.3.1.

It can be used as a forecaster with:

```
from learners import CholeskyLSTM
feature_name = 'x'
target_name = 'y'
initialization_states_linearized = 'dcc_line'
recurrent_dim = 5
hidden_layer_dim = 32
layers_number = 5
m = CholeskyLSTM(dsf,
                 #The feature names for the regressors, target and center
                 'x', 'y', initialization_states_linearized,
                 #The network hidden layers structure
                 recurrent_dim=recurrent_dim,
                 hidden_layers_dim=[hidden_layer_dim for i in range(layers_number)],
                 #The hyperparameters of the optimization process
                 init_scale=1e0, learning_rate=1e-4, beta_1=0.1)
m.fit(dsf, epochs=single_round_epochs, batch_size=batch_size, verbose=0)
states, theta, llkl, p0, log_p = m.density_forecast_multi_particles(
                                    dsf,on_test=True)
```

3.4.2 HVI for covariance matrix

This section presents the code to conduct a static Bayesian estimate of the mean and covariance matrix of a population using HVI, centred around the Ledoit-Wolf estimator defined in [35].

Sampling vector particles with the right dimension and applying the reparametrization A.4.1 it is possible to obtain mean-covariance samples. It is possible to centre the sampler around such estimator, speeding up HVI, by using a standard estimator for this latent quantities.

Example code:

```
from learners import HVIstaticMuVcv
#Compute the center estimator
baseline_estimator = 'lw'
dsf.withColumn(baseline_estimator, *T.staticLedoitWolf(*dsf['y']))
#Linearize it inverting the reparametrization
states_line = [M.mu_vcv_linearize(**states) for states in dsf[baseline_estimator]]
mu_vcv_linearized_name = 'lw_line'
dsf.withColumn(baseline_model_line, *states_line)

target_name = 'y'
layers_number = 5
m = HVIstaticMuVcv(
            dsf, 'y', mu_vcv_linearized_name,
            init_scale=6e-2, learning_rate=2e-5)

m.fit(dsf, epochs=single_round_epochs, batch_size=batch_size, verbose=0)
states, theta, llkl, p0, log_p = m.density_forecast_multi_particles(
                                    dsf,on_test=True)
```

3.5 Future developments

The package repository is available at

https://github.com/BolzMattia/ThesisCode.git.

Future development will implement VI for different kind of networks.

The author will warmly welcome any help, coming in the form of bug signalling, or improvements suggestion. Such contributions can be made directly pushing modifications on new branches of the GitHub repository.

Chapter 4

Comparison of mean-covariance estimators

This chapter compares different probabilistic forecasters for time-series on two real datasets, a financial one and a macroeconomic one. The forecasters set comprehend three versions for every model, a VI version, a maximum penalized-likelihood and a standard maximum likelihood. These models depend on an initialization point that is chosen within two well known standard forecasters from literature. Such initialization points are also natural baselines to evaluate the performances. Results will show that maximum likelihood and penalized maximum likelihood cannot improve the initialization performances, with few exceptions. In contrast, VI increases, in most cases, the forecaster quality in terms of the bias-variance tradeoff.

A comparison between models arises naturally. However, the goal of this chapter is to describe the common patterns across models, more than identify the best model in this specific context.

The models implementation code is available in the package named in the previous chapter.

Overview of the dataset is available in 4.1. The comparison relies on in-sample (is) and out-of-sample (oos) observations split. The comparison metrics include both the average and quantiles of the oos log-likelihood to evaluate the bias-variance tradeoff between different forecasters.

All the approaches rely on a mean-covariance likelihood function.

The chapter starts presenting data in 4.1; follows a complete description of the selected models 4.2; the experiment results are shown in 4.3; discussion of the outcome can be found in 4.4.

4.1 Data

The comparison uses two datasets, to better assess common patterns. In fact, results will show commonalities between the different methods performances on the two sets of data, as described in 4.3.

4.1.1 Financial returns dataset

The first dataset, called **PTF**, contains closing prices of a collection of financial instruments prices (stock indexes, bond indexes and currencies) weekly measured (every 5 days, excluding bank holidays). Observations goes from $25^{th} Jun$ 1999 to $24^{th} May$ 2019. The dataset counts 15 series and 1041 closing prices for each of them. The first two-third observations go into the training set, and the remaining in the test set.

The motivation for the weekly measures is that portfolio managers often uses this time-horizon for risk assessment and weights rebalancing. Although many papers works on daily observations this is considered unpractical in the asset manager industry, both from theoretical and practical point of view [37][38].

The choice of the financial indexes aim to reproduce a real portfolio with EUR as numeraire. The list has been suggested by professionals of the industry and contains currencies spot changes (USD and GBP), governative bond (both short-term and long-term), high-yield corporate bond (both European and US), main equity indexes around the world (Eurostoxx 50, S&P 500, NASDAQ 100, FTSE 100, MSCI China, MSCI emerging market) and commodities indexes (Physical gold, Oil index and the general Dow Jones commodities index).

The probabilistic forecast focus on the log-returns of the prices. This means that, said $p_{t-1,i}$ and $p_{t,i}$ two adjacent closing prices of the i^{th} instrument, the target variable is $y_{t,i} = \log(p_{t,i}/p_{t-1,i})$. Quantiles and moments of the log-returns are available in table 4.1. Table 4.1 lists the instruments and a few statistics of the log-variations. 4.1 Plots their values.

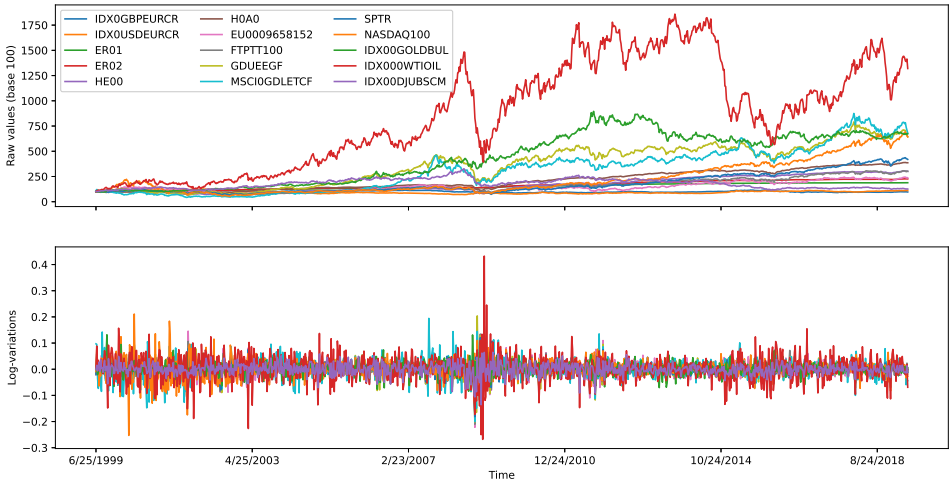

FIGURE 4.1: Plot of the **PTF** data. Above figure shows the price (or index value), rebased to 100. Below the log-returns for every instrument (or index).

4.1.1.1 Features

The set of features at every time-index, named x_t, will be the raw collection of every observation of the previous 6 weeks. Not only, cross-products between pair of variables are added to the features set. The total number of features is 1440. The only preprocessing is the usual mean centering and standard deviation scaling.

4.1.2 Macroeconomic index dataset

The second dataset, called **FRED**, contains monthly measures of various econometric indexes in the US. Data are available at St.Louis FED and cover from 1959 to 2015. The chioce of the dataset is motivated by its extensive usage in literature [39][40][41].

The problem will focus on a subset of 10 series of the whole available FRED data, with 675 values each. Again, the first two-third observations go into the training set, and the remaining in the test set. The probabilistic forecasting will deal with their log-variations instead of the raw values.

The reason behind the choice of the series is to represent the main macroeconomics measures as internal product, consumer prices index, real personal income, unemployment rate and monetary mass (M2) together with financial measures as risk-free return and S&P 500 dividend yield. The list selects the most representative series for every section of the original FRED dataset.

Series name (Series description)	mean	std	min	25%	50%	75%	max
IDX0GBPEURCR (GBP/EUR spot)	-0.0	0.0115	-0.0493	-0.007	-0.0003	0.0074	0.077
IDX0USDEURCR (USD/EUR spot)	0.0001	0.0136	-0.049	-0.0088	-0.0001	0.0083	0.062
ER01 (EU Corp, 1-3y)	0.0006	0.0017	-0.0142	-0.0002	0.0005	0.0015	0.0066
ER02 (EU Corp, 3+y)	0.0008	0.0034	-0.0307	-0.0009	0.0009	0.0027	0.0119
HE00 (EU bond HY)	0.0011	0.0128	-0.1318	-0.0023	0.0016	0.0053	0.0973
H0A0 (US bond HY)	0.0013	0.0097	-0.1086	-0.0019	0.002	0.005	0.055
EU0009658152 (Eurostoxx 50)	0.0008	0.0298	-0.2219	-0.0158	0.0032	0.0179	0.1456
FTPTT100 (FTSE 100)	0.0011	0.0236	-0.2101	-0.0115	0.0027	0.0141	0.1345
GDUEEGF (MSCI emrg mkt)	0.0018	0.0298	-0.2016	-0.0141	0.0034	0.0184	0.2038
MSCI0GDLETCF (MSCI China)	0.0018	0.0387	-0.1996	-0.0192	0.0036	0.0239	0.1943
SPTR (S&P 500)	0.0014	0.0241	-0.1814	-0.0103	0.0024	0.0141	0.1209
NASDAQ100 (Nasdaq 100)	0.0018	0.0355	-0.253	-0.0143	0.0037	0.0196	0.2109
IDX00GOLDBUL (Physical Gold)	0.0018	0.0243	-0.0841	-0.0114	0.0027	0.016	0.1316
IDX000WTIOIL (DJ Oil idx)	0.0025	0.0518	-0.2682	-0.0258	0.004	0.0324	0.4325
IDX00DJUBSCM (DJ Commod idx)	0.0002	0.022	-0.1357	-0.0118	0.0006	0.0135	0.0648

TABLE 4.1: List of titles of dataset PTF, with few statistics.

Figure 4.2 shows their values, and table 4.2 their names and a few statistics of the log-variations.

4.1.2.1 Features

The set of features at every time-index, named x_t, will be the raw collection of every observation of the previous semester. Not only, cross-products between pair of variables are added to the features set. The total number of features is 660. The only preprocessing is the usual mean centering and standard deviation scaling.

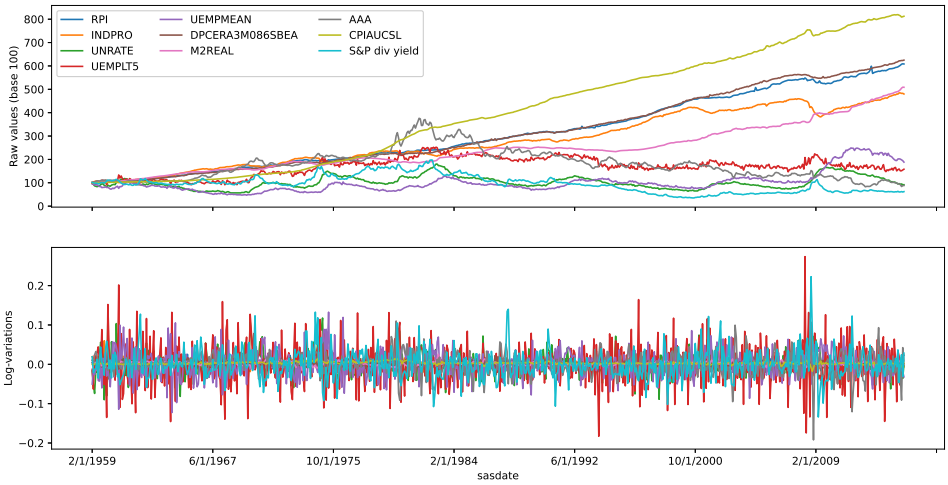

FIGURE 4.2: Plot of the **FRED** data. Above figure shows the index value, rebased to 100. Below the log-variations for every series.

Series name (Series description)	mean	std	min	25%	50%	75%	max
RPI (Real Personal Income)	0.0027	0.0054	-0.0531	0.0007	0.0029	0.0049	0.0357
INDPRO (Internal Product)	0.0023	0.0082	-0.043	-0.0015	0.0028	0.0067	0.06
UNRATE (Unemployment Rate)	-0.0001	0.03	-0.0902	-0.0194	0.0	0.018	0.1178
UEMPLT5 (Unemply 5- weeks)	0.0007	0.0539	-0.1828	-0.0316	0.0019	0.033	0.2737
UEMPMEAN (Avg unemply weeks)	0.0009	0.0345	-0.1229	-0.0212	0.0	0.0226	0.1326
DPCERA3M086SBEA (Real consumption)	0.0027	0.0053	-0.0264	-0.0002	0.0027	0.0058	0.0237
M2REAL (M2 Monetary stock)	0.0024	0.0048	-0.0152	-0.0008	0.0024	0.0052	0.0306
AAA (Moody's AAA yld)	-0.0002	0.0271	-0.1922	-0.0137	0.0	0.0138	0.11
CPIAUCSL (Consumer Price Idx)	0.0031	0.0032	-0.0179	0.0013	0.0027	0.0046	0.0179
S&P div yield (S&P common div yld)	-0.0007	0.0364	-0.134	-0.023	-0.0028	0.0166	0.2227

TABLE 4.2: List of **FRED** macroeconomics indexes, with few statistics

4.2 Models

The experiment compares models of different nature, using various inference techniques. This chapter lists and name such forecast techniques.

4.2.1 describe the first two, DCC and LW. These are standard parametric models from literature. Another model is a Bayesian version of LW, estimated using HVI 4.2.1.1. The last kind of models uses semi-parametric learners (neural networks) to map x_t into mean-covariance estimate μ_t, V_t. The parametric estimate of μ_t, V_t (obtained by DCC or LW) will be the initialisation point for such learners. The inference will apply both Variational and single-point techniques, as described in 4.2.2. This will produce single-point estimate and whole distribution over μ_t, V_t, depending on the specific approach.

4.2.1 Parametric models from literature

DCC [10] and the Ledoit-Wolf (**LW**) covariance estimator [35] will be the baseline for performances models.

Ledoit-Wolf static covariance estimator is an improved version of the sample covariance, tailored for high dimensions. Its static nature does not consider heteroskedasticity, making it a high bias-low variance estimator.

DCC is an autoregressive mean-covariance estimator. It computes deterministically, at every time-index, V_t with a convex combination of a transformation of the observations cross-products and the estimate at the previous time-index. DCC has theoretical issues and could lead to instable estimation [12]. However, it is considered an excellent default predictor for multivariate time-series. Such methodology is a low bias-high variance side of the tradeoff.

The central states of a semi-parametric technique will be called $\mu_{0,t}, V_{0,t}$. $z_{0,t}$ refers to the linearized version of $\mu_{0,t}, V_{0,t}$ obtained applying the inverse of the reparametrization in A.4.1. $z_{0,t}$ is also used to construct an empirical prior for penalised and Variational inference, following A.6.

4.2.1.1 HVI extension

Using the reparametrisation A.4.1 and the LW point estimate, it is possible to use the HVI methodology to estimate a complete distribution over μ_t, V_t.

Specifically, the location value of the estimate will be the point estimate of LW. The scale value is taken small enough to avoid numerical instability; anyway, it does not affect the final results significantly.

This forecaster name will be **HVI-LW**.

4.2.2 Semi-parametric models

It is possible to use a neural network to map regressors x_t to $e_t \in \mathbb{R}^d$ and then apply the reparametrization A.4.1 to obtain an estimate of the latent states μ_t, V_t.

Specifically, called r the parametrisation function A.4.1, the estimate will be:

$$\begin{aligned} \mu_t, V_T &= r(e_t + z_{0,t}) \\ z_{0,t} &= r^{-1}(\mu_{0,t}, V_{0,t}) \end{aligned} \tag{4.1}$$

with $\mu_{0,t}, V_{0,t}$ being an initialisation point-estimate of mean-covariance states. Different networks will connect $x_t \to e_t$. Also the initial point will vary, using both LW and DCC.

The inference scheme will vary too, between maximum likelihood, MAP, or VI. The last two require a penalty term. 4.2.2.5 describe the empirical penalty (or prior) applied for both. The initialisation point $\mu_{0,t}, V_{0,t}$ is used to compute such empirical term.

Next sections present the full list of semi-parametric procedures.

4.2.2.1 MLP

MLP is a standard multilinear projector, as defined in A.5.1. Specifically, MLP alternates 2 affine transformations to *selu* activation function.

Depending on the reparametrisation centre, the methodology will be called **LW-MLP** or **DCC-MLP**.

4.2.2.2 LSTM

LSTM uses a Long-short-term-memory A.5.3.1 single layer, followed by a two layers MLP. The activation function is *selu*.

Depending on the reparametrisation centre, the methodology will be called **LW-LSTM** or **DCC-LSTM**.

4.2.2.3 Autoencoder

The third kind of network is an autoencoder with a quadratic reconstruction loss regularisation. See A.5.2 for the details.

In particular, two MLP-layers with *selu* activation function, transform x_t to the encoded variable x'_t and then other two MLP-layers recode x'_t in $\tilde{x}_t$. Other two MLP-layers then maps x'_t to z_t. $mse(x_t, \tilde{x}_t)$ is added to the $ELBO$ estimate, acting as regularization. This correction does not allow to interpret this technique as a proper model, but it still generates a coherent forecaster.

Depending on the reparametrisation centre, the methodology will be called **LW-AE** or **DCC-AE**.

4.2.2.4 Variational autoencoder

This network is a variational variant of the previous one 4.2.2.3, as described in A.5.2.1. Specifically, the first two layers connects x_t to a pair of vectors μ_t, σ_t that represents the mean and standard deviations of the encoded variables $x'_t \in \mathbb{R}^{d'}$. Then, a single sample of the encoded information x'_t is generated as

$$x'_t = \mu_t + \epsilon_t \sigma_t, \epsilon_t \sim \mathcal{N}_{0^{d'},1^{d'}} \tag{4.2}$$

The entropy of x'_t is added as a regularisation term to the ELBO. Again, the correction does not allow to interpret this technique as a proper model, but it still generates a coherent forecaster. Notice that is still possible to use backpropagation to estimate the gradient of the score through the network parameters, being able to compute the derivative of the entropy wrt them, as in A.5.2.1.

Depending on the reparametrisation centre, the methodology will be called **LW-AE-VI** or **DCC-AE-VI**.

4.2.2.5 Empirical prior with 0-state

Empirical priors have shown effective on econometric and financial data [6]. For this reason, this paper uses an empirical regularization term on z_t, similar to the ones in [6], [11].

The assumption is that z_t follows a random-walk. An empirical estimate of the mean and covariance of the Gaussian transition kernel of such random-walk is estimated from the initialisation point $z_{0,t}$. See A.6 for the details.

4.3 Results

All the forecasting techniques (except for **DCC** and **LW**) rely on SGD training. When to stop the training process of SGD is still an open problem [42]. For this reason, all the figures shows the out-of-sample forecasting metrics (the ordinate) along the inference process (abscissa), to account for different stopping choices, as 4.3.

The twelve plots in 4.3, 4.5, 4.5 show the results obtained combining the two datasets (**PTF**,**FRED**) with the two centre estimators (**DCC**,**LW**) and the three kind of neural networks (**MLP**,**AE**,**LSTM**).

Every plot compare the same kind of network, centred on the same initial point on the same data, but estimated with maximum likelihood, penalised maximum-likelihood (**p-**) and Variational inference (**VI-**). The quadruplet data-center-model-inference approach will name the specific results, e.g. **PTF-DCC-AE-p**. ***** refers to all the choices for that slot of the quadruplet.

Similarly, HVI estimate of a standard forecaster is pointed by a triplet as **PTF-LW-HVI** and **FRED-LW-HVI**.

The maximum likelihood estimator perform less than the corresponding centre, in almost every case. This due to the high dimensionality of the problem, both as number of regressors and mean-covariance, that combines with the high semi-parametric dimension leading to overfit.

The penalised version often lead to a forecaster similar to the central estimate. This because both **DCC** and **LW** are a minimum for the penalty, being centred on their estimate, and a local maximum, being obtained via maximum likelihood. This two elements attract the penalised estimate around the centre. See for example **PTF-LW-AE-p** or **FRED-DCC-MLP-p** or **PTF-LW-LSTM-p**.

In other situations, the penalty was not enough to avoid overfitting, leading to bad results similar to maximum-likelihood. See for example **FRED-LW-MLP-p** or **PTF-DCC-AE-p** or **FRED-LW-LSTM-p**.

The variance of any element of ***-*-*-VI** is less than the variance of the original log-likelihood of the baseline model, except for **PTF-LW-AE-VI** and **FRED-LW-LSTM-VI**, where the variance is greater than the initialisation point.

The average log-likelihood of ***-*-*-VI** is almost equal to the baseline model in every case, except **PTF-LW-LSTM-VI**, **PTF-LW-MLP-VI**, and **PTF-LW-AE-VI**. The first case shows a slightly lower average than the centre, the second and the third a slightly bigger one, compared to the baselines **PTF-LW**,**PTF-LW**,**FRED-LW**.

This resume to say that **VI** shows worst performances than the centre estimation, in terms of bias-variance tradeoff, in one case (**PTF-LW-AE-VI** with greater log-likelihood variance), better perfomances in ten cases and uncomparable performances in one case (**PTF-LW-AE-VI** with higher average but also higher variance).

The performance increase is mainly due to variance reduction. This could be expected, because spreading probability mass across different parameter value can increase the sharpness of the forecast [8].

Figure 4.7 shows the volatility estimate of the first series of the data for **PTF-DCC**,**PTF-LW**,**FRED-DCC**,**FRED-LW**. For every pair, the considered semi-parametric is the one with the lowest log-likelihood variance.

VI tends to overestimate the volatility wrt the single-point estimators, in most cases. This is explicable considering the convexity of the Gaussian log-likelihood wrt the diagonal elements of the covariance. In fact, spreading probability mass around regions with higher volatility reduce the log-likelihood less than spreading around low volatility regions. It is possible for the model to 'better trade' log-likelihood for entropy, thus optimising ELBO, in high volatility regions.

HVI-LW shows better perfomances wrt standard point inference **LW**. Again, the increase comes from a lower variability of the log-likelihood, with almost the same average log-likelihood value.

Again, VI inference predicts higher volatility (both median and mean) than the original method in almost every case.

4.4 Conclusions

Across different data and different semi-parametric models, there are common patterns.

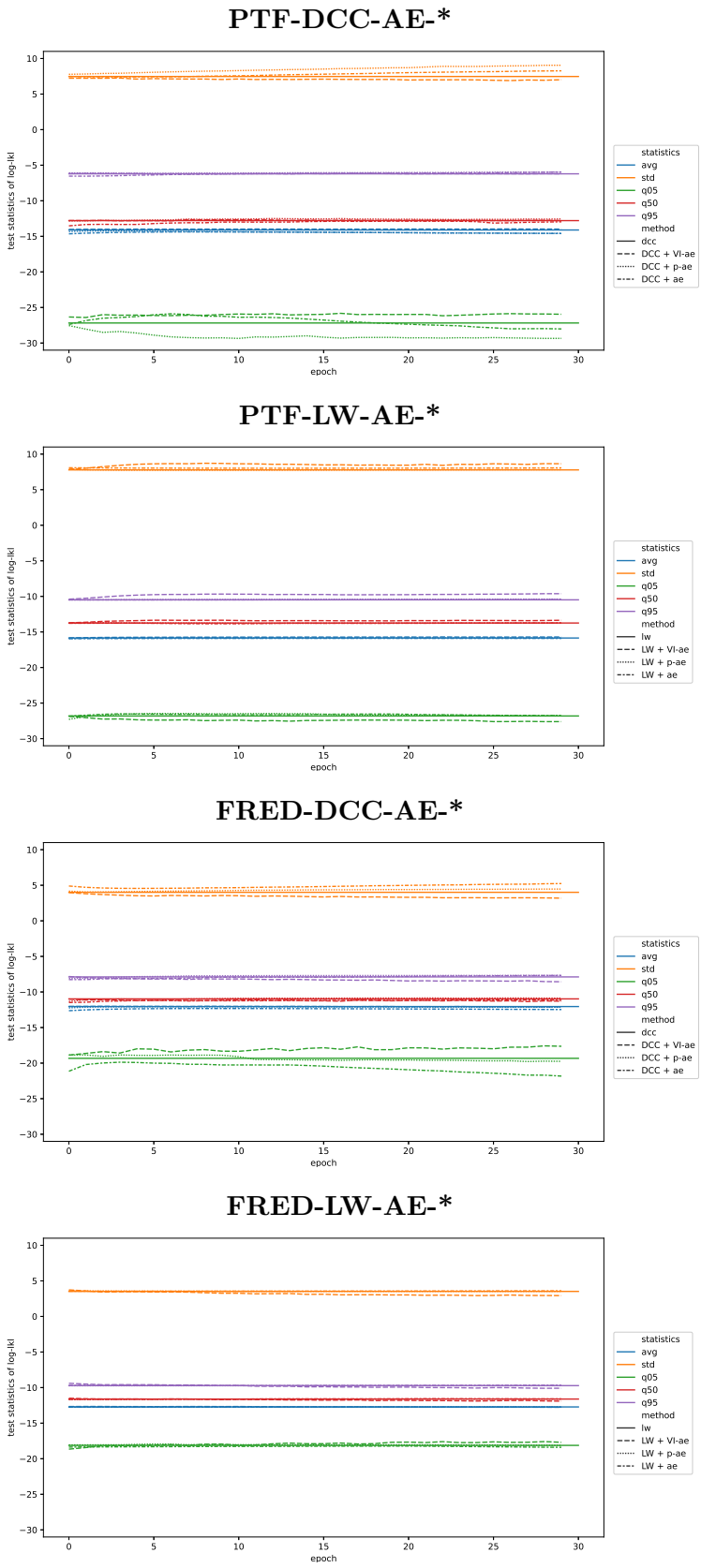

FIGURE 4.3: The different behaviour of the AE variants on the four combinations of the dataset and central estimators during the optimising process. The three different variants: **AE**, **p-AE**, **VI-AE** differs for the applied inference scheme: maximum likelihood, MAP or VI.

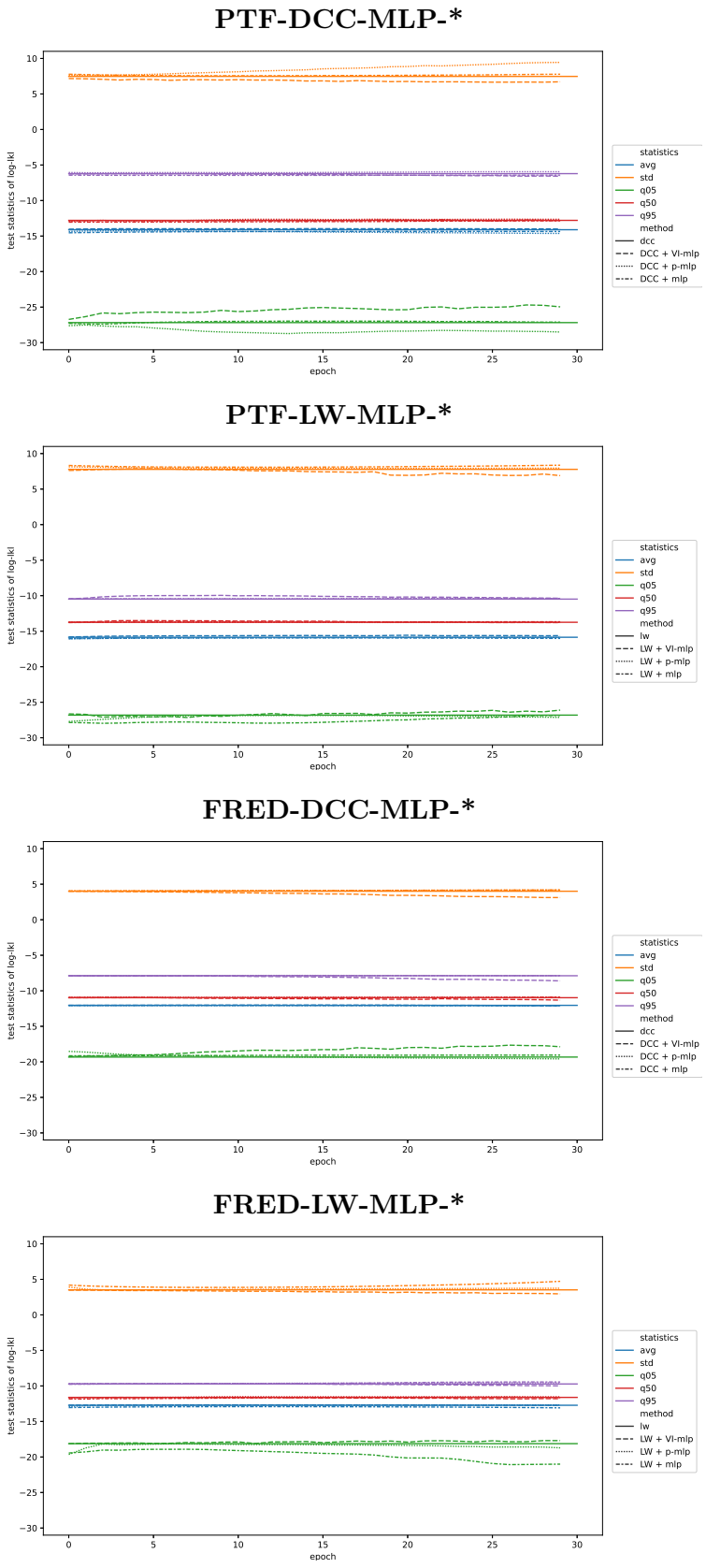

FIGURE 4.4: The different behaviour of the MLP variants on the four combinations of the dataset and central estimators during the optimising process. The three different variants: **MLP**, **p-MLP**, **VI-MLP** differs for the applied inference scheme: maximum likelihood, MAP or VI.

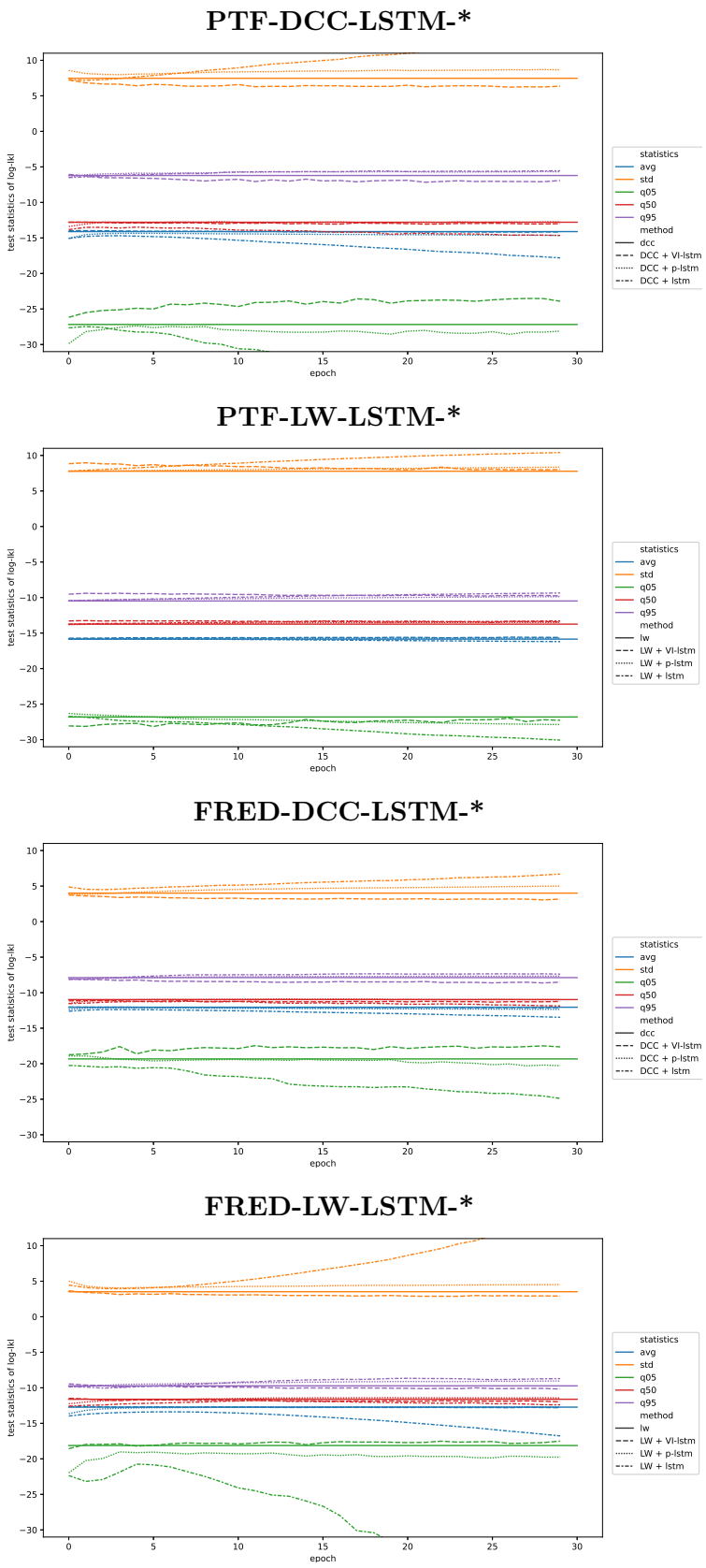

FIGURE 4.5: The different behaviour of the LSTM variants on the four combinations of the dataset and central estimators during the optimising process. The three different variants: **LSTM**, **p-LSTM**, **VI-LSTM** differs for the applied inference scheme: maximum likelihood, MAP or VI.

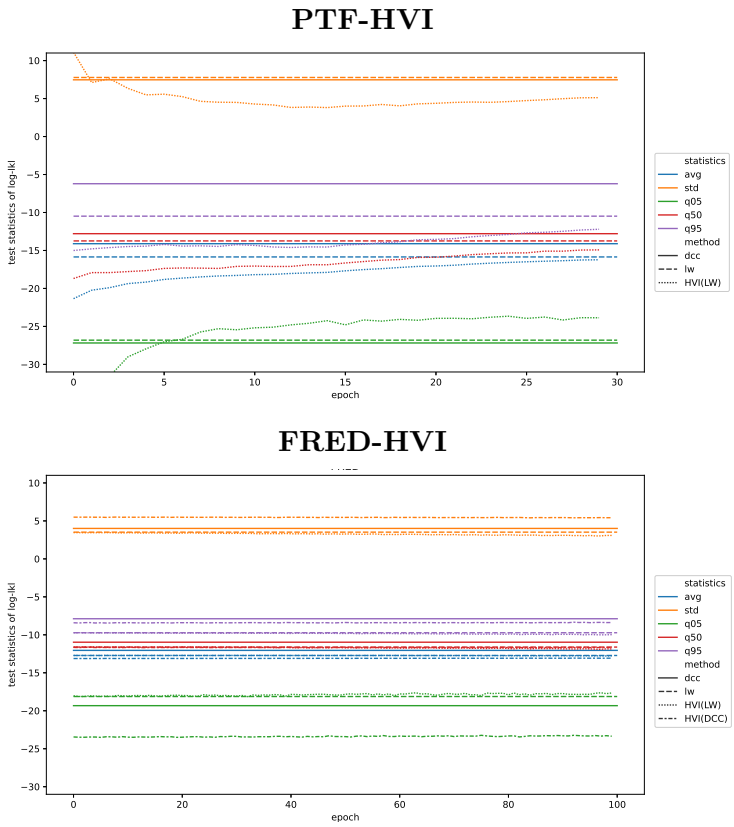

FIGURE 4.6: The average log-likelihood on test observations for **HVI-LW** (**PTF** above, **FRED** below). They both share a flat prior.

First of all, VI emerge as the best inference method, able to infer a reasonable posterior distribution for the parameters, although approximated. On the other hand, single-value estimate, even with penalised maximum-likelihood, often lead to overfitting.

Concerning VI estimate of semi-parametric models, there isn't a single model structure that is better than the others, but the results depend on the context.

In particular, VI tends to overestimate the volatility of the series in the sense of the trace of the covariance matrix wrt the single-value estimators, thus reducing the variability of log-likelihood of future observations significantly.

HVI inference can increase a single-point covariance estimator as Ledoit-Wolf. It is possible to see both the semi-parametric models and the semi-parametric inference approach of HVI as symbiont for point-estimators of mean-covariance states. Attaching a semi-parametric model to a single-value estimate of this latent quantity can result in a tangible forecast improvement. However, a full distribution inference is required with these flexible semi-parametric models and simple point estimates is not recommended.

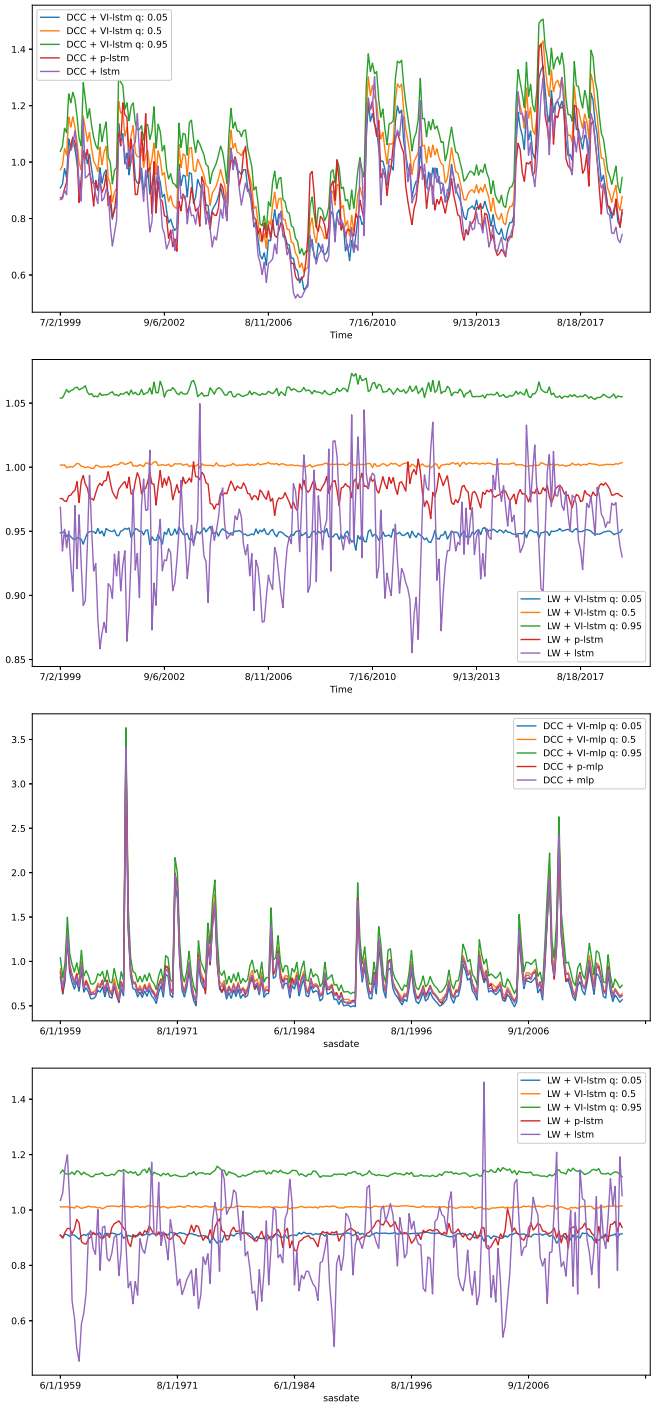

FIGURE 4.7: Different volatility estimate on the four combination of dataset and initialisation point. The $0.05, 0.5, 0.95$ quantiles are plotted for the VI versions of the models.

Appendix A

Appendix

This chapter presents the notations, the definitions and a few brief descriptions of different topics named in the previous chapters.

A.1 Notations

$\mathbb{R}^N$ refers to the N dimensional set of real numbers, equipped with the ordinary operations of vector space, the euclidean norm, and the induced topology.

Subscript letters refers to set indexes. For example, consider the vector

$$y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

then $y_i, i = 1, 2, 3$ refers to the single components.

y_i^t refers to the transpose of a vector

$$y^t = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$$

.

The euclidean norm of a vector is written $\|y\| = \sum_i y_i^2$.

The symbol 0_d define the vector $0 \in \mathbb{R}^d$. Analogously, $1_d = (1, \ldots, 1) \in \mathbb{R}^d$.

$0_d, 1_d$ can also refer to the null matrix and identity matrix, respectively. It should be clear whenever the symbol relates to a vector or a matrix. The exact definition will be explicit whenever confusion could occur.

$\mathcal{M}(N)$ refer to the set of square matrixes of order N. The subset of positive-matrix is $\mathcal{M}^+(N) \subset \mathcal{M}(N)$. The symbol $\mathcal{C}^+(N)$ represents the set of lower triangular matrix with positive diagonal elements. $\mathcal{C}^+(N)$ is the image of the Cholesky decomposition of $\mathcal{M}^+(N)$.

$|V|$ refers to the determinant of a square matrix V.

Taken a random objects x and a probability distribution D, $x \sim D$ claims that x is distributed as D. $p_D(a)$ refers to the probability value $p_D(a) = P(x = a), x \sim D$. Whenever D is a continuous distribution, the same expression represents the probability density.

The symbol $\mathbb{E}[x]$ is the expected value of x. A subscript like $\mathbb{E}_D[F(\cdot)]$ makes explicit the distribution D of the silent random variable $\cdot$.

The probability of any random object x conditional to an algebra of events Σ is written $p(x|\Sigma)$. $p(x|y)$ is the short notation for $p(x|\Sigma(y))$, that is the conditional probability wrt the event algebra $\Sigma(y)$ generated by the random object y.

The symbol $\#(X)$ points to the cardinality of any finite set X.

A.2 Known distributions

The paper cites several probability distributions, follows a list of their definition.

A.2.1 Gaussian distribution

Probability distribution over $\mathbb{R}^N$, with parameters $\mu \in \mathbb{R}^N, V \in \mathcal{M}^+(N)$. A vector X is Gaussian distributed, written

$$X \sim \mathcal{N}_{\mu,V}$$

if its density is

$$p(X) = \frac{1}{2}[\pi|V|]^{-N/2} \exp\left\{-\frac{1}{2}(X-\mu)^t V^{-1}(X-\mu)\right\}$$

It holds

$$\mathrm{E}(X) = \mu,\ Cov(X) = V$$

A.2.2 Wishart distribution

Wishart is a probability distribution over the set of positive-definite matrix of order N, $\mathcal{M}^+(N)$, with parameters $\nu \in \mathbb{R}, W \in \mathcal{M}^+(N)$. A matrix V is Wishart distributed, written

$$V \sim \mathbf{W}_{\nu,W}$$

if its density is

$$p(V) = c^{-1}|W|^{(\nu-N-1)/2} \exp\left\{-\frac{1}{2}\text{Trace}(W^{-1}V)\right\}, \quad \nu > N-1$$

with

$$c = 2^{(\nu N)/2}|W|^{\nu/2}\Gamma_N(\nu/2)$$

It holds

$$\text{E}(V) = \nu W$$

A.2.3 Normal-Wishart distribution

Normal-Wishart is a probability distribution over pairs m, V of vectors $m \in \mathbb{R}^N$ and positive-definite matrix of order $V \in \mathcal{M}^+(N)$. The distribution depend on parameters $k \in \mathbb{R}, \mu \in \mathbb{R}^N, \nu \in \mathbb{R}, W \in \mathcal{M}^+(N)$.

A pair m, V has such distribution, written

$$m, V \sim \mathbf{NW}_{k,\mu,\nu,W}$$

if $V \sim \mathbf{W}_{\nu,W}$ and $m|V \sim \mathcal{N}_{\mu,V}$.

A.2.4 Inverse-Wishart distribution

Inverse-Wishart is a probability distribution over the set of positive-definite matrix of order N, $\mathcal{M}^+(N)$, with parameters $\nu \in \mathbb{R}, W \in \mathcal{M}^+(N)$. A matrix V is Inverse-Wishart distributed, written

$$V \sim \mathbf{IW}_{\nu,W}$$

if $V^{-1} \sim \mathbf{W}_{\nu,W^{-1}}$

A.2.5 Normal Inverse-Wishart distribution

Probability distribution over pairs m, V of vectors $m \in \mathbb{R}^N$ and positive definite matrix of order $V \in \mathcal{M}^+(N)$, with parameters $k \in \mathbb{R}, \mu \in \mathbb{R}^N, \nu \in \mathbb{R}, W \in \mathcal{M}^+(N)$.

A pair m, V has such distribution, written

$$m, V \ \sim \ \mathbf{NIW}_{k,\mu,\nu,W}$$

if $V \sim \mathbf{IW}_{\nu,W}$ and $m|V \sim \mathcal{N}_{\mu,V}$.

A.3 Common functions

s define the softplus function $s(x) = \log(\exp(x) + 1)$. s is invertible and differentiable, with derivative

$$\frac{\partial s(x)}{\partial x} = \frac{\exp(x)}{1 + \exp(x)}$$

also known as sigmoid function.

selu represents the common self-normalizing activation function for neural networks. It is defined by:

$$selu(x) = \begin{cases} \alpha x & \forall x > 0 \\ \beta\alpha\left[\exp(x) - 1\right] & \forall x \leq 0 \\ \alpha = 1.67326 & \\ \beta = 1.05070 & \end{cases} \tag{A.1}$$

A.4 Set reparametrizations

This section defines the different reparametrizations used in this paper. A reparametrization of a set C is a bijective and differentiable function that maps $\mathbb{R}^d$ to C.

A.4.1 Set of positive definite matrix

This reparametrization is obtained combining two functions, the first (m_1) maps $\mathbb{R}^d$ in $\mathcal{C}^+(N)$, and the second assigns $m_2 : C \in \mathcal{C}^+(N) \to C^T C \in \mathcal{M}^+(N)$.

m_1 takes a vector with real components and places each of them in the positions of a lower triangular matrix c_1. After that, it applies the s transformation to the elements on the diagonal. m_1 is then a bijective and differentiable reparametrization.

m_2 takes a lower triangular matrix as input and multiply it for its transpose, to obtain the output. It is naturally differentiable and is bijective on the restricted domain $\mathcal{C}^+(N)$.

The composition $m_2 \cdot m_1$ is a reparametrization of $\mathcal{M}^+(N)$.

The dimension d required of the input must be $N(N+1)/2$ to account for the elements of the lower diagonal matrix.

This paper extensively uses a combined reparametrization of $\mu \in \mathbb{R}^n, V \in \mathcal{M}^+(n)$. In all these cases, it is enough to consider the cartesian product of the identity function (for μ) and the above-mentioned reparametrization (for V).

A.4.2 DCC parameters reparametrization

DCC parameters are:

1. The GARCH parameters of the univariate models for diagonal elements $\omega_i, \alpha_i, \beta_i, \forall i \in 1, \ldots, N$.
2. a, b parameters, controlling the updating mechanism of the correlations.

The reparametrization must map every point of $x \in \mathbb{R}^{3N+2}$ to a single set of parameters value.

This reparametrization function depends on hyperparameters $\omega_{0,i}, \alpha_{0,i}, \beta_{0,i}, a_0, b_0$ that determine its center (the image of 0_{3N+2} in the parameters space) and the scale hyperparameters $\omega_{s,i}, \alpha_{s,i}, \beta_{s,i}, a_s, b_s$ (the scale of the gradient of the reparametrization).

Every component x_k is first associated to a single parameter, for example x_{ω_i} points to the component that refers to the ω parameter of the i^{th} series. With this notation, the reparametrization can be written as:

1. $\omega_i = s\left(x_{\omega_i} s^{-1}(1/\omega_{s,i})\right) \omega_{s,i}\omega_{0,i}$
2. $\alpha_i = s\left(x_{\alpha_i} s^{-1}(1/\alpha_{s,i})\right) \alpha_{s,i}\alpha_{0,i}$
3. $\beta_i = s\left(x_{\beta_i} s^{-1}(1/\beta_{s,i})\right) \beta_{s,i}\beta_{0,i}$
4. $a = \tanh\left(x_a a_s + \tanh^{-1}(a_0 2 - 1)\right)/2 + 1/2$
5. $b = \tanh\left(x_b b_s + \tanh^{-1}(b_0 2 - 1)\right)/2 + 1/2$

where s is the softplus function.

In particular, the reparametrization covers the domains $\alpha_i, \beta_i, \omega_i, a, b \in (0, 1)$. Usually, ω_i is not constrained within the unitary interval. Anyway, it rarely takes values greater than 1. In such cases, it is straightforward to change the function to extend the domain linearly.

A.5 Neural Networks

The term Neural networks usually describe a comprehensive family of both learning methods inspired by structures of neurons.

A lot of research has been done on these methods, leading to a vast set of variants and theoretical results. A beautiful description with their peculiarity, pitfalls, and common variants is available in [2].

This paper is not focused on an in-depth discussion of the topic but uses them extensively as flexible parametric functions.

This section provides a brief recall of the underlying principles and introduces the names and notations for the kind of networks mentioned in the other chapters.

All neural networks share the characteristics of representing a parametric function:

$$N_\theta : x \in \mathbb{R}_0^d \xrightarrow{\theta} N(x) \in \mathbb{R}^{d_l} \tag{A.2}$$

as a sequence of l layers that alternate affine transformations ($A_i, i = 0, \ldots, l$, called projections) to non-linear ones ($s_i, i = 0, \ldots, l$, called activation functions). The result is:

$$\begin{gathered} N_\theta(x) = s_l(A_l(\ldots(s_1(A_1(x))))) \\ A_i : x \in \mathbb{R}^{d_{i-1}} \to x' \in \mathbb{R}^{d_i} \end{gathered} \tag{A.3}$$

$\theta \in \Omega$ corresponds to the collection of the internal parameters of all the projections and activation steps. It identify the function itself, although different θ could potentially represent the same function.

The non-linear activation steps are used to increase the representation power with composition; without them, composing directly two affine transformations will only result in another affine transformation.

Historically, every intermediate computation $x_i = s_i(A_i(\ldots s_1(A_1(x))))$ is called hidden layer, and every component of a layer is called neuron. This naming comes from the biological inspiration of networks of neurons. From a mathematical point of view, every

layer is a point in a real vector space. The transformation used to compute it can be seen as a parametric transformation from its input space to its output. The term 'number of neurons' sometimes refers to the dimension of an internal layer.

How to select a θ with given characteristics? If the goal can be written as an optimisation problem:

$$\hat{\theta} = \arg\max_{\theta} F(N_\theta(x)) \tag{A.4}$$

gradient descent can be applied through back-propagation to rapidly estimate $\nabla_\theta F(N_\theta(x))$, starting from a random initial θ_0. See A.5.6 for extra details.

Usually, F depends on data and incorporates the context information of the problem at hand.

Due to the high dimension of θ, randomising the evaluation of F gives more stable optimisation. For example, the most common randomisation procedure uses batches of data, assuming F depends on them, to speed up the process and enable parallel computation. This variant is called stochastic gradient descent (SGD) and is the dominant paradigm to learn θ.

Which kind of functions is possible to obtain alternating projection and activation? Surprisingly enough, $A_1 \circ s_0 A_0$ is sufficient to approximate every function up to a fixed precision, if $d_l = 1$ and d_1 is large enough. The proof for a specific activation function dates back to decades ago [29] and have been then generalised to other activation mechanisms. In practice, the theorem state that with 2 hidden layer and enough neurons, the network can reproduce any function arbitrarily well.

Such a result lead to great popularity in networks, particularly during the last part of the previous century. The popularity faded, until the empirical discovery that deeper networks can better learn elaborated patterns. This peculiarity led to the birth of the term deep learning [2].

A recent result [43] showed that, also theoretically, even with a limited dimension for the internal layers ($d_i \geq d_{-1} + 4, \forall i \in [0, l]$) a network is still able to reproduce any function if l is big enough and $d_l = 1$. This second result state that with at least an internal dimension equal to $d_0 + 4$ for every layer, it is possible to reproduce any function up to a fixed error (with enough layers).

In the literature, many different networks structures have been proposed.

The following sections describe the specific networks used in this paper.

Some of them also introduce regularisation terms, that are functional term S added to the score F, aiming to improve the learning process.

A.5.1 Multilinear projector (MLP)

MLP is the simplest network and is the building block of many other variants.

It is just a sequence of affine layers and non-linear activation, ending with an affine one. The ending choice is only to obtain a global function with codomain that covers a.e. the entire $\mathbb{R}^d$ (at least for some values of θ).

The choice of the activation function is still debated [44]. This paper uses the self-normalising *selu* function [27]. The reasons behind this choice are its stability and its popularity, not only its self-normalising property.

MLP adds no additional term to the score function F.

In this paper, the symbol $MLP_{l,d}$ will refer to:

$$\begin{gathered} MLP_{l,d}(x) = (A_l(\dots(s_1(A_1(x))))) \\ A_i : x \in \mathbb{R}^{d_{i-1}} \to x' \in \mathbb{R}^{d_i} \\ d_l = d \end{gathered} \tag{A.5}$$

Sometimes in literature the term **MLP** is used for networks with only 2 layers, that here correspond to $MLP_{2,.}$.

A.5.2 Autoencoder

An Autoencoder is "a neural network that is trained to attempt to copy its input to its output" [2]. The task of reconstructing the input can appear naive. Still, when applied to a network with an internal layer of dimension lower then the input itself, this task becomes analogous to compress the information in the input to a smaller space. This characteristic made autoencoders successful in learning data representations.

The idea behind it is to use an MLP (called encoder E) to transform the input in a vector of smaller dimension. Then, another MLP (called decoder D) tries to reconstruct the original input. Connecting another MLP to the encoder output (called 'encoded' variable) allows making predictions.

In this paper, the symbol $AE_{l,e,l',d}$ will refer to the set of functions:

$$\begin{gathered} E(x) = MLP_{l,e}(x) \\ D(E(x)) = MLP_{l,d_{-1}}(E(x)) \\ S(x) = \|x - D(E(x))\| \end{gathered} \tag{A.6}$$

Here the term S acts is linearly added to F and act as a regularizer. Such term pushes the encoder to preserve the information in the input. Otherwise, the decoder will not be able to reconstruct it.

Often, another MLP is inserted as a forecaster, taking $E(x)$ as input and projecting to the required output space.

A.5.2.1 Variational Autoencoder

Variational autoencoders are a variant of autoencoders that injects uncertainty in $E(x)$. This random noise forces a more stable encoding of the information in x, because fragile representations are heavily affected by randomisation.

They have been first introduced in [4] and extensively discussed in [45].

The most common way to introduce uncertainity is to use the reparametrization trick, as described in 1.4.4. Specifically, a variational autoencoder $VAE_{l,d,l'}$ is defined by:

$$\begin{aligned} E_m(x) &= MLP_{l,e}(x) \\ E_\sigma(x) &= MLP_{l,e}(x) \\ \epsilon \sim \mathcal{N}_{0_e,1_e} E(x) &= E_m(x) + E_\sigma(x)\epsilon \\ D(E(x)) &= MLP_{l,d_0}(E(x)) \\ S(x) &= \|x - D(E(x))\| \end{aligned} \tag{A.7}$$

In practice, random gaussian samples are used to obtain a stochastic $E(x)$ as a differentiable transformation of $E_m(x), E_\sigma(x)$. This smoothness does not break backpropagation allowing the usual optimisation with SGD.

A.5.3 Recurrent networks

Recurrent networks are network working with sequential data, as time-series or sentences. Their peculiarity is to take as input also their own output computed at the previous element of the sequence. In this way, these networks can reproduce an internal state that varies through time in a similar way to autoregressive models.

The input of this kind of networks is an ordered batch $x_0, \ldots, x_T$ instead of a single x. The general output is:

$$\begin{aligned} R(x_{-1}) &= R_0 \\ R(x_t) &= MLP_{l,d}\left(x_t \bigoplus R(x_{t-1})\right) \quad t = 0, \ldots, T \end{aligned} \tag{A.8}$$

The initial value R_0 is usually treated as a common parameters: initialised at random and learned through backpropagation.

This general scheme could suffer from gradient vanishing or exploding if T is too large. Unfortunately, a small T could not be able to capture distant time dependencies. Different modifications attempt to solve this issue, like the one in the next section.

A.5.3.1 Long-Short-Term-Memory networks (LSTM)

LSTM networks are recurrent networks with a particular internal structure, called "forgetting gate" that solve the issue of vanishing gradient that could affect recurrent networks learning. [15] proposed them for the first time.

The idea is to use an internal gate G that solve the gradient issue:

$$\begin{aligned} I(x_{-1}) &= R_0 \\ I(x_t) &= G\left[MLP_{l,d}(x_t \bigoplus R(x_{t-1})\right] \qquad t = 0, \dots, T \\ LSTM_{d,G}(x_t) &= MLP_{l,d}(x_t \bigoplus I(x_{t-1}) \quad t = 0, \dots, T \end{aligned} \tag{A.9}$$

Different versions of G exists. In this paper, $LSTM_{d,G}$ refers to the version implemented in [28].

A.5.4 Bayesian networks

Most neural networks applications select a single value for the parameters θ. Inferring a whole distribution, instead, can increment the generalisation power and reduce overfitting, as shown in [3].

Deriving an entire distribution on a high dimensional set as the network parameters can be difficult. Asymptotically exact methods like MCMC are rarely used. Simple methods as dropout [46] are popular. The reparametrisation trick is obtaining much attention since its introduction [1].

In this paper, the term Bayesian network or Variational network will refer to any network with non-singular parameters distribution.

A.5.5 Generative networks

Networks are semi-parametric functions. Applying them to synthetic numbers allows generating pseudo-random numbers with a different distribution. Using SGD, it is possible to adapt the distribution of these samples to maximise a given score.

In particular, they represent the state-of-the-art in producing artificial images and videos. See [2] for an introduction to this topic.

A.5.6 Backpropagation and SGD

This section describes how to select a reasonable value for θ that maximises $S(N)$.

The value of θ is initialised randomly as θ_0 and modified during several iterations until a stopping criterion is reached.

At every iteration i, the procedure randomly selects a batch of data $X_b \subset X$ and compute the score on it. Then it compute, using automatic differentiation and backpropagation, the derivative of this score wrt the parameters:

$$g_i = \frac{\partial}{\partial \theta_i} \sum_{x \in X_b} S(N(x), x)$$

θ_i is modified proportionally to g_i:

$$\theta_i \leftarrow \theta_i + \nu g_i \tag{A.10}$$

The proportionality coefficient ν can be fixed at priori, chosen through trial and error, or selected adaptively at every step.

The literature contains many different versions of SGD. In this paper, all the results use the specific implementation of the *Adam optimiser* implemented in [28].

A.6 Empirical prior on mean-covariance states

This section defines an empirical prior over mean-covariance states.

The prior is empirical in the sense that it starts from data and returns a distribution over the set of states vector μ_t, V_t.

Consider multivariate time-series $Y = \{y_t \in \mathbb{R}^n, \forall t = 0, \dots, T\}$.

Selecting a point estimator of μ_t, V_t, as DCC, it is possible to map $Y \to \mu_{0,t}, V_{0,t}$.

The above states can then be mapped to vectors z_t using A.4.1.

The prior assumes a random-walk process on z_t, similar to what is done in [6] or [11]. This imply to choose a distribution for the starting point $p(z_0)$ and a transition kernel $p(z_t|z_{t-1})$.

$p(z_0)$ is again a multivariate Gaussian distribution, with mean and covariance matrix equal to the sample mean of z_t and the population covariance matrix of z_t. Other estimators for this Gaussian distribution could be applied. For example, the Python package introduced in 3 uses the Ledoit-Wolf estimator for the covariance estimate.

In a similar way, the Gaussian transition kernel $p(z_t|z_{t-1})$ is obtained applying any estimator of the population covariance matrix to the finite differences $dz_t = z_t - z_{t-1}$. Again, Ledoit-Wolf correction is used in 3 and strongly advised, due to the dimension of z_t equal to $n + n(n+1)/2$.

The consistency of the population means and population covariance estimator implies the consistency of the random walk estimate.

Whenever such estimates would bring to singular covariance matrixes, a correction term proportional to the identity matrix is added, in the implementation of 3.

Bibliography

[1] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518): 859–877, 2017.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[3] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[6] Gary Koop and Dimitris Korobilis. *Bayesian multivariate time series methods for empirical macroeconomics*. Now Publishers Inc, 2010.

[7] Joshua Chan, Gary Koop, Dale J Poirier, and Justin L Tobias. *Bayesian Econometric Methods*, volume 7. Cambridge University Press, 2019.

[8] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[10] Robert Engle. Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. *Journal of Business & Economic Statistics*, 20(3):339–350, 2002.

[11] Fabio Canova and Matteo Ciccarelli. Estimating multicountry var models. *International economic review*, 50(3):929–959, 2009.

[12] Massimiliano Caporin and Michael McAleer. Ten things you should know about the dynamic conditional correlation representation. *Econometrics*, 1(1):115–126, 2013.

[13] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

[14] CF Jeff Wu. On the convergence properties of the em algorithm. *The Annals of statistics*, pages 95–103, 1983.

[15] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[16] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[17] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[18] R. E. Turner and M. Sahani. Two problems with variational expectation maximisation for time-series models. In D. Barber, T. Cemgil, and S. Chiappa, editors, *Bayesian Time series models*, chapter 5, pages 109–130. Cambridge University Press, 2011.

[19] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822. PMLR, 2014.

[20] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[21] Gareth O Roberts, Jeffrey S Rosenthal, et al. General state space markov chains and mcmc algorithms. *Probability surveys*, 1:20–71, 2004.

[22] Christian P Robert, Víctor Elvira, Nick Tawn, and Changye Wu. Accelerating mcmc algorithms. *Wiley Interdisciplinary Reviews: Computational Statistics*, 10 (5):e1435, 2018.

[23] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[24] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[25] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[26] Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019.

[27] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[28] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[29] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[30] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[31] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.

[32] Christa Cuchiero, Martin Larsson, and Josef Teichmann. Deep neural networks, generic universal interpolation, and controlled odes. *SIAM Journal on Mathematics of Data Science*, 2(3):901–919, 2020.

[33] Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. Coda: convergence diagnosis and output analysis for mcmc. *R news*, 6(1):7–11, 2006.

[34] Francisco JR Ruiz and Michalis K Titsias. A contrastive divergence for combining variational inference and mcmc. *arXiv preprint arXiv:1905.04062*, 2019.

[35] Olivier Ledoit, Michael Wolf, et al. Nonlinear shrinkage estimation of large-dimensional covariance matrices. *The Annals of Statistics*, 40(2):1024–1060, 2012.

[36] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

[37] Yan Zilbering, Colleen M Jaconetti, and Francis M Kinniry Jr. Best practices for portfolio rebalancing. *Valley Forge, Pa.: The Vanguard Group. Vanguard Research PO Box*, 2600:19482–2600, 2015.

[38] Daniel Kuhn and David G Luenberger. Analysis of the rebalancing frequency in log-optimal portfolio selection. *Quantitative Finance*, 10(2):221–234, 2010.

[39] Michael W McCracken and Serena Ng. Fred-md: A monthly database for macroeconomic research. *Journal of Business & Economic Statistics*, 34(4):574–589, 2016.

[40] Gary Koop. Bayesian methods for empirical macroeconomics with big data. *Review of Economic Analysis*, 9(1):33–56, 2017.

[41] Gary Koop, Dale J Poirier, and Justin L Tobias. *Bayesian econometric methods*. Cambridge University Press, 2007.

[42] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep optimal stopping. *Journal of Machine Learning Research*, 20:74, 2019.

[43] Yingzhen Li, Richard E Turner, and Qiang Liu. Approximate inference with amortised mcmc. *arXiv preprint arXiv:1702.08343*, 2017.

[44] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[45] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.