TESIS DOCTORAL

Contribuciones a la Inferencia Bayesiana Aproximada para Aprendizaje Automático

(Contributions to Approximate Bayesian Inference for Machine Learning)

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Simón Rodríguez Santana

Directores

Daniel Hernández Lobato
David Gómez-Ullate Oteiza

Madrid

UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS

TESIS DOCTORAL

Contribuciones a la Inferencia Bayesiana Aproximada para Aprendizaje Automático
(Contributions to Approximate Bayesian Inference for Machine Learning)

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Simón Rodríguez Santana

DIRECTORES

Daniel Hernández Lobato
David Gómez-Ullate Oteiza

Programa de doctorado en Ingeniería Matemática, Estadística e Investigación Operativa por la Universidad Complutense de Madrid y la Universidad Politécnica de Madrid

FACULTAD DE CIENCIAS MATEMÁTICAS (UCM)

Contribuciones a la Inferencia Bayesiana Aproximada para Aprendizaje Automático

(Contributions to Approximate Bayesian Inference for Machine Learning)

Tesis Doctoral

Simón Rodríguez Santana

DIRECTORES

Daniel Hernández Lobato

David Gómez-Ullate

Año 2021

Agradecimientos

El proceso de escritura de esta tesis ha sido complicado, con un abanico completo de momentos que van desde los más agradables a los que no lo son tanto. Por ello, me parece cuanto menos necesario agradecer a todos aquellos que en algún momento han tenido algo que ver con esto y conmigo, porque aunque la firma sea de una persona, estoy firmemente convencido de que, de no ser por la tremenda suerte de estar rodeado de gente tan excepcional, habría terminado por quién sabe qué derroteros. Pido disculpas anticipadas por no poder hacer verdadera justicia a este hecho y a toda la gente que, sabiéndolo o no, tanto me ha ayudado. Es harto probable que me deje nombres particulares atrás, pero ojalá me perdonen aquellos que queden en ese limbo. Gracias a todos, de corazón.

Antes de nada, quería agradecer a Daniel Hernández Lobato su labor fundamental para la realización de esta tesis. Daniel, gracias por acogerme dentro de tu investigación y por haberme enseñado tanto, aunque sé que me queda mucho que aprender de ti aún. Estoy absolutamente convencido de que de otra manera este trabajo no habría sido posible, y no podré agradecerte lo suficiente todo lo que has hecho por mi durante estos años. Agradezco además las conversaciones con los otros muchos investigadores que he tenido la suerte de conocer estos años. Dentro de ellos, quería hacer especial mención a David Gómez-Ullate, David Ríos y al resto de integrantes del grupo SPOR del ICMAT, así como a la gente del grupo de PML en Aalto. Aquí quiero resaltar en concreto a Aki Vehtari el facilitarme realizar una estancia con su grupo de investigación en la Universidad de Aalto, durante la cual traté de aprender y aprovechar todo lo posible a pesar de las dificultades de la pandemia. Gracias a todos por ayudarme a crecer dentro de la investigación.

Mi familia es el mayor privilegio del que dispongo. Siendo consciente de que es bastante extensa, y por temor a dejarme a nadie atrás, espero que sepan verse reflejados en algo tan escueto a pesar de merecer mucho más. Querría en primer lugar agradecer a Pino y Cuco por todo el cariño que me han dado durante mi vida. Siempre me han tratado como un hijo, y todo el calor con el que siempre me reciben al volver me hace saber que estoy en casa. A mi abuela Angelina, por sus consejos e historias, y a mi abuelo Jose, por lo que nos dejaste. A mi tío José, por su trabajo incansable. A mis primos, Óliver, Ángela, Álvaro, Eduardo y Claudia, por siempre estar ahí, durante toda mi vida (o durante toda la suya). A Jesús, Juan, Luis, Rita, Manolo, Victoria, Inmaculada, Ángeles, Loli, Delia, Elena, Gabriel y Antonio, por las discusiones, el humor, los recuerdos, los abrazos y el cariño. A mi abuela Julia, por aquellos plátanos, y a mi abuelo Julián. A mis primos Ancor, Gara, Alba, Saúl, Aday, Ismael, Marimar, Davinia, María, Abraham y Rayco, por el cariño, las enseñanzas y las fechorías, pero sobre todo por todo el tiempo pasado juntos. Por último, por extensión, a Flor y a Jesús, por estar siempre ahí y formar parte de la familia antes de que tuviera uso de razón. Gracias a todos.

Quiero agradecer a toda esa gente que, tanto en Gran Canaria como en Madrid, me han ayudado a sentirme bienvenido dondequiera que estuviese. En concreto, querría mencionar a Sara, Laura, Arminda, Héctor, Jonay y a Alma, que a pesar de que nos reunamos pocas veces al año, me hacen ver que sigo estando presente con ellos, allá donde estemos y por los caminos en los que nos encontremos. Por otra parte, en Madrid que se han encargado de arroparme y permitirme expandirme como persona de nuevas maneras. Aunque el grupo es sustancialmente más grande, quiero agradecer en particular a Luis, por el cariño y la constancia, así como a Inaxio, Óscar, Cecilia, Jacin, Víctor, Eva y a Paco todos los momentos y la paciencia que han tenido conmigo, así como todo el cariño que han sabido darme y que espero haya podido devolverles. Siempre les estaré eternamente en deuda por todo lo aprendido con ellos, y sólo puedo desear ver lo que nos queda por aprender en adelante. Además de esto, querría mostrar en especial mi agradecimiento a Héctor. Quiero expresarte lo importante que has sido para mi desde que llegué a Madrid y todo lo que me has ayudado a formarme como persona, siendo un apoyo que estaba ahí cuando más lo he necesitado. Aunque estemos lejos, siempre tendrás conmigo un sitio.

He coincidido con gente maravillosa durante la carrera y etapas posteriores, en la convivencia u otras experiencias compartidas. En esta línea, quería mencionar en particular a Álvaro, al que agradezco crecer tanto durante conversaciones sobre cualquier tema en mente. A Fran, por mantener una amistad y camaradería que he atesorado durante estos años. A Julen, por su cariño, entusiasmo y curiosidad, a pesar de que desde hace tiempo estemos algo alejados. A Marta, por su cercanía, su cariño y su efusividad, sin importar dónde nos encontremos. A Beatriz, por su empatía, su comprensión y por tener la valentía de sumarse a nosotros. A Javi, por el buen humor y toda la compañía que me ha brindado desde que llegó. A Christian, por su ternura, su sinceridad y su pasión. Agradezco haber aprendido tanto de ti, no solo a raíz de tonterías que compartimos, sino también como referencia que emular para crecer más como persona. A Robert, por ayudarme a expandirme hacia gustos e intereses insospechados, por la bondad con la que tratas a todos los que te rodeamos, aunque estemos lejos, y por todo los que nos queda por hacer juntos. Gracias a todos ustedes. A todos aquellos que aún dejo en el tintero, ruego me disculpen, y sepan que tendría miles de cosas que agradecerles igualmente.

Por un motivo u otro, hay ciertas personas que durante estos años han tenido especial relevancia a la hora de conducirme a realizar esta tesis. En primer lugar, quería agradecer a Roi toda su amistad y su agudeza durante estos años. Gracias a nuestras discusiones y a nuestros encuentros y desencuentros (muchos y variados, he de añadir), creo que no es exagerar decir que he llegado a estas líneas de investigación. Te estaré siempre agradecido.

Nadir, durante estos últimos años te has convertido en un miembro más de mi familia. Pocas veces he conocido a una persona más leal y bondadosa, me enorgullezco de haber crecido a través de mis conversaciones contigo y de contarte como un amigo.

Has sido fundamental durante algunos de los momentos más crudos de estos años, y espero poder devolverte el favor. Siempre tienes un hogar conmigo cuando lo necesites.

Alex, gracias por tanto cariño y amistad durante todos estos años. Desde la carrera hasta el día de hoy has sido una pieza fundamental de mi vida. Gracias por todas las discusiones, comidas, lecturas y ratos compartidos; soy mejor persona gracias a ello. A pesar de la distancia y las circunstancias, siempre estás conmigo cuando más te necesito, y espero que sepas que puedes contar conmigo para lo que sea. Querría además hacer extensivo este agradecimiento a toda tu familia. Gracias por acogerme siempre como si se tratase de mi segunda familia en Madrid, es un privilegio y lo agradezco enormemente.

Por otra parte, estos últimos años de tesis han estado marcados por la irrupción de una de las personas más especiales con las que cuento en mi vida. Julia, me faltan las palabras para agradecer el amor, el cariño y la devoción que te profeso. Tu amor me hace mejor persona, aprendo de ti cada día que estamos juntos, y estoy tremendamente orgulloso de ti, de tu talento, de tu inteligencia y de tu forma de ver el mundo. Creo ser poco merecedor de tal regalo, aunque haré lo que esté en mi mano para preservarlo y tratar de pagar con más cariño todo lo que me das. Esto es apenas el comienzo de lo que nos queda por recorrer juntos, y me ilusiona lo que viene. Te quiero. Aprovecharé también para agradecer de corazón a tus padres, Luis y Pilar, y al resto de tu familia, por hacerme sentir como uno más y por ayudarme cuando más falta me hacía.

Finalmente, quería dar las gracias a las personas que han tenido el papel primordial en mi vida a la hora de configurarme como soy, enseñarme a vivir y a enfrentarme a la vida. Tener la posibilidad de escribir esta tesis atestigua la suerte que he tenido desde el primer momento en mi vida. Por fortuna, tuve la enorme suerte de crecer en una familia cariñosa, sana, y que me ofreció las posibilidades necesarias para formame y crecer como persona, fuera lo que fuese eso para mi en cada momento. He gozado de libertad y apoyo incondicional, algo absolutamente privilegiado, y siendo consciente de ello temo que nunca conseguiré devolver todo lo que me ha otorgado ese regalo. Quiero agradecerle en primer lugar a mi hermano Pablo, compañero de vida. Gracias por tu coraje, tu tenacidad y tu inteligencia, porque me haces seguir hacia delante y cuestionarme mis asunciones cuando lo necesito. A mi padre, por todo lo que me ha enseñado y continúa enseñándome a día de hoy. Porque has sabido contagiarnos tu curiosidad, tu humor y por demostrarnos tantísimo amor y apoyo, incondicionalmente. Y por último, a mi madre. No sería nada ni nadie sin ti, sin tu cariño, tu devoción, tu buen humor y tu alegría. Por tus ganas de progresar, por tu ímpetu, por tu amor. No hay nadie que pudiera haberme hecho tan feliz como tu, *aunque fuera de plástico*. Todo el amor del que dispongo proviene de ellos, porque me han enseñado cuanto sé. Les quiero mucho, gracias desde el fondo de mi corazón. Solo puedo darles las gracias y pasarlo a quienes me encuentre en el futuro.

Gracias por todo.

Contents

List of Tables

List of Figures

Notation

AADM	**A**dversarial **$\boldsymbol{\alpha}$**-**D**ivergence **M**inimization
AVB	**A**dversarial **V**ariational **B**ayes
BNN	**B**ayesian **N**eural **N**etwork
BB-$\boldsymbol{\alpha}$	**B**lack-**B**ox **$\boldsymbol{\alpha}$**-divergence minimization
CRPS	**C**ontinuous **R**anked **P**robability **S**core
DNN	**D**eep **N**eural **N**etwork
ELBO	**E**vidence **L**ower **Bo**und
EP	**E**xpectation **P**ropagation
FBNN	**F**unctional variational **B**ayesian **N**eural **N**etwork
GP	**G**aussian **P**rocess
HMC	**H**amilton **M**onte **C**arlo
IP	**I**mplicit **P**rocess
KL	**K**ullback-**L**eibler divergence
KM	**K**ernel **M**ethod
LL	**L**og-**L**ikelihood
MAE	**M**ean **A**bsolute **E**rror
MC	**M**onte **C**arlo
MCMC	**M**arkov **C**hain **M**onte **C**arlo
ML	**M**achine **L**earning
RMSE	**R**oot **M**ean **S**quared **E**rror
NN	**N**eural **N**etwork
NS	**N**eural **S**ampler
PEP	**P**ower **E**xpectation **P**ropagation
SGD	**S**tochastic **G**radient **D**escent
SGP	**S**parse **G**aussian **P**rocess
SIP	**S**parse **I**mplicit **P**rocess
SP	**S**tochastic **P**rocess
VI	**V**ariational **I**nference
VIP	**V**ariational **I**mplicit **P**rocess

Abstract

Machine learning (ML) methods can learn from data and then be used for making predictions on new data instances. However, some of the most popular ML methods cannot provide information about the uncertainty of their predictions, which may be crucial in many applications. The Bayesian framework for ML introduces a natural approach to formulate many ML methods, and it also has the advantage of easily incorporating and reflecting different sources of uncertainty in the final predictive distribution. These sources include uncertainty related to, for example, the data, the model chosen, and its parameters. Moreover, they can be automatically balanced and aggregated using information from the observed data. Nevertheless, in spite of this advantage, exact Bayesian inference is intractable in most ML methods, and approximate inference techniques have to be used in practice. In this thesis we propose a collection of methods for approximate inference, with specific applications in some popular approaches in supervised ML.

First, we introduce neural networks (NNs), from their most basic concepts to some of their most popular architectures. Gaussian processes (GPs), a simple but important tool in Bayesian regression, are also reviewed. Sparse GPs are presented as a clever solution to improve GPs' scalability by introducing new parameters: the inducing points. In the second half of the introductory part we also describe Bayesian inference and extend the NN formulation using a Bayesian approach, which results in a NN model capable of outputting a predictive distribution. We will see why Bayesian inference is intractable in most ML approaches, and also describe sampling-based and optimization-based methods for approximate inference. The use of α-divergences is introduced next, leading to a generalization of certain methods for approximate inference. Finally we will extend the GPs to implicit processes (IPs), a more general class of stochastic processes which provide a flexible framework from which we can define numerous models. Although promising, current IP-based ML methods fail to exploit of all their potential due to the limitations of the approximations required in their formulation.

In the second part of this thesis we present our contributions to approximate inference with a specific focus on Bayesian NNs and IPs. First, we introduce a method to carry out approximate inference by minimizing α-divergences with flexible implicit approximate distributions. The resulting method, adversarial α-divergence minimization (AADM), optimizes a more general objective than the one in other approaches such as variational inference or expectation propagation. Because of this, AADM can capture complex patterns in the predictive distribution, which is not restricted to be Gaussian. Moreover, AADM introduces a new parameter that can be tuned to optimize specific metrics of the predictive distribution. We also carry out extensive experiments to show that AADM outperforms previous work on approximate inference for Bayesian NNs. Finally, a second contribution describes another method for approximate inference with IPs. Here, approximate inference is carried out in the function space. This circumvents some of the inherent problems of approximate inference in the parameter space, which can be high-dimensional and present strong dependencies between parameters. Our method, sparse IPs (SIP), is the first general-purpose approach based on IPs that is able of adjusting the prior distribution of latent function while also producing flexible predictive distributions. Moreover, due to the use of approximations based on inducing points, SIP remains efficient and scalable on large datasets with millions of data points. In the experiments carried out, we demonstrate SIP achieves better results than all of the previous methods, while also showing unique new features among IP-based approaches.

Resumen

Los métodos de aprendizaje automático o *machine learning* (ML) son capaces de aprender a partir de datos y producir predicciones para nuevos casos nunca vistos. Sin embargo, algunos de los métodos de ML más usuales son incapaces de informar sobre la incertidumbre de sus predicciones, la cual puede ser crucial en diversas aplicaciones. La perspectiva Bayesiana proporciona un marco natural para ello, otorgando la capacidad de considerar diversas fuentes de incertidumbre en el análisis y reflejarlas en las distribuciones predictivas finales. Esta incertidumbre puede tener diferentes fuentes, como los datos, la selección del modelo y sus parámetros asociados, las cuales pueden ser adecuadamente pesadas y agregadas usando las herramientas Bayesianas. Sin embargo, para la mayoría de métodos de ML, la inferencia Bayesiana exacta es intratable, y para casos prácticos hay que recurrir a aproximaciones de la misma. En esta tesis se proponen nuevos métodos de inferencia aproximada, con aplicaciones concretas para algunos de los métodos más populares en ML.

En primer lugar introduciremos las redes neuronales (NNs), desde sus fundamentos básicos hasta algunas de sus arquitecturas más comunes, así como los procesos Gaussianos (GPs), herramientas importantes empleadas en diversos problemas de aprendizaje. Además, veremos cómo los *sparse* GPs alivian los problemas de escalabilidad de los GPs mediante la introducción de un parámetro nuevo: los *puntos inducidos*. En la segunda mitad de esta introducción describiremos los fundamentos de la inferencia Bayesiana y extenderemos la formulación de las NNs al marco Bayesiano para obtener NNs capaces de producir distribuciones predictivas. Veremos aquí por qué la inferencia Bayesiana es intratable para muchos de los métodos de ML y revisaremos técnicas de aproximación basadas tanto en muestreos como en la optimización de parámetros. Además de esto, veremos las α-divergencias como una generalización de conceptos empleados en ciertos métodos de inferencia aproximada. Finalmente extenderemos la formulación de los GPs a los procesos implícitos (IPs), una clase más general y flexible de procesos estocásticos desde la cual podremos describir múltiples modelos útiles. Aunque prometedores, los métodos actuales de ML basados en IPs no son capaces de explotar todas sus propiedades debido a las limitaciones de las aproximaciones empleadas.

En la segunda parte de la tesis presentaremos nuestras contribuciones al campo de inferencia aproximada, con especial interés para las NNs Bayesianas y los IPs. Primero veremos un método para realizar inferencia aproximada usando α-divergencias con distribuciones aproximadas implícitas. El método resultante, *minimización adversaria de α-divergencias* (AADM), optimiza un objetivo más general que otros anteriores basados en inferencia variacional o *expectation propagation*, lo cual le otorga la capacidad de capturar patrones más complejos de los datos y mostrarlos en su distribución predictiva, la cual ya no estará restringida a ser Gaussiana. AADM incluye un nuevo parámetro que puede emplearse para optimizar diversas métricas en los resultados finales, y a través de numerosos experimentos se muestra que supera el rendimiento de métodos anteriores en el contexto de NNs Bayesianas. Por último, veremos una segunda contribución que hace uso de IPs para inferencia aproximada. Esta emplea optimización en el espacio de funciones, ya que el espacio de parámetros usual padece de problemas intrínsecos por su alta dimensionalidad y las interdependencias entre los mismos. Nuestro método, *sparse* IPs (SIP), es el primer sistema basado en IPs completamente general, capaz de ajustar su modelo de probabilidad *a priori* y de producir distribuciones predictivas flexibles simultáneamente. Además, debido al uso de la aproximación de puntos inducidos, SIP es escalable y eficiente para conjuntos grandes de datos con millones de instancias. En los experimentos SIP demuestra mejor rendimiento que los demás métodos, presentando además nuevas propiedades únicas entre los sistemas basados en IPs.

Chapter 1

Introduction

1.1 Motivation

In modern societies, gathering and managing data is a common practice. One effect of this procedure, while also one of its main consequences, is the exploitation of the resulting datasets for numerous tasks through statistics analysis and pattern recognition techniques, also called *machine learning* (ML). The framework of ML consists on the design and implementation of methods to automatically induce patterns from data, which are assumed regular up to a certain degree (Bishop 2006). Although taking advantage of previous data is not a new concept, the performance achieved by modern techniques in ML has lead to a sharp increase in public interest on this research field, which is also accompanied by a more important role in society due to the ubiquity of ML in many aspects of life.

However, in certain scenarios it may be crucial to quantify the uncertainty in the predictions provided by the ML algorithms to make informed decisions, which is something that many of the currently used methods are not capable of doing (Gal 2016). Therefore, it is essential to develop techniques that can provide this type of information. To this end, the research community has focused on combining ML with Bayesian statistics as a possible solution to this problem, since the Bayesian framework provides many useful tools that can be employed to further improve previous methods. Bayesian statistics relies on encoding information through probability distributions, and through simple manipulations it allows for updating a set of prior beliefs according to data observations in a procedure usually referred to as *Bayesian inference* (MacKay 2003). A probability distribution is the result of this process, where the uncertainty coming both from the data and the choice of model itself is reflected. The main obstacle here is, in most cases, the intractability of some of the calculations involved, which cannot be solved analytically. Therefore, inference is mostly conducted in an approximate sense, *i.e.* the intractable terms are obtained in an approximate manner to complete the inference process.

The quality of the predictions and the associated uncertainties provided by approximate inference methods strongly depend on the properties of the selected approximation method. Usually, there is a trade-off between the simplicity of the approximation and its convenience and expressiveness: simpler models may be easier to fit and formulate, while may provide more restrictive predictive distributions. On the other hand, more complex models may reproduce much more flexible distributions at the expense of a more complicated mathematical formulation, as well as increases in the computational cost. In this thesis we will focus on extending the existing formulation for approximate inference techniques, proposing new models that enhance the versatility, scalability and accuracy of current approaches. We will see that the Bayesian approach to ML allows for the construction of methods whose predictions are more informative and robust, while allowing to incorporate many useful features inherited from Bayesian statistics, from extracting more information out of small datasets to encoding previous information to help form the final predictions. To motivate these topics, we will briefly review separately some of the developments that have lead to this research.

1.1.1 Development of Machine Learning

During the last decades, the usage of ML has become widespread due to its flexibility and performance across many different contexts. It has also proved to be essential in trying to develop advanced solutions to new complex real-world problems, such as autonomous vehicles (Fujiyoshi et al. 2019), sustainable agriculture (Sharma et al. 2020), predictive diagnosis of neurodegenerative diseases (Liu et al. 2020), and cybersecurity (Xin et al. 2018), to name only a few. The versatility and performance needed to tackle these issues have only been achieved recently thanks to advances in modern ML techniques, coupled with an increasing availability of larger computational power. Therefore, it is important to introduce some of the basic concepts that will serve as context in the following chapters.

During the length of this thesis, we will mostly refer to ML in the context of *supervised machine learning* (Murphy 2012). Within this framework we are given a certain training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, with N pairs of observations $\mathbf{x}_i$ and target values y_i. The task here is to construct a model that is able to learn a predictor for y when only the corresponding value of $\mathbf{x}$ is observed. Depending on if the labels take values in a discrete or in a continous set we will refer to those problems as *classification* and *regression*, respectively (Murphy 2012). In a supervised learning set, the model's parameters are fitted so that the estimated targets resemble as closely as possible the real ones. For this, we need to define an objective function that quantifies the similarity of the estimated predictions from the targeted values, and we will correct the values of the parameters in the model to minimize this error rate. However, this is not the only learning method: *unsupervised learning* deals with data without labels (Hinton, Sejnowski, et al. 1999), while *semi-supervised learning* uses data where only a few instances are labeled and tries to use that information to assign labels to a bigger and unlabeled set of data (Chapelle et al. 2009). Finally, *reinforcement learning* is another important approach and one that is closely related to decision processes, game theory and many other fields. Here, the methods developed are constructed to maximize a reward function by choosing between a set of possible actions by trial and error, where no explicit data pairs as the ones earlier are used (Sutton and Barto 2018).

Within supervised machine learning, one of the most proliferous techniques currently are *neural networks* (NNs) (LeCun et al. 2015). In their core, NNs are complex structures built from the combination of other simpler parts, i.e. *perceptrons* (Rosenblatt 1958), that perform a very simple task applying a non-linear function to a linear transformation of the input. In the next years to the excitement that followed the creation of perceptrons, other tentative ML work would be introduced, as for example the Bayesian Methods of Probabilistic Inference in the 1960's (Solomonoff 1964). After a hiatus of almost two decades, interest in NNs would rise again thanks to the advances on different techniques such as *backpropagation* (Rumelhart et al. 1986), which would allow to train complex NN models with the newly-developed computational resources provided by the usage of Graphical Processing Units (GPUs). This would pave the way for the widespread use of AI that we have nowadays.

Due to the advances achieved during the 1990's in ML, NNs would become one of the most popular type of algorithms in use today. During these years, the basis for many of the current widespread methods would be set, including *Support Vector Machines* (SVMs, Cortes and Vapnik 1995) as well as several NN architectures such as *Recurrent Neural Networks* (RNNs, Pearlmutter 1990), *Long-Short Term Memory NNs* (LSTMs, Hochreiter and Schmidhuber 1997) and early versions of *Convolutional NNs* (CNNs, LeCun et al. 1998) in the late part of the decade. These NN models can be framed inside *deep learning*, *i.e.* NN models where there are several processing phases between the input and the estimated target value, which allow NNs to produce much more flexible results. The main consequence of this broad use of ML in a general sense, and more particularly NNs, is the change of perspective from a knowledge-driven to a data-driven approach, in which the algorithms are able to extract the necessary information from the data on their own, automatically finding important features present in the database and fitting their parameters accordingly. This is a key aspect of *deep learning* (DL) methods, which by the 2000-2010's had become one of the most common ML techniques thanks to this ability and to the wide variety of tasks they could be implemented in while achieving strong performance levels. DL is currently employed in very different tasks that range from face recognition (Parkhi et al. 2015) to medical diagnosis (Yadav and Jadhav 2019), text summarization (Song et al. 2019), weather and climate predictions (Elhoseiny et al. 2015), self-driving cars (Fujiyoshi et al. 2019), etc. They also are usually referred to as *deep neural network* models (DNNs).

Nevertheless, the flexibility of DL models comes with certain drawbacks. As an example, they are prone to *overfitting* the data, *i.e.* fitting closely the training data at the expense of worse results in the testing phase. Moreover, complex models can require copious amounts of data to be trained properly, which may not always be the case. Furthermore, the estimated outputs do not allow us to extract any information about how certain the model is about that outcome. This is mostly caused by the *black-box* nature of these algorithms, which implies that it is difficult for a human to understand what is being done in each part of the process. This represents a complex problem which affects many usages of DL, specially in cases such as when there may be intrinsic biases present in the data and may want to avoid the effects of this in the final predictions (Serna et al. 2019). There are many possible paths to deal with these problems, and one of the most popular ones is substituting the point-wise predictions from the usual methods for predictive probability distributions. This change would inform the user about the uncertainty of the output from the algorithms, potentially encapsulating all uncertainty sources into the final results. This would allow the user to make more informed decisions based on this detailed data, which could be of utmost importance for certain sensitive problems. It is here where the Bayesian formulation presents an important tool to extend models into this probabilistic approach.

On the other hand, *Gaussian processes* (GPs) represent a different kind of supervised learning model that are capable of conducting exact Bayesian inference (Williams and Rasmussen 2006). GPs represent a stochastic process, *i.e.* a collection of indexed random variables, such that any finite collection of those random variables is normally distributed. The random variables in this case represent any function that receives the observed data as input and outputs an estimate of the target values, and due to the Gaussian restriction imposed, the calculations involved will have a closed-form expression. However, GPs have important obstacles: they scale poorly with big datasets, and their predictive distributions will only be Gaussian. *Sparse GPs* (Quinonero-Candela and Rasmussen 2005; Titsias 2009; Snelson and Ghahramani 2005) help to prevent the first of these issues by introducing an extra mechanism in the GP formulation: the usage of *inducing points*. This new set of parameters allow for summarizing the complete GP and improve the memory requirements needed to execute the GP. However, the second problem of GPs is much more difficult to deal with. This represents a major disadvantage for GP-based models, since real-world problems present more complex behaviors. Nonetheless, GPs represent an important introduction to approximate inference methods in the function-space, which will prove to be crucial in later chapters of the thesis.

1.1.2 Bayesian Machine Learning

Bayesian statistics provides a framework that can be used in the context of ML to compute uncertainties in a predictive model, among other things. In Bayesian statistics, probability expresses a *degree of belief* in events (Bishop 2006). The Bayesian approach to learning consists on encoding our prior beliefs in a *prior probability distribution* and updating it according to the observed data through Bayes' theorem, which can be stated in words as

$$\text{posterior} \propto \text{likelihood} \times \text{prior}. \tag{1.1.1}$$

Here, the prior distribution represents previous beliefs about the behavior of the model, given previous information and considerations about the data and the model itself. On the other hand, the likelihood factor expresses how probable the observed data set is for different settings of model parameters. Finally, the posterior distribution results as the combination of both terms in the right side of the expression, representing the uncertainty in the model parameters having observed the training data. This procedure is usually referred to as *Bayesian inference*. The language introduced by this formalism is naturally fit to describe ML models, where probability distributions of the parameters are modified according to the data observed, obtaining an updated version of those parameters' distributions.

As an example within ML, applying the Bayesian approach to DNNs results in the formulation of Bayesian Deep Neural Networks (BNNs). The usual approach for BNNs implies substituting the usual parameters of NNs with probability distributions to account for the sources of uncertainty on the model, which can be related to the uncertainty in the model's structure but also in its parameters. As far as in 1987, Denker et al. 1987 hints at a Bayesian integration over the network parameters, although a more solid formulation for BNNs would be presented later by MacKay 1992. Nevertheless, the BNNs there proposed could not be easily implemented since they do not satisfy current requirements for their widespread use, mainly due to concerns about their scalability and versatility. In general, Bayesian inference for complex models is intractable and has no closed-form solutions, which posed a major difficulty to the development of these models as well. This intractability is related in most cases to obtaining the correct normalizing constant that would make the posterior distribution in (1.1.1) a properly normalized probability distribution (*i.e.* the posterior integrates to 1). The lack for a closed-form expression here represents a major hindrance when conducting Bayesian inference, as well as other problems that arise dealing with integrals in higher dimensions or evaluating sums that involve exponential number of terms (Bishop 2006; MacKay 2003).

The stagnation of the field caused by all the issues raised in the inference procedure would be overcome by using different approximation techniques for most practical applications. In these cases, an approximate solution to some of the intractable quantities would be constructed in order to be able to complete the inference process. We can group approximate solutions to this problem in two main groups: *sampling-based* and *optimization-based* solutions. In the first case, methods are mostly based on Markov chain Monte Carlo (MCMC), a technique that employs Markov Chains whose stationary distributions coincides with the distribution we are trying to approximate, in most cases, the posterior distribution of the parameters of the model (Neal 2011; Bishop 2006). The samples from the Markov Chain can be used to obtain an estimate of the exact target distribution, which in the asymptotic limit provides the exact distribution. However, these methods come with the issue of high computational cost, even when more efficient formulations such as *Hamilton Monte Carlo* (HMC) are used. For many practical purposes, this poses a strong limitation which makes sampling-based methods prohibitively expensive.

The optimization-based approach to approximate inference is more recent (Jaakkola 2001; Minka 2001a; Beal 2003). These methods usually rely on approximating the exact distributions with simpler, tractable distributions. Here we have approaches such as Variational Inference (VI) and Expectation Propagation (EP) (Jordan et al.

1999; Minka 2001a; Bishop 2006). In both cases, assumptions are needed to simplify the calculations and to obtain closed-form solutions for many calculations needed in the inference process. These restrictions usually introduce a strong bias in the resulting approximation, restricting its flexibility in exchange for a more manageable theoretical approach. Alternative approaches based on *implicit distributions* represent an innovative way to obtain increased flexibility in the models, although they can be more complex to train (Li and Liu 2016). In these models, one can easily sample from the approximating distribution, while it lacks a closed-form density. This introduces a new set of methods that could, in principle, provide more general and scalable results (Salimans et al. 2015; Li and Liu 2016; Mescheder et al. 2017). Furthermore, in many of these setups the approximating distribution is obtained as the result of minimizing the divergence between the proposed approximation and the exact posterior distribution, *i.e.* the *Kullback-Leibler* divergence (KL). The properties of the KL divergence directly affect the outcome of the inference process, and therefore much research has been put into characterizing it. The KL divergence can be seen as a particular case of the more general *Renyi divergences*, also called α-divergences (Van Erven and Harremos 2014; Amari 2012; Li and Turner 2016). New techniques have attempted to exploit the properties of α-divergences to improve the inference performance, although this is still an open research topic (Hernández-Lobato et al. 2016).

Finally, there have been recent efforts into extending the formulation of inference problems from parameter-space to function-space. Although the formulation of the methods becomes more complex, conducting optimization in function-space simplifies many intrinsic problems in weight space, *e.g.* multiple modes, strong correlations between different parameters and high dimensionality in bigger models (Sun et al. 2019). One of the most important tools here are *implicit processes* (IPs), which serve as a general framework to implicitly define multivariate distributions over finite collections of random variables (Ma et al. 2019). IPs can be seen as a generalization of the GP formulation, which allows to get rid of the Gaussianity assumption made in those models. However, due to the complexities in the mathematical formulation, conducting inference with IPs is a challenging task. Despite of this, there have been important advances in recent years with promising results (Ma et al. 2019; Sun et al. 2019).

1.2 Contributions

Given the ideas presented in Section 1.1, the main contributions of this thesis revolve around the extension of approximate inference methods to try to improve their scalability and performance, mostly in the context of supervised regression problems. In general, we can summarize the main contributions made as the following:

1. Extend on the previous formulation of implicit approximate inference models by including a more general set of divergences in the objective function. To this end, we propose a new general method based on minimizing α-divergences

that allows for flexible approximate distributions. This versatility could not be achieved by previous models such as *Adversarial Variational Bayes* (Mescheder et al. 2017), since its objective function presents important limitations, *e.g.* it is restricted to optimizing the KL divergence. By employing the more general class of α-divergences we are able to exploit their different properties (Van Erven and Harremos 2014). We call this method *adversarial α-divergence minimization* (AADM), where we expand on the setup of Hernández-Lobato et al. 2016 to use it in implicit models, which adds extra flexibility to the approximate distribution (Rodríguez Santana and Hernández-Lobato 2020). We have evaluated AADM in the context of Bayesian neural networks, which serve as base models to display the novel properties of the proposed approach. We show that this new method is able to reproduce complex behaviors in the predictive distribution such as multiple modes, heteroscedastic noise, heavy tails, etc. Moreover, extensive experiments are used to prove that AADM achieves better results overall in terms of various metrics in regression problems. We have also conducted experiments in classification setups, where AADM provides competitive results. Furthermore, we also show that the choice of parameter α can be optimized to improve the performance of the method in the selected metric. In general we observe that in most cases, optimal performance is achieved by $0 < \alpha < 1$. Moreover, we show that $\alpha \to 1$ improves further the log-likelihood loss at the expense of deteriorating the squared error, while $\alpha \to 0$ does the opposite. Finally, we show that this improved flexibility is obtained at a similar computational cost to previous methods such as AVB, making our method highly scalable, flexible and efficient (Rodríguez Santana and Hernández-Lobato 2020).

2. Improve the current approaches for function-space approximate Bayesian inference through the usage of *implicit processes* (IPs), and propose a new, general-purposed approach that is able to conduct inference without compromising the flexibility of the IP models. Existing methods employ IPs to approximate the exact prior and/or the posterior of the model. As an example, in one case they are able to find a good approximation to the prior by using an IP, although they must resort to a Gaussian approximation for the predictive distribution (Ma et al. 2019). On the other hand, another possibility is using another IP to approximate the posterior at the expense of not being able to train the parameters of the prior IP approximation (Sun et al. 2019). We propose here the first general-purposed method that can carry out both tasks, enabling us to train the parameters of both the prior IP and the IP approximation to the posterior in a simultaneous fashion, while also providing complex predictive distributions in an scalable manner. To this end, we rely on an inducing-point representation of the prior IP, as it is often done in the context of sparse Gaussian processes (Titsias 2009; Snelson and Ghahramani 2005). The resulting method conducts approximate inference using IPs with implicit distributions, obtaining an approximate distribution that is decomposed into a

sparse GP and an IP. We name this method *Sparse Implicit Processes* (SIP). SIP is also capable of reproducing complex features such as multimodality, heavy tails, heteroscedasticity, etc. Moreover, due to the usage of inducing points, the method is kept scalable (in some setups, achieving substantially shorter convergence times than other approaches). Moreover, we show that SIP is capable of efficiently allocate its resources *s.a.* the inducing points' locations, to improve its performance. Finally, we have also conducted extensive experiments which show that SIP outperforms previously proposed methods due to its flexibility, while also being the only IP-based approach capable of training both the prior and the posterior according to the observed data (Sun et al. 2019; Ma et al. 2019).

Publications

For the sake of reproducibility, all the code developed for the contributions and experiments of this thesis is publicly available online[1]. In addition to this, the following paper has been published

> Rodríguez Santana, S. and Hernández-Lobato, D. (2020) Adversarial α-divergence minimization for Bayesian approximate inference. *Neurocomputing.* ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2020.09.076.

while although of similar importance to the development of this thesis, currently undergoing the review process

> Rodríguez Santana, S., Zaldivar, B. and Hernández-Lobato, D. (2021) Sparse Implicit Processes for Approximate Inference. Submitted to the *25th International Conference on Artificial Intelligence and Statistics* (AISTATS 2022).

Finally, although not described in detail here, work related to this thesis can be found in

> Naveiro, R., Rodríguez Santana, S. and Ríos Insua, D. (2019). Large scale automated forecasting for network safety and security monitoring. *Applied Stochastic Models in Business and Industry*, 35(3), 431–447, https://doi.org/10.1002/asmb.2436.

1.3 Dissertation Structure

The organization of the remaining chapters of this thesis is as follows:

> **Chapter 2** is a review of neural networks and Gaussian processes. The basic concepts are introduced and defined in both cases. The NNs section goes over the fundaments of shallow nets, their activation functions and some

[1] https://github.com/simonrsantana

regularization techniques. Afterwards we describe several architectures of *deep NN* models, mainly feed-forward and convolutional neural networks, and point out their core properties. Finally we describe Gaussian processes in their most basic form, starting from the kernel formulation and exploring some of their properties. To finish, we introduce one approximation to sparse GPs and the concept of inducing points.

Chapter 3 introduces Bayesian inference in a more detailed manner, starting from basic Bayesian formulation. Bayesian learning is then introduced in the context of ML and, more importantly, Bayesian NN models. After explaining the main obstacles for exact Bayesian inference in this context, we cover several approximate inference techniques. Within this context, Variational Inference (VI) is further detailed, including a new approximate solution for sparse GPs that makes use of the VI formalism. The expectation propagation algorithm is also introduced, followed by a first implicit model for approximate inference. Some sampling-based approaches based on Markov chains are also summarized. Finally, we describe current state-of-the-art methods based on function-space inference with implicit processes (Ma et al. 2019; Sun et al. 2019), illustrating their core properties and limitations.

Chapter 4 focuses on extending the existing formulation for approximate inference in parameter space to employ α-divergences. These represent a more general type of divergence whose properties can be beneficial for inference tasks. Employing the formulation provided by previous literature works such as *power expectation propagation* (Minka 2004) or *Black-box α-minimization* (Hernández-Lobato et al. 2016), we propose a new approach to minimize α-divergences with an implicit model for the approximate distribution by making use of an adversarial approach, which we refer to as *Adversarial α-divergence minimization* (AADM). We explore its properties through numerous experiments, showing it achieves better overall performance than previous methods proposed in the literature.

Chapter 5 introduces *Sparse Implicit Processes*, a new technique to conduct approximate inference in the space of functions using implicit processes. Using the introduction to IP-based methods in Chapter 3, we formulate the new method, which employs an IP prior and a combination of another IP and a sparse GP in the posterior. This results in a in a mixture of Gaussians predictive distribution capable of reproducing complex patterns, while the model remains fully trainable in a scalable manner. We report the results of extensive regression experiments, showing that SIP's performance improves overall the achievements of the previous state-of-the-art, while it is the first general-purposed, flexible and fully trainable model based on approximate inference with IPs.

Chapter 6 summarizes the conclusions of this thesis and proposes new lines for future research.

Chapter 2

Neural networks and Gaussian Processes

In this chapter we will conduct a brief review of two of the most popular approaches inside modern machine learning: neural networks (NNs) and Gaussian processes (GPs). NNs represent the most widespread method currently due to their adaptability and performance in many different tasks. This is related to the multiple architectures that can be used here to take advantage of different data, allowing to account for spatial and temporal correlations in some cases. We will introduce the basic concepts on which NNs rely on, followed by a description of some of the most successful NN models. On the other hand, GP-based models are useful to conduct Bayesian inference using functions as random variables. Imposing a set of restrictions on the functions available, all of the important quantities for inference have closed-forms, although that comes at the expense of low scalability and forcing the predictions to be Gaussian. This will provide us with a first approach for conducting inference in the space of functions, which we will use again in following chapters. Finally, we will introduce sparse GPs, a model that enables GPs to be used in cases with large amounts of data, greatly reducing their computational requirements.

2.1 Fundaments of Neural Networks

We have experienced a rapid increase in public interest during the past few decades on topics related to computer science, statistics and, more specifically, machine learning and pattern recognition. The reasons behind this interest can be traced to, although not exclusively, the appearance of new and powerful hardware, as well as algorithms capable of exploiting this new set of capabilities available. The creation and accessibility of GPUs (as well as more powerful CPUs) has allowed for the development of a new array of techniques that, in turn, achieve a much higher performance than previous methods on a very wide range of tasks (Oh and Jung 2004). Some of the most popular methods for dealing with complex problems, and also one that has become almost ubiquitous nowadays, are *neural networks* (NNs)

(Hu et al. 2015; Fujiyoshi et al. 2019; Song et al. 2019).

NNs are specially interesting nowadays because the have allowed to make use of both different data structures in an efficient and highly precise manner, which represents a strong contrast to what other popular methods are capable of (Gelman et al. 2013; Bishop 2006). Specially, we here refer to non-tabulated data *e.g.* images, audio, video etc. When dealing with these datasets, regular approaches resorted to complex mathematical and statistical modelling needed to deal with the hardships intrinsic to these problems: much of the effort was invested on dealing with the data itself by identifying its features, and also on how to train and perfect the model so it had the ability to extract as much information as possible from the dataset (Goodfellow et al. 2016). Here, NNs represent a change of paradigm, on which the modelling effort is exchanged by the computational burden that represents training the system itself. This is due to NNs being able to automatically detect important features and structure in the data thanks to the usage of simple but effective approaches to processing the information through the network (Ruck et al. 1990). Moreover, what was a prohibitive computational cost for training these models a couple of decades ago has now become a solved problem (at least for most cases) thanks to the wide access to GPUs and an ever increasing amount of computational power. Models that can be run in most modern computers can achieve what not long ago was considered state-of-the-art performance. This is particularly the case in some complex tasks, ranging from image classification (Krizhevsky et al. 2012) to face and speech recognition (Hu et al. 2015; Li and Wu 2015), text summarization (Song et al. 2019), captioning (Vinyals et al. 2015), etc.

One of the main features of NNs, asides from their ability to perform *automatic feature detection*, is the flexibility of their framework. Many different architectures can be employed to tackle different tasks, while the same principles for training apply to most of them (if not all) (Goodfellow et al. 2016). Most models can be described using very similar terms that could be employed to formulate the most simple setups, being the *Multilayer Perceptrons* (MLPs) maybe one of the most common ones (Rosenblatt 1961; Goodfellow et al. 2016). In this basic form, NNs can be easily formulated and, to some extent, interpreted (Goodfellow et al. 2016). However, for more complex models, interpretability has long been seen as an impossible task, labeling most complex NNs as black-box models, which has been an important line of research for the last years (Abbasi-Asl and Yu 2017). In some limits, however, NNs are also strongly related to other types of methods that may be much more interpretable. This is the case for Gaussian processes, a class of models which we will introduce later as well.

In this part of the text we will describe the most basic concepts that are used in NNs since they play a major role in a lot of ML solutions to real-world problems. We will introduce their most fundamental components and see how to extend their basic capabilities through changes in their architecture and other features, while also describing their main drawbacks. Moreover, in the following chapters we will also see how to extend their features to extract more complete information in their predictions for each problem at hand.

2.1.1 Shallow Neural Networks

Most of the popular NNs architectures can be described in the same terms since they usually share most of their main principles, regardless of their specific architecture or application (Goodfellow et al. 2016). This can be said for some of the most widespread setups for NNs, some of which we will see later. Therefore, to explain the main fundamental basis of NNs we will start by briefly covering the simpler examples, and continue building from there.

One of the main concepts behind NNs are their *processing units*. Although NNs could be considered somewhat new models by some, the introduction of this core idea dates back to the 1950s, when Rosenblatt formulated the *perceptron* (Rosenblatt 1958) . This simple model would lay the ground for what we know now as NNs.

The perceptron was initially formulated as a classifier, although its formulation could be easily extended to a regression objective. This model consists of a simple linear transformation of an input $\mathbf{x}$ by a set of *weights* $\mathbf{w}$ and a *bias* parameter b:

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 1, & \text{if} \quad \mathbf{w}^T\mathbf{x} + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.1.1}$$

where σ is a non-linear function that outputs 1 if the input is positive, and 0 otherwise. This transformation can be described as a threshold function, mapping a certain input value to a binary output depending on this condition. Due to this behavior, the perceptron is also referred to a an *artificial neuron* with a Heaviside function used as an *activation function* (that is, the neuron is *activated* if the input is positive, where returning a 1 is the analog of firing). This setup enabled the perceptron to create linear decision boundaries to classify data (or also fit a linear regression model).

Different perceptrons could also be combined in parallel, resulting in *single-layer perceptrons*. However, these models were only capable of learning linearly-separable patterns, and otherwise they would not be capable of correctly performing the classification task (this includes the inability for solving a Boolean XOR problem). The development of NNs would mostly stall here since this problem was seen as an important hindrance for the development of useful models. This issue could not be solved unless more complex combinations of perceptrons could be trained in a *multiple layer* manner, but the computational power available did not allow for it yet. In the following years, important advances were made regarding the ability to train more complex models, specially with the formulation and application of the *backpropagation* algorithm to NNs (Werbos 1975; Rumelhart et al. 1986). Backpropagation provided an scalable and flexible technique to estimate the gradients of the parameters in a system with respect to an objective function, further enabling the construction of more complex systems of different perceptrons. This, coupled with the rapidly increasing computational power available, allowed for the creation of the first trainable *multilayer perceptrons* (MLP), which in some instances are also called *shallow neural networks* (as a counterpart to *deep neural networks*, which we will introduce shortly as well).

Shallow NNs are usually seen as very simple models, constituted by a single layer of computing units between input and output (which is why they are also referred to as *one-hidden-layer neural networks*). Each one of those *units* is closely related to the original perceptron. In this case, however, we will have two consecutive transformations applied to the input data. Again, we will denote by $\mathbf{x}$ the inputs of the model (column vector with Q elements). The part of the system on which the data is provided is usually referred to as the *input layer*. Following this input, we will have a *weight matrix* $\mathbf{W}_1$ that connects the input and the hidden layer, which will act as a linear map of dimensions $Q \times K$, and also certain biases $\mathbf{b}_1$ to complete the linear transformation (row vector of dimension K). The result of this linear transformation is usually referred to as the *activations* for the layer, which we will denote as $\mathbf{a}_1$ (row vector of dimension K as well). As in the previous example, these activations will be passed through an element-wise non-linear activation function $\sigma(\cdot)$, which in the previous example was simply the standard Heaviside function, providing the layer output $\mathbf{h}_1$ (row vector of shape K). Finally, we process these last outputs through the hidden layer, where we conduct a new linear transformation given by a weight matrix $\mathbf{W_2}$ (dimensions $K \times D$) and biases $\mathbf{b}_2$ (shape D). For this system, the final output will be directly the activations of this second layer ($\mathbf{a}_2$, a column vector of dimension D), without passing them a second time through an activation function such as before. Therefore, the whole process for a given input $\mathbf{x}$ through this system can be described with the following expressions:

$$\begin{aligned}
\hat{y} &= \mathbf{a}_2 \\
&= \mathbf{h}_1\mathbf{W}_2 + \mathbf{b}_1 \\
&= \sigma(\mathbf{a}_1)\mathbf{W}_2 + \mathbf{b}_1 \\
&= \sigma(\mathbf{x}^{\mathbf{T}}\mathbf{W}_1 + \mathbf{b}_1)^T\mathbf{W}_2 + \mathbf{b}_2. \qquad (2.1.2)
\end{aligned}$$

The dimensions of each component in the system reveals the number of units being used in each part of the processing: for the first layer, we will have K units that have to receive the input from $\mathbf{x}$, with Q components. Afterwards, the second and final layer consists in D units that receive the K-dimensional output from the first layer. Since the output for this model is directly the activations for the second layer, we will need to have that D is also the dimension of the desired output y. A representation of this process is illustrated in Figure 2.1, on which the input $\mathbf{x}$ is passed through the first linear transformation, given by $\mathbf{W}_1$ and $\mathbf{b}_1$ and where $\mathbf{w}_i^{[1]}$ represents the i-th column of $\mathbf{W}_1$. The output is then passed to the nonlinear activation function, and finally undergoes the last linear transformation with the parameters of the hidden layer ($\mathbf{W}_2$ and $\mathbf{b}_2$). In this case we have set $\mathbf{x}$ to be a 3-dimensional vector and we will have 4 units in the hidden layer, represented by the four distinct components of $\mathbf{w}_1^{[i]}$. The number of hidden units in the output layer is left unspecified, although it must coincide with the dimension of the output $\hat{y}$.

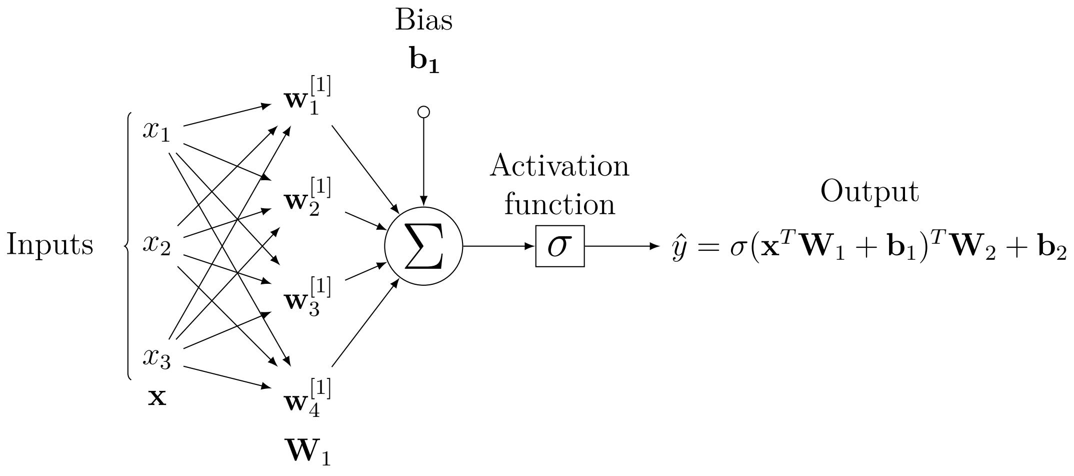


Figure 2.1: One-*hidden-layer* NN

2.1.2 Activation Functions

As we have seen, the core idea behind NNs thus far is the combination of linear and non-linear transformations to a given input. More specifically, the ability of NNs to reproduce complex behaviors completely depends on having non-linearities present, since otherwise the whole processing of the input could be summarized in a linear transformation. The choice of the proper activation function for a given task is therefore crucial both from the point of view of training (*backpropagation*) and for the performance. There has been intensive research on the properties of many possible nonlinear functions as candidates for activation functions. Some of the most popular ones are the following:

- **ReLU**: *Rectified Linear Units* (ReLUs) are one of the most popular choices, either in their standard form or with some modifications. The function can be defined as

  $$\sigma_{\text{ReLU}}(x) = \max(0, x), \tag{2.1.3}$$

  that is, the positive part of the argument. The ReLU function has important properties such as the efficient computation of both the function and its derivative (which aids when training the models) and sparsity (with a randomly initialized network, approximately half of the outputs will be zero). However, it does present some important drawbacks, for example the fact that it is unbounded and that it is not differentiable at $x = 0$. This function is also widely used in NNs, and in more complex models it has shown better performance than other commonly used activation functions. Many different iterations have been proposed for this activation function due its simplicity, being the **Leaky ReLU** one of the most popular ones. On their parametric version, the function consists in introducing an small slope α when $x < 0$:

  $$\sigma_{\text{Leaky ReLU}}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x < 0. \end{cases} \tag{2.1.4}$$

When $\alpha = 0.01$, this function is commonly referred to simply as Leaky ReLU.

As a final remark, ReLU non-linearities also present the important feature of avoiding the vanishing gradients problem. Using other non-linear transformations, the gradient of the activation function can get close to 0 at a certain range of x, which difficults the learning since it depends on these values. However, this is not the case for the ReLUs, since their gradient is always constant when the input is positive. If a regular ReLU is employed, the gradient is only zero in the region where $x < 0$, although this issue can be easily circumvented by choosing a Leaky ReLU model and allowing for a small learning when the input values are negative. This issue plays an important role in other activation functions, such as the one we will introduce next.

- **Logistic**: The logistic function is also commonly used due to its behavior close to zero, its smoothness and the fact that it gives bounded results. It is given by:

$$\sigma_{\text{Logistic}}(x) = (1 + \exp(-x))^{-1}. \tag{2.1.5}$$

  This function is usually seen as an alternative *smoothed version* of the more classical Heaviside model. This is specially useful for binary classification tasks, since it provides more information than its counterpart. In this case, the derivatives are more costly than those of the ReLU, although the additional computational cost can be assumed with current setups. The logistic activation function can be implemented in the last part of a model, where the output can be interpreted as the probability of a certain input belongs to one of two categories. Finally, the logistic function presents a vanishing gradients problem when the input value is too far from the origin, where the gradient of (2.1.5) becomes 0.

- **Hyperbolic tangent**: The hyperbolic tangent presents a similar shape to that of the logistic function in (2.1.5), while also presenting bounded results and a linear behavior close to the origin. This function can be written as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.1.6}$$

  From this expression we see that, the larger the input, the closer the output will be to 1. On the other hand, the smaller it is, the closer the resulting value will be to -1. The behavior is therefore very similar to the logistic function with a small translation and change in scale (and therefore suffers from the vanishing gradient problem as well). The derivatives of the functions can be estimated with cost similar to the ones of the logistic function as well. Therefore, it provides another smooth way of mapping all possible inputs into a bounded result and has readily accessible gradients, which makes it a very popular choice likewise.

2.1.3 Objective Functions

One of the other key ingredients to NNs is the usage of an objective function to estimate the performance of the established setup for the problem at hand. This objective function provides a formal specification of the problem, and thus its choice is of crucial importance here. As could be expected, its shape and properties will strongly condition the final values of the parameters in our model, since they will be tuned in order to optimize it. In some cases, the optimal value for objective function can be found analytically through a closed-form solution, although in most of the cases we will face in the rest of this thesis this is usually not the case: either the complexity of some parts of the system or the evaluation of the objective function itself will prevent us from obtaining a closed-form solution. However, there are ways of obtaining an approximate optimal solution through numerical approximate methods.

In general, most machine learning algorithms assume an objective function that, while training, can be written as a sum over the evaluation of the same objective function over the training points such as the following:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} L_i(\theta), \tag{2.1.7}$$

where θ represents the model's parameters, N the total number of training instances, and L_i typically represents the objective function evaluated for the i-th data instance. In supervised learning settings, usually each $L_i(\theta)$ refers to an evaluation of the objective function using parameters θ and a pair of input and output data, $(\mathbf{x}, y)$. Using this description, by employing a method such as backpropagation, we can estimate and propagate the gradient of the objective function across the parameters of the model. This allows us to update their values according to this gradient in order to optimize the function. Once the gradients are obtained, the values of the model's parameters can be updated using some technique of *gradient descent* (and in some cases, using an approximation to the exact gradient can help with the optimization procedure). In general, the update of the parameters is conducted as

$$\theta := \theta - \eta\nabla L(\theta) = \theta - \frac{\eta}{N}\sum_{i=1}^{N}\nabla L_i(\theta) \tag{2.1.8}$$

where η is a parameter that controls the scale of the update, usually referred to as *learning rate*. In some cases, SGD can be implemented without needing to evaluate (2.1.8) for all the training inputs, allowing for a *mini-batch* gradient descent training.

The study of optimization techniques constitutes a complex and rich field of research with historical roots, but with major importance today in the machine learning research since it is completely crucial to the training and development of new methods. In particular, some of the most popular techniques for conducting training in ML, specifically NNs, make use of changing learning rates according to previous values of the gradient for the parameters. This allows for a faster convergence to the optimal value in the objective function, since the model can be said to make use of the properties in the optimization space to quickly arrive to the optima. Although there are many different techniques that can be used here, one of the most relevant ones, with widespread use, is *Adaptive Moment Estimation* or *Adam* (Kingma and Ba 2014). In Adam, running averages of both the gradients of the objective function and their second moments are used. Through a combination of momentum terms and adaptive learning rates, convergence to the optimal value of the objective function is accelerated. As well as previous methods, Adam takes advantage of concepts such as the momentum of the updates, but does so in a manner that allows it to converge faster than regular optimization techniques while remaining scalable, efficient, easy to implement and with great performance across many different datasets.

Asides from the optimization techniques, and as we pointed out earlier, the choice of the objective function is of utmost importance in each task. Depending on the type of problem we are trying to model, certain functions are commonly used in order to obtain better results. As an example, for regression problems it is very popular to set the objective function (also referred to as *loss function*) as a distance measure between the estimations of the model and the test data. We will write these functions using θ to describe the model parameters (in the example of Figure 2.1, θ would be $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$). The quadratic loss function, also called **mean squared error** (MSE) loss, is given by

$$L_\theta(\mathbf{x}, y) = \frac{1}{2N} \sum_{i=1}^{N} ||\mathbf{y}_i - \hat{\mathbf{y}}_i||^2 \tag{2.1.9}$$

where $\{\mathbf{y_i} | i = 1, ..., N\}$ are N observed outputs and $\{\hat{\mathbf{y}}_\mathbf{i} | i = 1, ..., N\}$ are the outputs for the model with corresponding inputs (in our example, the activations of the second layer of processing). This is a common choice when the main objective is minimizing the error of the predicted outputs with respect to their observed values. Many other distance definitions can be used depending on the specific details of the regression task. As an example, we could also use the **mean absolute error** loss, which is simply

$$L_\theta(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} |\mathbf{y}_i - \hat{\mathbf{y}}_i|. \tag{2.1.10}$$

Different final results may be obtained for each distance metric we employ in the loss function, since they tend to weight in a different manner different values for the error in the regression task (*e.g.*, MSE may penalize more strongly big error terms than MAE due to its squared term, and thus may be an interesting choice if strong deviations are not desirable in the specific problem they are being used in). These two particular functions represent two of the most popular and widespread loss functions for regression tasks, although there are certainly many more to choose from and which can be implemented and tailored to each use (*e.g.* quantile loss, $\log\cosh$ loss, Huber loss) (Friedman et al. 2001).

On the other hand, if we intend to perform classification tasks it might be useful to employ an objective function that allows us to discriminate between as many classes as needed. In general, these will need to provide us with an estimated value that serves as a proxy to know if certain data instance belongs to a specific class. Therefore, in these systems it is particularly useful to employ activation functions such as (2.1.5) in the output of our system. In this manner, the output of our system can be easily interpreted as said probabilities (we could also make use of a Heaviside function, although in this case the binary output will be equivalent of labeling a given input as belonging to one class or another). Both in the task of binary or multi-class classification, the cross-entropy loss function is commonly used. This function is based on the usage of a *softmax* function between D available classes for each data instance. The score assigned for class d of a given predicted output $\hat{\mathbf{y}}$ associated to an input x, will be:

$$\hat{p}_d = \frac{\exp(\hat{y}_d)}{\sum_{d'} \exp(\hat{y}'_d)} \qquad \text{for } d \in \{1, ..., D\}. \tag{2.1.11}$$

If for that same input x the observed label is y, we can construct the loss function using the previous score function. Taking the log of $\hat{p}_d$ and combining it with the associated y for each data instance, we construct the **cross-entropy** loss function, which in the binary case is simply given by

$$L_\theta(\mathbf{x}, y) = -y \log(\hat{p}_y) - (1 - y) \log(1 - \hat{p}_y), \tag{2.1.12}$$

where y is 0 for one of the classes and 1 for the other (binarized label), and $\hat{p}_y$ is the probability of $\hat{y} = y$, given by (2.1.11). This can be generalized to the case with D different classes, where the **multiclass cross entropy** loss function can be defined as

$$L_\theta(\mathbf{x}, y) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{d=1}^{D} y_{i,d} \log(\hat{p}_{i,d}), \tag{2.1.13}$$

where i refers to each data instance ($i \in \{1, \cdots, N\}$), $y_{i,d}$ is a binary label which takes value 1 if for instance i the observed class is d and is 0 otherwise, and finally $\hat{p}_{i,d}$ is the probability of labeling data instance i as class d. The sum over all data instances can also be performed *batch-wise* in some cases, and it serves as a manner to quantify the classification error rate.

Models such as NNs are pretty flexible, and given an enough complex architecture they are seen as universal approximators (Hornik et al. 1989). This means NNs can represent a wide variety of functions, given the NN at hand is complex enough in terms of the number of units in each processing layer (width of the layer) and enough numbers of layers are applied, as we will see shortly. However, this also means that these models can be prone to *overfitting* the data if they are flexible enough. When this occurs, the model fits its parameters to model closely the training dataset at the expense of the generalization error, which is evaluated in the test set and exceeds the training error greatly. In most cases this is caused by employing a model whose complexity goes beyond what may be needed to model the data, and its also specially common if there are few data points to train the system and these few datapoints available are being used multiple times. This is an important issue that can compromise the behavior of the system, and therefore there has been important research efforts on how to deal with it. The methods that try to prevent the system from overfitting are usually called *regularization techniques*, and here we will briefly introduce some of the most widespread ones in the NN research literature: *Lasso* (L1) and *Ridge* (L2) regularization. The discussion of both is relevant here since they can be implemented as slight changes to the objective function, and although it is not the case for *dropout* regularization, we will also include it here for the sake of completeness since it is one of the most important regularization techniques in current NN research development.

- **Lasso regularization (L1):** The least absolute shrinkage and selection operator or *Lasso* was first introduced in the context of geophysics with the study of band-limited reflection seismograms (Santosa and Symes 1986), although the term Lasso and its introduction to the ML literature would be thanks to the work of Tibshirani (Tibshirani 1996). The Lasso regularization implies adding an extra term to the objective function in the form of the sum of the absolute values of the model's parameters

$$L_\theta^{\mathrm{L1}}(\mathbf{x}, y) = \mathcal{L}_\theta(\mathbf{x}, y) + \lambda \sum_{m=1}^{M} |w_m|, \tag{2.1.14}$$

  where w_m represents each of the M-model's parameters available, λ is a scale parameter and $\mathcal{L}_\theta(\mathbf{x}, y)$ is the original loss function. When this term is added, minimizing the resulting loss function $L_\theta^{\mathrm{Lasso}}$ will also mean minimizing this sum of absolute values, thus discouraging the parameters from setting parameter values too high. Due to the usage of the absolute value, the Lasso regularization is said to create sparse models since it forces many parameters to zero. This is particularly useful if we are trying to compress our model, although may present issues when dealing with high dimensional data and few training points).

- **Ridge regularization (L2):** The Ridge regularization technique (Hoerl and Kennard 1970) uses the L2 norm of the vector of parameters in a similar fashion to what is done in the L1 case. This can be written as

$$L_\theta^{\mathrm{L2}}(\mathbf{x}, y) = \mathcal{L}_\theta(\mathbf{x}, y) + \lambda \sum_{m=1}^{M} w_m^2 = \mathcal{L}_\theta(\mathbf{x}, y) + \lambda \mathbf{w}^T \mathbf{w}, \tag{2.1.15}$$

  where $\mathbf{w}$ is a column vector containing all the model's parameters, of size M and with entries $\mathbf{w}_{[i]} = w_i$. The usage of the L2 norm does not yield as much sparsity as the L1 regularization, but instead encourages the parameters to take values close to zero (it penalizes in a stronger manner outliers with high values). Therefore, the resulting model will be not as sparse as in the L1 case, but all the non-zero parameter values will be more concentrated towards zero, and smaller values for the parameters help restrict the overall flexibility of the system.

- **Dropout:** This approach to regularization is not implemented through the objective function, although since its introduction to the ML research community (Hinton et al. 2012; Srivastava et al. 2014) it has become one of the most widespread ones. Although there are others, dropout is the most popular type of *stochastic regularization technique* (SRTs) on which the regularization is conducted in a stochastic manner, controlled by a random variable. It is mostly used in bigger NN systems, *i.e.* in Deep NNs, which we will describe in further detail in the next section. In these cases, it is specially efficient in order to perform *model averaging*, which means producing outputs from different networks and averaging their results to obtain a better estimate for the final value. This effectively addresses the issue of overfitting, while also being easily implemented in most NN setups. The concept behind dropout is illustrated in Figure 2.2, where the model being depicted is non other than a NN with two hidden layers of processing units, being each unit represented here as a circle. For each layer in a NN, we randomly *drop out* certain units, tuning their output to 0 with given probability p. This p can be tuned according to the position of the processing layer in the network, the number of units in said layer, etc. This has the same effect as *removing* those units from the structure itself, making the resulting system a *thinned version* from the original one. Taking this into account, the system can be said to average between the outputs of many *thinned* networks at test time, which reduces the chances of overfitting and may improve the performance of the model overall. The

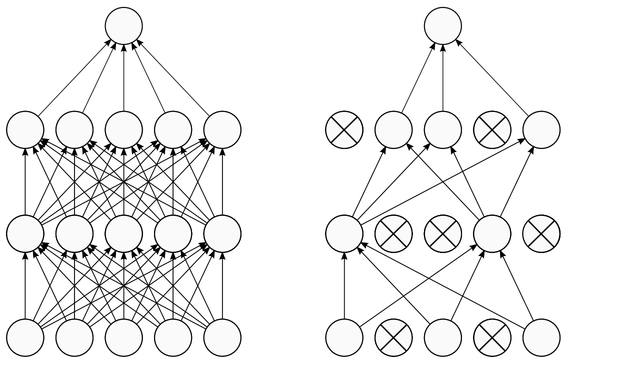

(a) Original network (b) Network after applying dropout

Figure 2.2: Dropout applied to a network. *Left:* Original network, with two consecutive layers of 5 processing units each. *Right:* Same network with dropout applied, where the crossed unit's output has been set to 0 (dropped).

stochastic process for turning off certain units can be seen as noise injected in the system. This noise can be interpreted as noise added in the parameter space, or it can also be re-interpreted as noise coming from the feature space as shown in (Gal 2016).

Moreover, although we will see it in further detail in the following chapter, the implementation of dropout has a similar effect to some approximate Bayesian inference techniques such as *Variational Inference* (see Section 3.3.1 for more details). Under certain constraints for the prior probability distribution and the selected approximating distribution set in our models, dropout produces the same results to those of variational inference. In fact, Gal 2016 conducts the necessary calculations and arrive at the precise set of constraints under which the results of these type of SRTs and variational inference are the same.

Finally, although we have described some of the main parts of shallow NNs, these models still lack the performance expected for many complex tasks nowadays. To solve this issue, we must pay attention to one of the main concepts about NNs that we still have yet to explore here: their architecture. This is the key aspect that will allow us to transition from shallow NNs to *Deep* NNs, where exploiting the inner structure of the models can allow us to greatly improve the performance in a great number of problems.

2.2 Deep Learning

As previously mentioned, ML practices are ubiquitous nowadays thanks to the presence of large amounts of data along with high performance algorithms that allow us to extract valuable information from these sets in an automated fashion. One of the most used techniques in many different fields and a wide range of problems are NNs, which thus far we have only described in a very limited manner. For the most part, we have centered our discussion in shallow NNs, *i.e.* models of networks with a certain number of connected processing units arranged in parallel in what we have called a *layer*. At most, the models described previously had one of this processing layers, and since are deemed *shallow* models (with the exception of the network illustrated in Figure 2.2, with two processing layers). On the other hand, *Deep NNs* (DNNs) can be seen as a simple extension of shallow NNs in the sense that they include a series of layers of units one after another, therefore feeding the output from one layer to the next. A NN is said to be *deep* if it is constituted by 2 or more *hidden* layers, *i.e.* it has 2 or more processing layers between the input and output layers. From here, the concept of *Deep Learning* (DL) is also distilled, making reference to the usage and training of DNN models.

Some of the first DNNs models were formulated long before there were enough advances, both in the computational and theoretical side, to make them effective and useful at a large scale. In some cases, these models were already attempting to tackle difficult problems such as those related to computer vision (Fukushima and Miyake 1982). The first attempts at using backpropagation in DNNs models showed that these models could be trained, albeit in a time-costly manner (LeCun et al. 1989). For many years, the prohibitive computational cost would keep DNNs as an impractical model to use. This would remain the case until the early 2000s, where the combination of increasing computational resources such as the development on *Graphical Processing Units* (GPUs) and the theoretical developments in the training of these deep models would push DNNs to have an important role in ML as a whole and with strong applications to many different problems (Hinton et al. 2006; Hinton 2007). This all layed the ground work for the early 2010s, where the performance of DL algorithms would improve on the previous state of the art, in some cases by surprising margins. This is showcased by the victory of a DNN model in the classification of the ImageNet database, where researchers beat the previous results by employing a NN specialized in image processing (Krizhevsky et al. 2012). This event is used in some cases to mark the starting of the *deep learning revolution*, in which DL became one of the core fields inside Artificial Intelligence research and began being implemented in a very wide variety of complex problems in many different fields across industry and other scientific research fields.

One of the key ideas that have lead to the ubiquity of DL methods are their flexibility and their ability to deal with complex data set without the need for performing feature extraction in beforehand. Through manipulation in their architecture and operators, DNNs allow for a wide range of tasks with datasets which were normally not easy to deal with. This mainly refers to *unstructured data*, *e.g.* images, video and audio files, which in some cases could be used as input data although they needed a thorough preprocessing phase, in some cases needing hand-crafted features. This means that DL algorithms are able to perform *automated feature selection* in an unsupervised manner, which means that given an input and output pair $(\mathbf{x}, y)$, where $\mathbf{x}$ may be an instance of unstructured data (*e.g.* an image) and y a label, the system will be able to automatically detect which parts of $\mathbf{x}$ are relevant to obtain output y. However, the performance of DNNs and their impressive properties come with some caveats. To name a few, DNNs usually require large amounts of data to train all of the parameters present in the network and thus can still be expensive to train, even though some important parts of the computations involved can be parallelized and accelerated with GPUs. Moreover, there is no clear manner to make a choice for the hyperparameters of a model as well as its architecture, and therefore we may need to resort to *cross-validation* techniques to ensure our model performs adequately (which also increases the time and computational costs needed). Finally, and in some cases the most important point, DNNs are usually black-box models, in the sense that the inner working for a given deep NN cannot be easily interpreted by a human (and therefore correcting certain behaviors may be a truly challenging task). Even though these may be important drawbacks in some cases, in many instances they can be dealt with for the most part, which is one of the reasons behind the huge increase in interest in DL techniques of the past decade.

In this part of the text we will introduce a few of the most used DNN architectures, although by no means this intends to be a comprehensive list. DL is used extensively in later chapters of this thesis, and hence we will mostly focus on those models that may come relevant to us in later parts of the text.

2.2.1 Feed-forward Deep Neural Networks

Feed-forward DNNs are one of the simplest DL architectures, since they use the building blocks of previous shallow NNs and simply stack a number of *hidden layers* of processing units in consecutive fashion. As we mentioned before, each layer is an array of units in parallel that process the input given by the previous part in the model, and in the particular case of hidden layers, they are confined to the space between the input and output layers. As we also pointed out, a NN is considered *deep* if the number of hidden layers L is $L \geq 2$ (otherwise, we return to the *shallow* NN case).

In their most basic form, DNNs are simply stacks of layers that process one after another the input given in the input layer, $\mathbf{x}$, and that provide a final estimated output $\hat{y}$. As is done in shallow NNs, this output is then compared against the observed output y through an objective function, and the parameters for the layer are fitted in order to minimize the final loss of the system. These types are similar to the one depicted in Figure 2.3, on which we have an input layer for a 4 dimensional input vector $\mathbf{x}$, followed by four hidden layers of units, and finally an output layer, where the estimated output will be a 2-dimensional vector $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2)$. The four hidden layers in the figure have 6, 7, 6 and 4 units respectively. Each of these units is represented by a white circle, and in each one of them we conduct a succession of a linear and a non-linear transformation, as we did before in Section 2.1.1. This means that the input for each unit is being linearly transformed through a weight matrix $\mathbf{W}^l$ and a bias vector $\mathbf{b}^l$, where l represents the layer's position ($l \in \{1, \cdots, 5\}$). Both the weights and biases can be described in terms of the specific weights and biases for each unit in each layer, namely $\mathbf{w}_i^l$ and $\mathbf{b}_i^l$, where i refers to the unit in that layer, $i \in \{1, \cdots, N_l\}$, and where N^l is the number of units in layer l. The index $l = 5$ will refer to the final output linear transformation, and the hidden layers will be used when $i \in \{1, 2, 3, 4\}$. The output of each part of the model is fed to the next layer, starting from the input layer and *flowing* all the way through until the output layer is reached.

Using the definitions we established for shallow NNs, we can write the output for a given processing layer $\mathbf{h}^l$ given the output of the previous layer $\mathbf{h}^{l-1}$. This is simply:

$$\mathbf{h}^{[l]} = \sigma(\mathbf{a}^l) \qquad \mathbf{a}^l = \mathbf{h}^{\mathbf{l-1}^T}\mathbf{W}^l + \mathbf{b}^l, \tag{2.2.1}$$

where $\mathbf{h}^{l-1}$ is a column vector with size equal to the number of units in layer $l-1$, $\mathbf{W}^l$ is size $(N_{l-1} \times N_l)$ and $\mathbf{b}^l$ is a column vector of size N^l. We see that here we use the concept of *activations* of a layer as well as *activation functions*. Indeed, the combination of linear transformation with these non-linearities is what gives DNNs its power to approximate almost any function.

The example shown in Figure 2.3 represents a relatively small feed-forward fully-connected DNN, which means that each layer output is simply passed forward and where every unit in one layer is connected to all units in the following layer. However, this is not always the case necessarily, since we may have interest on performing certain operations to the inputs in each layer depending on what type of task we may be facing (we will see some examples shortly). Moreover, the architecture of DNNs such as this one can be extended, both in its *width* (number of units in each layer) and its depth (number of layers stacked). However, one must be aware that doing so will effectively increase the number of parameters in the model, maybe making it unfeasible to train unless we dispose of enough time and computing power. Moreover, we have to keep in mind the potential drawbacks that may arise with the choice of these models. This is the case with the cost of performing hyperparameter selection (choosing the best architecture, activation functions, etc.) and the lack of interpretability of the inner values processed by the NN. Moreover, we will need to pay attention to all of the regular issues for every other ML model such as overfitting, which may become specially prevalent here if we increase the flexibility of the DNN. If used correctly, these types of NNs have shown the ability to outperform previous state-of-the-art in various complex tasks.

The feed-forward DNN model represents a milestone in the development of DNNs. Most other important DL techniques can be described in terms similar to these, although they may have slightly different architectures which have been designed to take advantage of the type of data that is being dealt with. This includes networks designed to deal with image and text as input data, where the structure of the data itself (pixels and words) provides relevant information. In the first case, we usually refer to problems such as image classification, image segmentation, image recognition, etc. On the other hand, for texts we have issues related to text summarization, speech recognition, automated language translations, etc. There are many different DNN architectures alternative to the feed-forward DNN, although we will limit ourselves here to *Convolutional DNNs*, a system prepared to deal with and extract information from images.

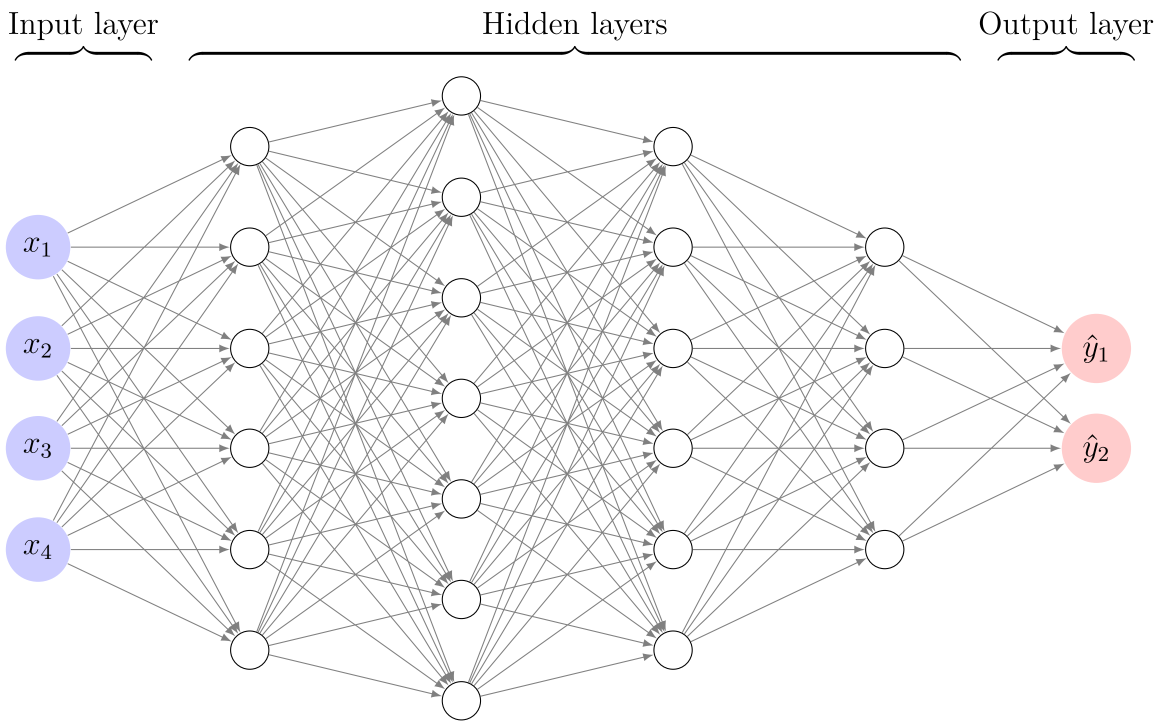


Figure 2.3: Fully-connected feed-forward deep NN

2.2.2 Convolutional Neural Networks

The feed-forward NN model can be extended to make the best out of spatially-distributed data. The distribution of data can hold important information that models such as those cannot properly extract all of the important information contained in the spatial correlations between different input values. The most obvious case here are image data, where the intensity value for each pixel is (usually) strongly correlated to the values of neighboring pixels, since the spatial arrange of elements in a picture is a crucial part of the image information. However, this is not the only case, since developing a technique sensitive to spatial correlations in the data can be potentially employed in any case we see fit, which may include *Natural Language Processing* tasks (such as sentence modelling, classification and prediction), time series forecasting and anomaly detection, study of chemical features depending on the distribution of the atoms in molecules, etc.

Convolutional Neural Networks (CNNs) are the DL models that have been adapted to all of these problems (and many others) due to their versatility and performance. They can be said to be one of the most popular DL methods nowadays, although the basis for such models have been formulated during the 1980s and 1990s (Denker et al. 1989; LeCun et al. 1989; LeCun et al. 1998). They consist in simple changes to the original formulation of the feed-forward fully-connected DNN, on which removing part of these connections is the key concept to exploit the local correlations inside data. Indeed, CNNs extends the basic formulation of the linear transformation and the activation function, and employ new types of processing layer to the DNN architecture. These new layers are the ones responsible from extracting low and high-level features from the data, converting them to numerical form, and then feeding them to a fully connected DNN. Therefore, in a sense, CNNs can be seen as a series of new types of layers stacked on top of a fully-connected DNN. These first part of the model is specially adapted to complex spatially-distributed data and is capable of extracting the relevant information from it. Then, once this procedure is done, the results are passed through a regular DNN and the final output is obtained, be it the type and shape we may need given a specific problem. To understand more in detail how CNNs work, we will now briefly describe the three types of new layers introduced in CNNs which are intrinsic to these types of models and that allow them to make use of spatial correlation information in the data. These are the following:

- *Convolutional layers:* Taking an image as an input to for a given layer, each unit in a convolutional layer performs a *convolution* on the input image employing a *kernel* whose weights have been randomly initialized. In this context, the convolution of the kernel and the image pixels is usually given by the dot product between the kernel components and the image pixels at a given region. The kernels, also called filters, are usually much smaller than the input image itself (specially if the convolutional layer is close to the input layer). The N convolutional kernels in a layer can be seen as N stacks of K matrices, where K is the number of channels of the input image. To fully process an image, each kernel defined will be run through the whole image, moving a predetermined number of positions (pixels) in each step until the whole image is covered. Each kernel can be said to have a *small perceptive field* in terms of their smaller height and width compared to those of the original image, but in each case each kernel is defined in all of the channels of the input (full depth). The resulting output will be a 2-D map of activations for each filter. In Figure 2.4 we can see a depiction of this process, where we must keep in mind that we are referring to an image as a collection 2-D maps of intensities across a certain number of channels. In the figure, N kernels of depth 3 are applied to a region in the input image, which also has $M = 3$ channels. The results from each convolution operation is represented in the output image as a region with a different color for each kernel, where now the number of channels in the output will match the number of kernels employed (N). During this process the output image will have smaller dimensions, with the exact change in shape given by the size of the kernels used and the length of the steps set to cover the whole image.

Convolutional layers can be understood as each unit focusing on small portions of the previous image one by one, and the stacking of these operations is what allows the system to extract even high-level information about correlations in the original image that may occur in in larger scales than what each individual kernel is able to focus on. If several convolutional layers are stacked one after another, the final convolutions will be made in an image for which each pixel can represent a large portion of the original image. Therefore, this constitutes a way of encoding that high and low-level spatial information originally present in the input data.

- *Pooling layers:* Although convolutional layers are effective in terms of concentrating the information of a region of the image, in order to reduce the dimensionality of the images used as input, *pooling layers* have to be used. Usually placed directly after convolutional layers, pooling layers simply take the input image and reduces its size by approximating certain parts of the resulting image with some representative value. There are many conventions taken here, since the main purpose here is to simply down-sample the input image to reduce its size. Some of the examples include substituting each $n \times n'$ block (usually $n = n'$) in the original image by its maximum, average, center value, etc. Consecutive applications of pooling operations allow the image to be converted into smaller and more manageable data, which in the end allow it to be rearranged into a vector that can be fed to the *fully connected* layers.

- *ReLU layers:* As could be expected, ReLU layers simply take the input image and passes it through ReLU functions such as we introduced them in Section 2.1.2. This effectively removes the negative values obtained as a result in previous convolutional layers, and introduces the non-linear transformation which we mentioned was crucial for the inner working of DNNs. The ReLU functions here can be changed to other type of non-linear activation function depending on the task at hand.

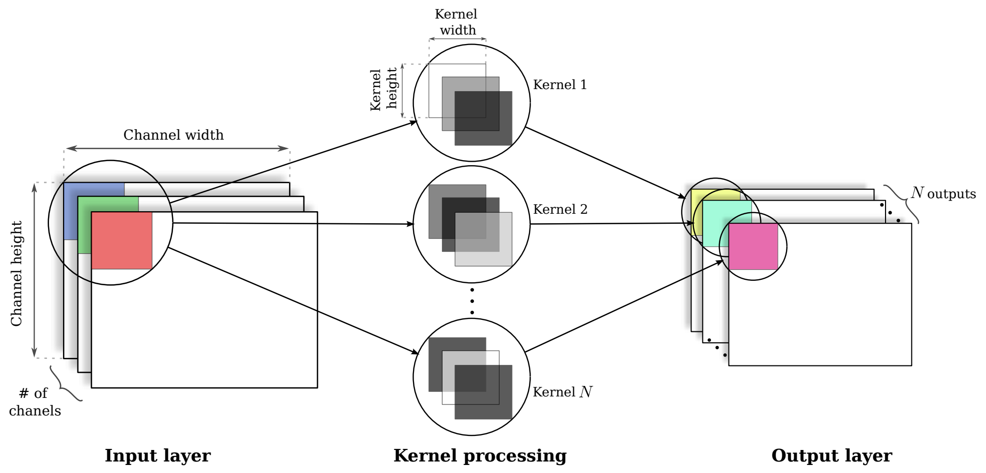


Figure 2.4: Convolutional layer in a CNN. The input image has several channels, and for each one a kernel is convolved with each image patch (depicted by the left-most circle). Multiple kernels are used, and the output of each of them convolved with the input patch is represented as a colored region in the right-most image circles. Different colors imply different values for the filters on each kernel, since they are all given the same inputs. Best seen in color.

- *Fully connected layers:* Finally, as we mentioned earlier, CNNs also have a number of fully-connected layers such as the ones in the previous section. These layers take as input the results of the iterative process of stacking convolutional, pooling and ReLU layers, and process these values into the desired output. These types of layers are the last ones to be used in a CNN, since they depend on first extracting the information of the images through the previous process. Once this is done and the input has been sufficiently compressed, CNNs flatten the results of the last convolutional part of the model into a vector and feeds it to these fully-connected part. That is when regular NN computations can be performed to obtain the resulting outputs.

A key idea of CNNs is the *sharing of weights* inside each convolutional layer: since each kernel is applied with the same values to the whole image, the number of parameters in a CNN does not scale as fast as to make them prohibitive to train. This, alongside their ability to extract important features from complex data such as images with multiple channels, has made CNNs one of the most popular models across different industry applications and research fields. The most obvious impact of the development of CNNs is showcased by the huge improvements made in the field of computer vision, on which since the conception of trainable CNNs and their posterior refinement, they have remained the top-performing models, and in some cases, by wide margins. This dates back to Krizhevsky et al. 2012, and from that point CNNs have become present in security systems, autonomous vehicles, medicine, finance, neuroscience, and many other areas of development (Kalchbrenner et al. 2014; Li et al. 2014; Hu et al. 2015; Sarraf and Tofighi 2016; Yamashita et al. 2018;

Hosaka 2019). One important example that may be illustrative on how these models are being used is climate and weather predictions and renewable energy production forecasting, as can be seen in (Liu et al. 2016) and (Díaz-Vico et al. 2017). In this cases, the input of the CNN is an image with several channels, each one containing information about a different aspect of the climate data for the region observed (in the first case they use mostly air pressure information, while for the second one they encode temperature as well as pressure at different heights, maximum and minimum temperatures, levels of solar radiation at different wavelengths, wind speed, etc.). CNNs make use of all this spatially-distributed data to yield better better results than previous algorithms applied to this same databases. The possibilities for CNNs and DNNs in a broader sense are vast, and their performance and achievements can be crucial to the development of potentially huge new technologies such as automated medical diagnostics from medical images, self driving cars, drug design and discovery and many others, while also being an important tool to help us deal with complicated issues such as climate change and forecasting and control of epidemic diseases. However, for more robust predictions it would be important to have information about the certainty of their estimations, which is something we will see in the following chapter.

2.2.3 Other Neural Network Systems

We have seen some examples of different usages and architectures that are available when constructing NNs. There are many other relevant models asides from the ones we have introduced earlier, and here we will briefly mention two more to serve as further examples of the flexibility of NNs and their adaptability to different contexts and issues.

- *Generative Adversarial Networks* (GANs), firstly introduced by Goodfellow et al. 2014 represent a new available structure for DNNs which sets up a min-max adversarial game between two neural networks (a generative model G and a discriminator D). The discriminator network $D(x)$ computes the probability that a point x in data space is a sample from a certain data distribution which we are trying to model (positive samples) rather than a sample from out generative model $G(z)$ (negative samples). Thus, the function $G(z)$ is used by the generator in order to map samples z from the prior $p(z)$ to the data space, and it is trained to maximally confuse the discriminator into believing that samples it generates come from the original (positive) data distribution. The solution to the game here involved can be expressed as

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z))] \tag{2.2.2}$$

  Both D and G use stochastic gradient descent in two stages:

  - Train the discriminator to distinguish between true (original) and fake (made by the generator) samples

  - Train the generator so as to fool the discriminator with the generated samples.

  When trained this system can be used to generate new instances for the train data. The combination between the generator model and the discriminator allows the first to create new data instances that, in principle, can be interpreted as very similar instances to the training data. As an example, if the GAN is trained on a certain image dataset, it could learn to generate new images that could look somewhat authentic to human observers. Because of this, GANs were seen as an important type of generative model that was able to conduct *unsupervised learning*, since it needed no data labels to be trained (Karras et al. 2019). This usefulness has also been extended later to *supervised* (Isola et al. 2017) and *semi-supervised* learning (Salimans et al. 2016), as well as *reinforcement learning* (Ho and Ermon 2016). However, GANs are notoriously hard to train since their convergence is not always guaranteed, which is a currently open research topic (Mescheder et al. 2018; Salimans et al. 2016).

- *Recurrent NNs* (RNNs) represent a general class of NN architectures that employ information of previous data passes through the network to conform the outputs in later passes. The NN architecture here is set up so that it conforms a directed graph along through a temporal sequence, using previous outputs as inputs in different stages of the process. This encompasses a wide range of models, most of which make use of time-dependent data and try to exploit time correlations to improve the performance. The basic structure for RNNs is shown in Figure 2.5 and the model works as follows: the input data is represented as $x^{[t]}$, where t is an index that orders the data instances (which could refer to time ordering). For each t, the input is processed alongside $a^{[t-1]}$, which symbolizes here certain outputs from the run with index $t-1$, and that may include the outputs from hidden units, the final output of the system at $t-1$ or any type of similar results. Both of these are processed by the NN for each value of t, which is represented with green squares in the figure, and in each run the NN may produce an estimated output symbolized by $\hat{y}^{[t-1]}$.

  Performing the computation in this time-dependent manner, where the ordering of the data being used also encodes information, makes these NN models specially useful when dealing with time-sensitive data such as those tasks related to language. In this regard, one of the most extended models are the *Long Short Term Memory* NNs (LSTMs), firstly introduced by Hochreiter and Schmidhuber 1997. LSTMs are able to solve some of the issues that hinder the performance of vanilla RNNs, mainly the *vanishing gradients* problem on which the system fails to properly update its parameters according to data. LSTMs are more robust to this problem by the usage of a *forget gate*, an extra part of the system that stores part of the previous information passed to the model and updates it according to new data being presented. This approach has made LSTMs the go-to approach in language processing tasks, ranging from

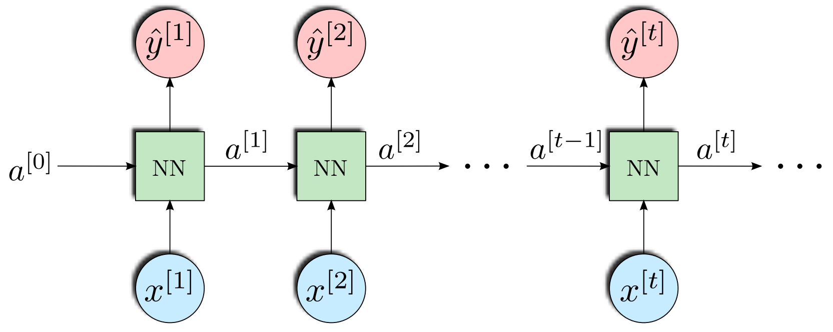


Figure 2.5: Basic unfolded representation of a general RNN model. Blue circular nodes represent data inputs, red circular nodes represent estimated outputs and green squares represent NN processing. Best seen in color.

speech recognition (Sak et al. 2014; Li and Wu 2015) to language modelling (Jozefowicz et al. 2016), automated translation (Sutskever et al. 2014) and image captioning (Vinyals et al. 2015). These are only a few examples, although there are many others topics on which LSTMs have excellent performances and other topics on which RNNs and LSTMs are still being optimized (Gao et al. 2017; Qing and Niu 2018).

2.3 Gaussian Processes

During the last two decades, a lot of research has been conducted into *kernel machines* inside machine learning. One of the most popular examples here are *Support Vector Machines*, but *Gaussian processes* (GPs) have gathered increasing interest as well. GPs represent a model class that is capable of providing a theoretically sound probabilistic approach to learning in kernel machines (Rasmussen 2003). Thanks to the theoretical and practical developments during these past few years, GPs can be considered as a strong contender for supervised machine learning applications. In this part of the text we will review in further detail the formalism of GPs to understand in this approximation to Bayesian inference in an analytical fashion. We will see that GPs are capable of obtaining closed-form expressions for all the important quantities needed to conduct inference, which is an uncommon quality among many other popular machine learning approaches. This makes GP models specially interpretable when compared with current go-to approaches such as NNs, which is another attractive quality that has lead to their usage across many different setups (Rasmussen 2003; Bishop 2006).

GPs have been the object of extensive research to characterize and understand their core properties. In fact, in specific areas, GPs have become the standard method for modelling the data. This can be illustrated by their common use in weather and climate-related problems, where GP models are widely used for interpolation and prediction. Inside the context of weather and climate data, examples of the usage of GPs are wind and solar energy forecasts (Chen et al. 2013; Salcedo-Sanz et al. 2014), precipitation forecast (Wang et al. 2021) and simulation (Kleiber et al. 2012), forecasting greenhouse gas emissions (Fang et al. 2018), to name a few. Other important examples include geostatistics (Datta et al. 2016), physics (Fox 1978), chemistry (Cui and Krems 2016), genetics (Chu et al. 2005), etc. Moreover, *kriging* plays an important role in these contexts (Cressie 1990). Kriging is simply a multidimensional GP regression model usually employed to conduct interpolation and prediction in complex spatially-distributed data. It is a common technique in environmental studies (Bayraktar and Turalioglu 2005), hydrogeology (Delhomme 1978; Chiles, Delfiner, et al. 1999; Zimmerman et al. 1998) and other fields of research such as astrophysics (Pastorello et al. 2014) and material sciences (Zhang et al. 2014), among many others. Therefore, although we cannot talk about a complete widespread usage of GPs, they currently play an important role in many applications of approximate inference due to properties such as their performance, interpretability, ease-of-use and their analytical formulation of the problems, among others. Nevertheless, we will also see some of their most important drawbacks, mostly related to their scalability and the restrictive formulation of the predictive distribution. There has been important advances on tackling these very same problems from different perspectives, and in some cases they can be avoided altogether. An specially important contribution to the work on this thesis is the formulation of *Sparse GPs* (Snelson and Ghahramani 2007; Titsias 2009), a method that reduces the scalability problem at the cost of introducing a parameter in the model. Although this is not the only way to obtain better scalability in GPs, we will see that it is a very effective approach to that end.

Finally, asides from their close relation to techniques such as *Support Vector Machines* and other different *kernel machines*, GPs are also mathematically equivalent to other important approaches such as Bayesian linear models and spline models. More importantly for our case, there is a strong connection between GPs and very large NNs, as we will also describe here. Thanks to this equivalence, in some cases GPs may be a good alternative to these other models, such as certain types of neural networks, since they may be easier to interpret and to handle. Nonetheless, we will firstly describe briefly the basis of the GP formulation, kernel methods, and from there we will advance to the description GPs and more.

2.3.1 Kernel Methods

Among many other frameworks for performing regression and classification tasks, *kernel methods* (KM) are one of the most popular approaches. A comprehensive description of these methods is out of the scope of the main objectives in this text, but a brief introduction to the concept of kernel function may be beneficial to a more organic view of the GP formulation. This is due to the fact that GPs can be seen as an specific formulation of kernel method that make use of a Gaussian kernel to define its prior (Bishop 2006; Williams and Rasmussen 2006). By doing this, GPs automatically inherit important properties that have already been described and properly characterized for regular kernel methods, and therefore will benefit from the previous work in this field. Moreover, KM lay the basic components that are the same that set the basis for many other important techniques, among which we find *Support Vector Machines* (Hearst et al. 1998; Steinwart and Christmann 2008), *Principal Component Analysis* (Jolliffe 2005), *Ridge regression* (Hoerl and Kennard 1970) among many others (Williams and Rasmussen 2006).

Parameter-based models such as NNs make use of training points in order to obtain estimated values for their respective parameters. After this procedure takes place, these points are effectively *discarded*, since the model does not need to retain them as long as they have already been used for optimizing these aforementioned parameters (Williams and Rasmussen 2006). On the other hand, *kernel methods* are constructed as a memory-based approach that makes us of the concept of *kernel functions*. This means that, in most cases, KM will require storing the whole dataset in memory or, at least, a significant part of it. This formulation differs strongly from the parameter-based models, and it is done so that the input data can also be used in the testing phase. This procedure is governed by the fact that the kernel function is seen as a *feature space mapping* from the input space to a more abstract representation, and it is in this newly obtained representation on which the model is capable of quantifying how similar two different data instances are between them (Bishop 2006). The same can be said in terms of the inference problem: in these cases, the similarity measure is conducted between the new (test) and old (train) points. The resulting outcome of the similarities for each new point will deliver us the predictive values output by the method.

As one could expect from the previous description, the core ingredient in KM is the kernel function $k(\cdot, \cdot)$, whose inputs are two arbitrary training points. This function is responsible for that similarity measures between points in the feature space, and therefore being able to properly characterize it and use it is of crucial importance. In general, this kernel function can be decomposed into a scalar product between the same function evaluated in the two different input points:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \tag{2.3.1}$$

with $\phi(\cdot)$ representing the feature-space mapping we mentioned earlier. This formulation allows for many different $\phi(\cdot)$ functions to be employed in the kernel function: as long as we can represent $k(\cdot, \cdot)$ as an inner product of a function times itself in some space, we can use this description to construct a valid kernel. Moreover, certain properties can be distilled from this expression alone, *e.g.* the kernel function must be symmetric in its inputs $(k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i))$.

As we mentioned, there are many possible kernel functions that can be chosen from. This flexibility has lead to the development of many different techniques that can benefit from a general-purposed framework such as this one. In most cases, we can define a kernel by making sure that its associated Gram matrix $\mathbf{K}$ is positive semi-definite. This matrix is constructed as $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, with inputs $\{\mathbf{x}_i, \mathbf{x}_j\} \in \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, where $\{x_k\}_{k=1}^N$ represents our training dataset. However, some cases has been shown to work even when $\mathbf{K}$ is not positive semi-definite (Ong et al. 2004).

One of the main features of kernel formulation is the ability to exchange kernels according to the needs of each problem. As long as a kernel can be decomposed into a inner product inside of a given feature space it will constitute also a valid kernel, while also simple combinations of them will define a new valid kernel as well, including (but not restricted to) linear transformations, exponentiation, additions, etc. As an example of how this can be used to construct one of the most widely used kernels, consider the following function

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma}||\mathbf{x}_i - \mathbf{x}_j||^2\right). \tag{2.3.2}$$

This function constitutes a well defined kernel since the exponential, product and sum of valid kernels results in a valid kernel as well, and here we can decompose it into

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\mathbf{x}_i^T\mathbf{x}_i/2\sigma^2)\exp(\mathbf{x}_i^T\mathbf{x}_j/\sigma^2)\exp(-\mathbf{x}_j^T\mathbf{x}_j/2\sigma^2). \tag{2.3.3}$$

This reconstruction of the original function allow us to see how each factor constitutes a valid kernel with the associated inner product well defined. Moreover, this can be the starting point from which we can transform each of these components to better suit the problem we try to solve in each case. Therefore, we could safely use functions that could potentially be more useful for a given task in problems where originally it would be very difficult. This is specially convenient for cases on which a nonlinear transformation could ease the task at hand, as for example separating different classes in a classification task by introducing a simple nonlinear transformation that makes the decision boundary linear in the new transformed space. This is known as the *kernel trick* (Scholkopf 2001), and is widely employed in approaches such as *Support Vector Machines* and many others.

2.3.2 Inference with Gaussian Process

As we mentioned earlier, GP models represent a principled approach based on kernel methods that can be seen as a convenient and simple manner to conduct inference in the function space (rather than in the parameter space, such as what is being done in NNs). To this end, in order to implement any GP model we will need to make certain assumptions about the shape of the prior over the space of functions themselves, and this will in return allow us to obtain in a closed-form fashion all of the important quantities we may need to train and predict with our model.

In general, GPs are defined as a distribution over functions $f(\cdot)$ so that for any finite $\{\mathbf{x}_i\}_{i=1}^N$, $(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N))$ follows an N-dimensional normal distribution (Williams and Rasmussen 2006). Due to this Gaussian form, there are closed-form solutions for the quantities we are interested in for doing inference, namely the posterior distribution of the model parameters given the data, as well as the predictive distribution. However, as one might expect, although placing this type of prior over the functions allows for the calculations to be made analytically, this also restricts our predictions to follow a Gaussian distribution. Therefore, GPs are only employed when this normality constrain on the prediction is not an important hindrance to the problem at hand. Moreover, we will see there are important concerns about the scalability of GPs, although we will discuss as well one of the most recurrent methods to deal with this issue.

Using this previous definition of GPs, consider a linear model for the training outputs for which we can define the marginal distribution over $\mathbf{y}$ as

$$\mathbf{y} = \mathbf{w}^T \phi(\mathbf{x}), \qquad p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_\theta). \tag{2.3.4}$$

On the first part, we define the weight vector $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbb{I})$ for the N function samples obtained, being α the precision parameter. For the marginal of $\mathbf{y}$, we set the means to zero for convenience (without loss in generality). Finally, the covariance matrix $\mathbf{K}_\theta$ is obtained through $\phi(x)$. We will it element-wise as

$$[\mathbf{K}_\theta]_{i,j} = C(\mathbf{x}_i, \mathbf{x}_j; \theta), \tag{2.3.5}$$

where we define the *covariance function* $C(\cdot, \cdot; \theta)$ as

$$C(\mathbf{x}_i, \mathbf{x}_j; \theta) = \frac{1}{\alpha} \phi(x_i)^T \phi(x_j), \tag{2.3.6}$$

and where we have explicitly included the dependency on θ, a set of parameters that define the mapping $\phi(x)$. This covariance function plays a major role in the following calculations for the inference process with GPs since all the relevant quantities can be obtained once it is defined. In general, we can define the covariance function directly, rather than through the choice of basis functions $\phi(x)$ (in which case, θ will refer solely to the parameters of the selected covariance function itself). If chosen in beforehand, the shape of this function can be used to enclose information already known from the dataset prior to the analysis, *e.g.* periodic behavior, linear trends, etc. To do this, we will simply have to choose the suitable function with these behaviors.

It is when describing the covariance function where the connection to the kernel methods is made apparent: the covariance function plays here the same role that the kernel function does in KMs, and the covariance matrix is the Gram matrix associated to each kernel. Using this idea, we can use all the tools already available for the kernel methods to define valid covariance functions here. This makes GP models pretty flexible in terms of their formulation since performing simple combinations between already valid GPs also provides a new valid GP model that can attend to multiple patterns in the data at the same time. For example, as famously shown in Gelman et al. 2013, the addition of different GPs can be used to model complex patterns, encapsulating all sorts of different seasonal trends as well as short-term variations within the same model by simply using one GP for each type of behavior. This allows also for a flexible construction of models, on which iterations and refinement of individual parts of the global model can be done easily. Moreover, the results provided in each part of the additive model remain somewhat interpretable, which favors further understanding of the different underlying patterns on which the data can be decomposed into. Therefore, when dealing with a complex problems we can use this fact to model separately the different contributions that may constitute the data, and then improve the model in a piece-wise manner.

There are many different covariance functions to choose from, and that choice will be made depending on the type of data and task we are facing in each particular case. Some of the most popular ones are the *Squared Exponential* (SE), the Ornstein–Uhlenbeck (OU), the periodic exponential and the Matérn functions:

$$C_{\mathbf{SE}}(\mathbf{x}_n, \mathbf{x}_m) = \sigma^2 \exp\left(-\frac{(\mathbf{x}_n - \mathbf{x}_m)^2}{2\ell^2}\right), \tag{2.3.7}$$

$$C_{\mathbf{OU}}(\mathbf{x}_n, \mathbf{x}_m) = \sigma^2 \exp\left(-\frac{||\mathbf{x}_n - \mathbf{x}_m||}{\ell}\right), \tag{2.3.8}$$

$$C_{\text{Periodic}}(\mathbf{x}_n, \mathbf{x}_m) = \sigma^2 \exp\left(-\frac{1}{\ell^2} 2\sin^2\left(\frac{\pi||\mathbf{x}_n - \mathbf{x}_m||}{\nu}\right)\right), \tag{2.3.9}$$

$$C_{\text{Matérn}}(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\Gamma(\gamma)2^{\gamma-1}}\left(\frac{\sqrt{2\gamma}}{l} d(\mathbf{x}_n, \mathbf{x}_m)\right)^{\gamma} K_{\gamma}\left(\frac{\sqrt{2\gamma}}{l} d(\mathbf{x}_n, \mathbf{x}_m)\right). \tag{2.3.10}$$

Both the SE and the OU covariance functions have only two parameters: σ^2, which works as an amplitude parameter (scale factor), and ℓ, the length-scale, which controls the smoothness of the functions sampled from the GP. These parameters appear with the same functionality in the periodic kernel on (2.3.9), plus an additional parameter ν which is used to parametrize the typical wavelength of the periodic oscillations (*i.e.* distance between two consecutive maxima). Finally, in the Matérn class of covariance functions is given by (2.3.10), with positive parameters l and *gamma*, where K_γ is a modified Bessel function (Abramowitz and Stegun 1964) and $d(\mathbf{x}_n, \mathbf{x}_m)$ is the Euclidean distance between $\mathbf{x}_n$ and $\mathbf{x}_m$ (Williams and Rasmussen 2006). The Matérn covariance functions can be seen as a generalization over the SE and OU cases, which can be recover as particular cases of it. In general, the flexibility of behaviors available from these covariance functions are key to why GPs are suited for many different tasks, since one can encode a lot of prior information in the behavior of the functions here. It is this aspect which makes additive models of GPs so interesting, since different patterns in the data can be separately modelled using distinct GPs, each one with a suitable ready-made covariance function (Gelman et al. 2013).

The behavior of the covariance function conditions the results provided by the GP. The examples mentioned here are particularly common ones since they allow for modelling patterns present in numerous problems. As an example of this, due to the squared exponential term inside the SE function, the functions sampled using this as covariance function present a smooth behavior (in general), whose features can be tuned according to each problem by changing the parameters (σ^2, l). In order to showcase this change in behavior, in Figure 2.6 we present three different settings and sample functions from the GP prior that uses each one of them: from left to right, we have ($l = 1$, $\sigma^2 = 1$), ($l = 0.2$, $\sigma^2 = 1$), ($l = 1$, $\sigma^2 = 5$). One can easily see that higher σ" values result in a wider spread of the functions around the mean (here set to 0), while higher values of l makes the functions sampled smoother overall. On the other hand, the Ornstein-Uhlenbeck has been used to parametrize stochastic processes (Williams and Rasmussen 2006), while the periodic function is useful when dealing with recurrent behaviors in the data with given seasonality.

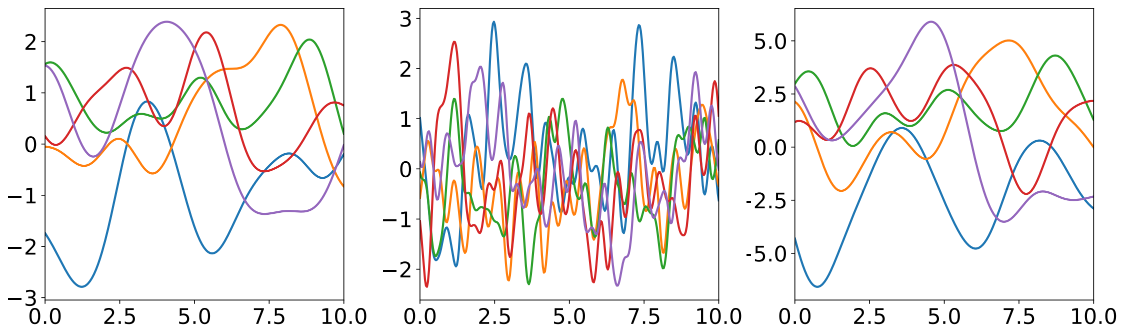


Figure 2.6: Function samples from the SE covariance function prior with different parameters: from left to right, ($l = 1$, $\sigma^2 = 1$), ($l = 0.2$, $\sigma^2 = 1$), ($l = 1$, $\sigma^2 = 5$).

To conduct inference we can also take into account a possible noise contribution to the data. In this case, the target values will be obtain adding this noise to the original output of the GP, that is:

$$y_i = \mathbf{w}^T \phi(x_i) + \epsilon_i = \hat{y}_i + \epsilon_i, \tag{2.3.11}$$

where we assume a linear model with $\mathbf{w} \sim \mathcal{N}(0, \alpha^{-1})$, and $\epsilon_i \sim \mathcal{N}(0, \beta^{-1})$ representing the additive noise, being α and β two different precision parameters (if we want to discard the noise, we simply set $\mathbf{y} = \hat{\mathbf{y}}$). The conditional probability distribution of all the target values $\mathbf{y}$ given the training outputs and will be

$$p(\mathbf{y}|\hat{\mathbf{y}}) = \mathcal{N}(\mathbf{y}|\hat{\mathbf{y}}, \beta^{-1}\mathbb{I}_N), \tag{2.3.12}$$

with $\mathbb{I}_N$ as the identity matrix of dimensions $N \times N$. Combining this expression with (2.3.4) we can obtain the marginal probability of the target values by integrating out the training outputs. This leaves

$$p(\mathbf{y}) = \int p(\mathbf{y}|\hat{\mathbf{y}})p(\hat{\mathbf{y}})d\hat{\mathbf{y}} = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_\theta + \beta^{-1}\mathbb{I}_N). \tag{2.3.13}$$

In the case of not taking into account the additive noise in Eq.(2.3.11), one would simply set $\beta^{-1} = 0$ and remove its contribution to the covariance matrix on the marginal here. Again, the key aspect to this expression is estimating the different components of the covariance matrix $\mathbf{K}_\theta$. For this, we will define the test data as $\{\mathbf{x}_i^\star, \mathbf{y}_i^\star\}_{i=1}^M$ and the train data as $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, being the inputs x's and the output y's in each case. Given the GP prior, the joint distribution for the test and training outputs is going to be defined as

$$p(\mathbf{y}^\star, \mathbf{y}) = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\kappa}_\theta & \mathbf{k}_\theta^{\mathrm{T}} \\ \mathbf{k}_\theta & \mathbf{K}_\theta + \beta^{-1}\mathbb{I} \end{bmatrix} \right). \tag{2.3.14}$$

Here, a new pair of covariance matrices is needed, which will account for the covariances in the test points given the training points observed. These matrices are also obtained from the covariance function, similarly to what we used in (2.3.5), although in this case they will depend on the values of the test input points:

$$[\mathbf{k}_\theta]_{i,j} = C(\mathbf{x}_i, \mathbf{x}^\star_j; \theta)\,, \qquad [\boldsymbol{\kappa}_\theta]_{i,j} = C(\mathbf{x}^\star_i, \mathbf{x}^\star_j; \theta)\,. \tag{2.3.15}$$

Intuitively, $\mathbf{k}_\theta$ holds values estimated from the proximity of the test inputs given the observed training points, while $\boldsymbol{\kappa}_\theta$ represents the covariances estimated with the covariance function inside the test data only.

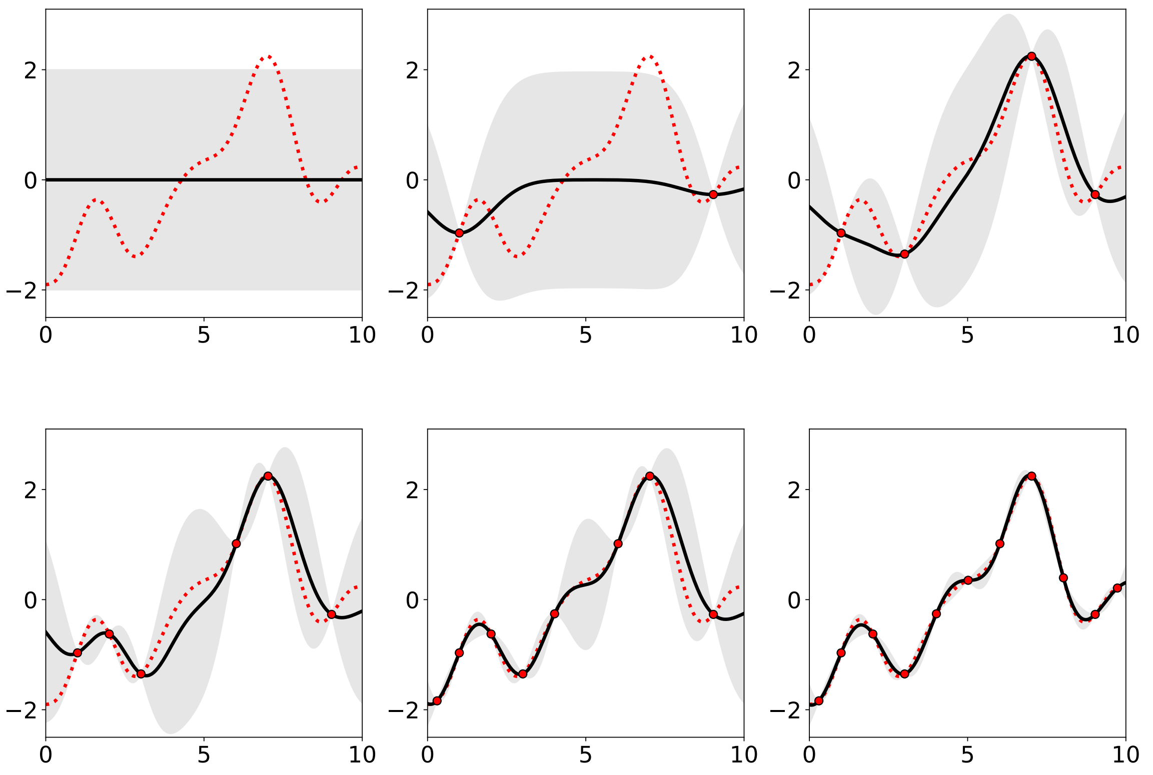


Figure 2.7: Process of fitting a GP prior using an increasing number of points (beginning from the top left, with increasing number of points to the from left to right and from top to bottom). The GP prediction mean is plotted as a black line, the 95% CI as the shaded area and the ground truth as a discontinuous red line. The more points are added, the better the fit of the GP is to the ground truth. Best seen in color.

Given the previous definition for the joint distribution, the predictive distribution will be:

$$p(\mathbf{y}^\star | \mathbf{y}, \mathbf{x}^\star, \mathbf{x}) \sim \mathcal{N}(\mathbf{m}, \boldsymbol{\Sigma}), \tag{2.3.16}$$

with mean and covariances given by

$$\mathbf{m} = \mathbf{k}_\theta^{\mathrm{T}}[\mathbf{K}_\theta + \beta^{-1}\mathbb{I}]^{-1}\mathbf{y}\,, \quad \boldsymbol{\Sigma} = \boldsymbol{\kappa}_\theta - \mathbf{k}_\theta^{\mathrm{T}}[\mathbf{K}_\theta + \beta^{-1}\mathbb{I}]^{-1}\mathbf{k}_\theta\,. \tag{2.3.17}$$

Samples for the predictions can easily be obtained through this expression by evaluating both the mean and covariance functions using a source of Gaussian noise (Williams and Rasmussen 2006).

Finally, we also need to be able to adjust the model's hyperparameters θ according to observed data. This falls inside the framework of *model selection* for GPs, which is much more rich and complex than what we will present here. In our case, we will simply begin by formulating the log of the marginal likelihood, $p(\mathbf{y}|\theta)$, which is given by

$$\log p(\mathbf{y}|\theta, \mathbf{x}) = -\frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{K}_\theta + \beta^{-1}\mathbb{I}| - \frac{1}{2}\mathbf{y}^{\mathrm{T}}(\mathbf{K}_\theta + \beta^{-1}\mathbb{I})^{-1}\mathbf{y}. \tag{2.3.18}$$

The term *marginal likelihood* refers to the marginalization conducted over the function values $\mathbf{f}$, which are defined as $\mathbf{f}|\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_\theta)$. The different terms in this function can be easily interpreted as well: the first contribution is a normalization constant, the second can be seen as a penalty depending only on the covariance functions (*regularization term*) and the third is the only one related to the data fit of the model (Williams and Rasmussen 2006). To fit the hyperparameters according to the data we must obtain the partial derivatives of (2.3.18) w.r.t. θ. Considering $\boldsymbol{\theta}$ as a vector containing all different hyperparameters, these derivatives follow

$$\frac{\partial}{\partial\theta_i}\log p(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^T\mathcal{K}^{-1}\frac{\partial\mathcal{K}}{\partial\theta_i}\mathcal{K}^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}\left(\mathcal{K}^{-1}\frac{\partial\mathcal{K}}{\partial\theta_i}\right) \tag{2.3.19}$$

$$= \frac{1}{2}\mathrm{tr}\left((\boldsymbol{\alpha}\alpha^T - \mathcal{K}^{-1})\frac{\partial\mathcal{K}}{\partial\theta_i}\right), \tag{2.3.20}$$

for each possible hyperparameter θ_i, and where we have defined

$$\mathcal{K} = \mathbf{K}_\theta + \beta^{-1}\mathbb{I} \qquad \boldsymbol{\alpha} = \mathcal{K}^{-1}\mathbf{y}. \tag{2.3.21}$$

Once $\mathcal{K}^{-1}$ is known, the computational cost of evaluating these derivatives is small, so using this approach to optimize the hyperparameters can be a feasible approach in many cases. The maximum value obtained here corresponds to the most likely interpretation of the data from the hyperparameters. Therefore, maximizing 2.3.18 is usually a sensible approach for optimizing θ (Williams and Rasmussen 2006). Other techniques such as *cross validation* can also be useful to the same end (Williams and Rasmussen 2006). This will allow us to optimize the values of the parameters of the model when training. We see an example of fitting a GP to noiseless data ($\beta^{-1} = 0$) in Figure 2.7. In each figure, the GP predictive mean and 95% CI are represented as a black line and a grey-shaded area, while the ground truth for the data is shown as a discontinuous red line. Starting from the top-left image, the GP prior behaves as a standard normal distribution. Introducing observed training points (red dots) and training the GP in those makes the GP interpolate between values (also, the variance in the training points is zero since we have noiseless data for this illustrative example). The more points are added, the better the fit of the GP to the underlying ground-truth for the data. Finishing with the bottom-right image, the GP closely resembles the ground-truth once enough data points are employed.

GPs provide a simple way of obtaining closed-form solutions to the calculations involved in Bayesian inference problems. Due to the Gaussianity assumption, obtaining the predictive distribution and the marginal log-likelihood is feasible by just employing calculations related to multivariate normal distributions. GPs are also non-parametric models, since the only *hyperparameters* involved are the free parameters we choose to describe our covariance functions. The rest of the parameters involved are, in fact, integrated out. Therefore, GP-based models constitute a strong candidate for practicing Bayesian inference thanks to the tractability of all the expressions involved and the fact that their non-parametric formulation may help with complex real-world problems. However, they present two very important drawbacks as consequences:

1. The predictive distribution is exclusively Gaussian.
2. Their scalability is compromised when the training dataset is large.

The first of the two issues is difficult to solve in most cases. Other distributions can be used to set the prior on the space of functions, which gave rise to alternative formulations *s.a. Student-t Processes* (Shah et al. 2014). However, the flexibility of these methods is limited since the calculations need to be estimated analytically. Therefore, GP predictive distribution is mostly restricted to very specific types of distributions.

On the other hand, the scalability issue is related to the inversion of the covariance matrix in (2.3.17) and (2.3.18). The shape of $\mathbf{K}_\theta$ is $N \times N$, being N the number of training points. Therefore, due to the inversion of this matrix, training the GP scales as $\mathcal{O}(N^3)$. This growth rate with N is prohibitive when the dataset is large enough, therefore preventing us from using GPs when there is a large number of training points (where the exact number will depend on other facts such as the dimensionality, sparseness or other data features). However, some proposals have achieved a much higher degree of success here than with the Gaussianity issue. One of the most important additions are *sparse Gaussian processes* (SGPs), which we will introduce briefly here.

2.3.2.1 Connection between Gaussian Processes and Neural Networks

As we mentioned earlier, there is a strong connection between GPs and NNs. Thus far we have seen that both methods provide a good approach when trying to represent very wide ranges of functions, that in turn may be used in many different applications. In the case of GPs, the model is non-parametric (at least in its most basic form), and its flexibility is conditioned only on the number of points available to train the system and the computational requirements to train the system itself. On the other hand, we have seen that the flexibility and performance of NNs greatly depends on the architecture used, both in terms of the number of hidden layers or their width. In general we can say that a NN with a certain number of hidden units N can be considered universal approximators (Hornik et al. 1989), meaning that given a large enough N, the NN can approximate almost any function with arbitrary accuracy. We saw in Section 2.1.3 that, if a NN model is much more complex than what may be needed for the data available it may lead to overfitting (be it because there are few training data points or for any other reason). Therefore, in most practical cases it seems reasonable to set a maximum limit to N, which in each specific case will depend different factors (computational resources, size of training data, time available, etc.). It is when we remove this restriction when the relation between NNs and GPs become apparent, which has been a known result since the late 1990s (Neal 1996).

When setting up a NN, in order to initialize the computations we may set a *prior distribution* over all the NN's parameters $\mathbf{w}$, who are usually also referred as weights (including both the weights and biases with together). One of the main results in Neal 1996 is the fact that, for a wide class of priors over the NN parameters, the resulting output from the NNs would converge to the same output of a GP if the number of units in a layer tends to infinity ($N \to \infty$), which is there shown to hold with one-layer feed-forward fully connected NN. Therefore, we could say that in the limit of infinite width, a NN with one hidden layer is equivalent to a GP. Since this is true, one could also construct the equivalent *kernel* function for the covariances of the outputs, which is shown in Williams and Rasmussen 2006 for two specific choices of the activation function of the hidden units, while this has been extended in later works (Cho 2012). However, one of the main drawbacks of taking the limit of width in multi-output problems (such as multi-class classification or multivariate regression), is the fact that, as in GPs, the $N \to \infty$ one-layer feed-forward NN will have independence among the final output values. This is common in GPs, although it the interdependence of certain components in the NN models which allow fir their improved performance in many cases. This could be seen as if in this limit, each hidden unit does not effectively influence the rest of the units as much as in the finite N case, and therefore these dependencies are lost when $N \to \infty$ (Bishop 2006).

Some later research has been focused on this connection, trying to extract useful models and information on both NNs and GPs through it by combining these concepts (Daniely et al. 2016; Lee et al. 2017). In these cases, the experiments revolve around the construction of models such as GPs equivalent to NNs (Lee et al. 2017) and then comparing the performance of the method against its NN counterpart, quantifying the effects that may have changing certain model parameters in the way. This will allow for a more detailed comprehension between both methods, and hopefully new and better techniques to apply in real-world problems.

2.3.3 Sparse Gaussian Processes

Sparse GPs represent an alternative formulation to regular GPs to try to improve their scalability through the introduction of *pseudo-inputs*, also called *inducing points*. This concept reduces the memory requirements from regular GPs when the number of inducing points (M) is small in comparison with the number of total training points (N, $M \ll N$). This also allows for a much more manageable size to perform the operations that depend on covariance matrix, which was the major hindrance to scalability in (2.3.17, 2.3.18). The number of inducing points is going to become a sensitive parameter for these models, whose performance is going to be strongly related to the selected value for M.

As defined earlier, the random function sampled from a GP will behave such that for any finite dataset $\{\mathbf{x}_i\}_{i=1}^N$ we have that

$$\mathbf{f} = (f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N)), \quad \mathbf{f} \sim \mathrm{GP}(\boldsymbol{\mu}, \mathbf{K}) \tag{2.3.22}$$

where means $\boldsymbol{\mu}$ and covariances $\mathbf{K}$ are defined, for each possible input, as

$$\mu_i = \mathbb{E}[f(\mathbf{x}_i)], \qquad [\mathbf{K}]_{i,j} = \mathbb{E}[(f(\mathbf{x}_i) - \boldsymbol{\mu}_i)(f(\mathbf{x}_j) - \boldsymbol{\mu}_j)]. \tag{2.3.23}$$

Consider now a new pseudo-dataset of size M, with pseudo-data $\{\mathbf{z}_i, \mathbf{u}_i\}_{i=1}^M$, where $\mathbf{z}$ represents the pseudo-inputs and $\mathbf{u}$ the pseudo-outputs (which we will consider noiseless since they are not real observations). This means that we will have

$$\mathbf{u}_j = f(\mathbf{z}_j) \quad \text{with} \quad \mathbf{z}_j \in \{\mathbf{z}_1, \cdots, \mathbf{z}_M\}. \tag{2.3.24}$$

With this, we split the uncountably infinite set of random variables represented in the GP into two separate parts: the finite subset $\{\mathbf{z}_i, \mathbf{u}_i\}_{i=1}^M$, and the remaining uncountable infinite set, which can be denoted as $f(\cdot)$ evaluated for all locations except $\{\mathbf{z}_i\}_{i=1}^M$.

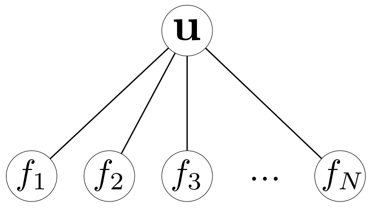


Figure 2.8: Graphical model for the FITC approximation. Each f_i is only dependent from $\mathbf{u}$, and conditioned on it each f_i becomes independent from the others.

Here we will introduce the *Fully Independent Training Conditional* approximation for Sparse GPs (FITC), although in later chapters we will see an alternative description based on *variational inference* (see Section 3.3.1.2). In the FITC approximation, conditional independences are introduced, as the conditional distribution of $\mathbf{f}$ given $\mathbf{u}$ is simply given by

$$p(\mathbf{f}|\mathbf{u}) = \prod_{i=1}^{N} p(f_i|\mathbf{u}), \tag{2.3.25}$$

which is illustrated in the Figure 2.8. This decomposition means that each f_i is independent from other f_j ($j \neq i$) given $\mathbf{u}$, hence the fully-independent of the approximation (Quinonero-Candela and Rasmussen 2005). Given this, the joint distribution of both types of outputs will be

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{u} \end{pmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K_{xx}} & \mathbf{K_{xz}} \\ \mathbf{K_{xz}} & \mathbf{K_{zz}} \end{bmatrix} \right), \tag{2.3.26}$$

where we have fixed the means to zero and the different covariance matrices are defined in terms of the selected covariance function as done in (2.3.5)

$$[\mathbf{K_{zz}}]_{i,j} = C_\theta(\mathbf{z}_i, \mathbf{z}_j), \quad [\mathbf{K_{zx}}]_{i,j} = C_\theta(\mathbf{z}_i, \mathbf{x}_j), \quad [\mathbf{K_{xx}}]_{i,j} = C_\theta(\mathbf{x}_i, \mathbf{x}_j), \tag{2.3.27}$$

with $\mathbf{K_{xz}} = \mathbf{K_{zx}}$ due to the symmetry property of the covariance function we discussed in (2.3.5). It is important to note here that $\mathbf{K_{z,z}}$ has dimensions $M \times M$, while $\mathbf{K_{x,x}}$ is $N \times N$. This difference in shapes is the key reason why the pseudo-input approach is formulated, since the much smaller size of $\mathbf{K_{z,z}}$ will prove to be crucial to reduce the computational requirements needed to employ GPs. Also, $\mathbf{K_{x,z}}$ and $\mathbf{K_{z,x}}$ are not square matrices and therefore do not define proper covariance matrices.

Marginalizing over $\mathbf{u}$ provides a new approximate prior for $\mathbf{f}$

$$p(\mathbf{f}) = \int p(\mathbf{f}|\mathbf{u})p(\mathbf{u})d\mathbf{u} \approx \int \prod_{i=1}^{N} p(f_i|\mathbf{u})p(\mathbf{u})d\mathbf{u} = \mathcal{N}(\mathbf{f}|0, \tilde{\mathbf{K}}_{\mathbf{xx}}) \tag{2.3.28}$$

where the new covariance matrix will be given by

$$\tilde{\mathbf{K}}_{\mathbf{xx}} = \beta^{-1}\mathbb{I} + \text{diag}\ (\mathbf{K_{xx}} - \mathbf{Q_{xx}}) + \mathbf{Q_{xx}}, \qquad \mathbf{Q_{xx}} = \mathbf{K_{xz}}\mathbf{K_{zz}^{-1}}\mathbf{K_{zx}} \tag{2.3.29}$$

being β the noise-precision constant parameter and $\mathbf{Q_{xx}}$ has rank M. Using this approximation we can write the new predicted value at each input location $\mathbf{x}_i$ in terms of the evaluations on the pseudo-input locations, which is given by

$$p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{f}|\mathbf{K_{xz}}\mathbf{K_{zz}^{-1}}\mathbf{u},\ \text{diag}[\mathbf{K_{xx}} - \mathbf{Q_{xx}}]). \tag{2.3.30}$$

To make predictions $\mathbf{f}^*$ at the test input locations $\mathbf{x}^*$, we use the new approximate GP prior, obtaining

$$p(\mathbf{f}, \mathbf{f}^*) = \mathcal{N}\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \tilde{\mathbf{K}}_{\mathbf{xx}} & \mathbf{Q_{xx^*}} \\ \mathbf{Q_{x^*x}} & \mathbf{K_{x^*x^*}} \end{bmatrix}. \right) \tag{2.3.31}$$

where the terms with $\mathbf{x}^*$ underscript are defined in an analogous manner to the original ones but using the values of the test input locations. Conditioning this joint distribution on the observed data we can obtain the final predictions

$$p(\mathbf{f}^*|\mathbf{f}) = \mathcal{N}(\mathbf{f}^*|\mathbf{m}_{\mathbf{f}}^*, \mathbf{\Sigma}_{\boldsymbol{f}}^*) \tag{2.3.32}$$

where the mean and covariance functions are

$$\mathbf{m}_{\mathbf{f}}^* = \mathbf{Q_{x^*x}}\tilde{\mathbf{K}}_{\mathbf{xx}}^{-1}\mathbf{f} \tag{2.3.33}$$

$$\mathbf{\Sigma}_{\boldsymbol{f}}^* = \mathbf{K_{x^*x^*}} - \mathbf{Q_{x^*x}}^T\tilde{\mathbf{K}}_{\mathbf{xx}}^{-1}\mathbf{Q_{x^*x}} \tag{2.3.34}$$

Therefore, all the calculations needed up to this point will, at most, depend on the inverse of $\mathbf{K}_{\mathbf{zz}}$ and the product between matrices. As mentioned earlier, the shape of this covariance matrix is only $M \times M$, where $M \ll N$, and therefore obtaining its inverse is much more manageable than dealing with the inverse of $\mathbf{K}_{\mathbf{xx}}$ that we had in (2.3.17). Using this approximation, thanks to the structure we have employed to decompose $\mathbf{K}_{\mathbf{xx}}$, its inverse and the calculations needed to train and predict with the model will scale with a $\mathcal{O}(NM^2)$ cost. Specifically, the computational cost will be dominated by the calculation of $\mathbf{K}_{\mathbf{zz}}$ and its inverse, which scales as $\mathcal{O}(NM^2)$. Once this term is obtained, the predictive moments can be obtained in the scales of $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ for the mean and covariances respectively. The location of the inducing points can be optimized by computing and maximizing the marginal likelihood, considering them as parameters from the prior and proceeding as we did in (2.3.18) for the rest of the parameters on the exact GP case.

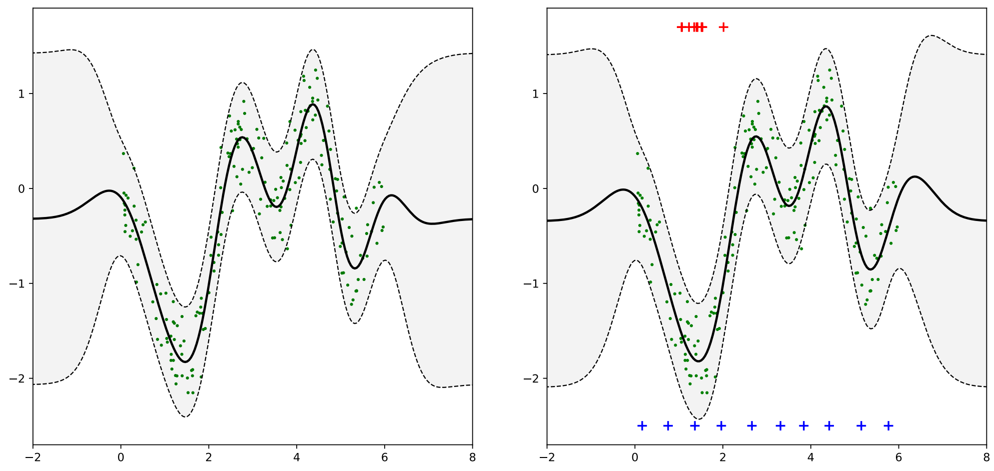


Figure 2.9: Exact GP (*left*) and sparse GP (*right*) fits, with the predictive mean (black line) and $\pm 2\sigma$ region (shaded area inside discontinuous lines). The data used is the same as in (Snelson and Ghahramani 2005), showing the training points as green dots. In the SGP case, the top and bottom crosses mark the starting (*top, in red*) and ending (*bottom, in blue*) position of the 10 inducing points employed. Best seen in color.

In Figure 2.9 we see a comparison between the results of using an exact GP model and a sparse GP approximation. This example is a repetition to what is shown in (Snelson and Ghahramani 2005), using a similar setup and the same dataset they provide. The plots here represent exact GP inference results on the left and SGP on the right, showing in each case the mean of the predictive distribution (black line) and the $\pm 2\sigma$ region as a shaded area (delimited by discontinuous lines), with the training data as green dots. In the case of the SGP we also have the initial location of the 10 inducing points selected, *i.e.* the initial values for $\mathbf{z}$ as red crosses at the top, while the finishing locations are signaled at the bottom with blue crosses (the y value for these markers has no meaning, is just chosen for illustration purposes). We see that the predictive distribution of the SGP model resembles closely that of the exact GP model: the predictive mean of both cases are very similar to each other, and the places on which the predictions differ more are situated in the right-most part of the dataset, where there is a slight variation in their behavior. However, in the SGP case we are using calculations that are far less costly than in the exact case. This is specially significative since in this case since there are only 10 inducing points, and the results are comparable to those obtained by the exact approach that takes into account hundreds of data points (in this case, the training dataset contains 200 instances).

Finally, the inducing points locations can be interpreted as regions on which the GP is focused on while training, *summarizing* the training set into a much smaller number of points. This may arise some concerns regarding the sensitivity of the model to the number of inducing points, and in general we must be aware of the effect that the choice of M makes: in the limit on which M is too small for a given dataset, the SGP model may underperform since it could not be able to recognize all of the important features from the data. On the other hand, if we make $M = N$ and position the inducing points in the locations of the training points, we recover the exact GP model since $\mathbf{K_{zz}} = \mathbf{K_{xx}} = \mathbf{K_{zx}}$. Therefore, we can assume that the model should perform reasonably well somewhere in the range $0 < M < N$.

Chapter 3

Approximate Bayesian Inference

Bayesian statistics provides a powerful framework for extracting, manipulating and combining information from data. The Bayesian approach is a natural fit to many ML approaches where it may be important to exchange regular point-wise predictions with more informative predictive distributions. The probability distributions obtained at the end of the inference process incorporate the uncertainty over different parts of the model, resulting as the combination of prior knowledge with observed data. However, Bayesian inference in ML is intractable for most cases, and therefore practical implementations resort to approximate methods for inference. We will review some of the most relevant methods here, especially focused on the optimization-based and implicit approaches, but also touching on sampling-based methods. Finally, after introducing the concept of α-divergences, we will describe other alternative approximate inference methods that rely on implicit process, a generalization of the GP formulation of the previous chapter that gets rid of the Gaussianity assumptions. This represents a promising line of research, although the methods proposed thus far allow for further improvement in their formulation.

3.1 Bayesian Machine Learning

Thus far we have introduced some of the most popular ML techniques, such as GPs and NNs. In most cases, these are employed to make use of huge amounts of labelled data in supervised learning tasks. Among other features, their performance and adaptability have lead to deep changes in the way ML and AI research is done altogether, specially during the last decade. However, as its usually the case, data can be contaminated by noise, while also new instances that deviate strongly from the training set can lead to worse final predictions. Moreover, there have been important concerns raised about the robustness of algorightms to missing data, outliers and purposefully mislabelled data in an adversarial setting (Naveiro et al. 2019). All of these remarks are important problems that must be faced in order to have better ML tools to apply to any problem we see fit, and one of the main attempts at doing this is obtaining information about the uncertainty of a given method about its output values.

Quantifying the uncertainty of an output for a certain method is a simple measure, but a crucial one at assessing how confidently we could trust the outcome predicted. This uncertainty information has huge value in many cases as well, ranging from medical diagnosis to self driving cars, weather predictions, financial forecasts and many others. Accounting for this uncertainty would inform the user about how *certain* the method is about a given prediction, where the level of uncertainty could be influenced by any source of error, which may come in one of two forms: *aleatoric* uncertainty (also called statistical uncertainty), caused by the stochastic behavior of the experiments and the data collection itself, or *epistemic* uncertainty (also referred to as systematic uncertainty), which is caused by a lack of knowledge about the best model to fit the data. Aleatoric uncertainty is *irreducible*, meaning that it is intrinsic to the data itself and cannot be reduced (unless the experiment or the source of the data is changed), while on the other hand epistemic uncertainty can be reduced by gathering more information on our models and improving them. This has been an important topic of research inside ML, since it would provide much needed information which could be of extended use across many fields of interest (Der Kiureghian and Ditlevsen 2009; Senge et al. 2014; Kendall and Gal 2017).

The Bayesian approach for the interpretation of probabilities provides a framework that can be used for the purpose of representing, manipulating and computing uncertainty in a predictive model. This perspective is especially well suited to ML due to its core components, which differ from the standard frequentist view where the probabilities are seen as simply rates of occurrence of random events. From the Bayesian point of view, however, probability describes a *degree of belief* in events (MacKay 2003; Bishop 2006; Gelman et al. 2013). This means that Bayesian statistics is *subjective* in the sense that final beliefs are obtained by updating previous beliefs through the observation of data, which may conform to that set of previous beliefs or not. These beliefs are expressed in terms of *probability distributions* over the possible outcomes. The Bayesian learning approach consists therefore in encoding our prior set of beliefs in a *prior distribution*, and then updating it according to the data observed through the *Bayes theorem* to obtain and updated set of beliefs. This process is usually referred to as *Bayesian inference* (Bishop 2006; Gelman et al. 2013).

In the context of supervised learning, the Bayesian approach functions as a way of obtaining probability distributions about the quantities of interest and then combining all of those probability distributions in our final predictions, which will account for all the sources of uncertainty we will have introduced in our model to that point. This procedure begins by setting a prior distribution over our model's parameters and updating this distribution by the observation of labelled data. This will conform a posterior distribution on our model's parameters, that we can then use to infer the probability distribution of the output of the model once it is given an unlabelled data instance (testing phase) using Bayes' theorem (Bishop 2006). To describe it in more detail, the complete process of Bayesian learning can be divided into three parts (Gelman et al. 2013):

i. Setting up a complete probability model, where a prior distribution over all components in the model (both observed and unobserved) needs to be specified. It is in this part of the process where we should encode our previous information on the system, as well as any useful information that may be considered relevant to the problem at hand. Usually, the more information that can be incorporated in this part of the process, the better. If little is known about the behavior of the data, the experiment or the expected outcomes, one should also reflect this by establishing less restrictive distributions (*e.g. uninformative* priors).

ii. Conditioning the previous information on the observed data, updating the prior distributions set in the previous step according to the observed data points that serve as training for our model. This will output a posterior distribution that contains a mixture between our previous beliefs and the data instances observed, and therefore will provide us an updated version of our assumptions.

iii. Using our updated set of beliefs, encoded in the posterior distribution, to make predictions and evaluate the fit of our model to the data being presented to it. Here we will need to assess how good is the model fitting the data, how reasonable may the resulting predictive distributions be given our previous information, how much do these differ from the initial set of assumptions we made at the beginning, etc.

Depending on the results obtained in the final part of the process, one could modify and alter some of the assumptions made in previous parts of the modelling process to obtain better outcomes in the final part. The whole idea of this process is, for the most part, to be recursive and employ it as many times as needed until a satisfactory model is obtained for the data presented. The main value behind this approach is that the Bayesian framework provides a powerful approach to account for the uncertainty sources to the model using the tools provided by research on probability theory. Inside current research fields, Bayesian modelling and its adaptability to different real-world issues is an important topic that mixes knowledge from statistics, computer science and data analysis. The combination of Bayesian inference with Bayesian data analysis has lead to the formulation of a complete Bayesian workflow that attempts at presenting a systematic and comprehensive way to go about modelling complex datasets, performing model selection, improving our comprehension of the data and being able to assess the quality of model's predictions, among many other features (Gelman et al. 2020).

Due to the formulation of the approach itself, ML techniques based on Bayesian inference count with several advantages that make this a very attractive approach in most cases. As an example, these techniques are more robust to overfitting and odd behavior in the data such as missing values or outliers, since the distributions themselves will reflect the information needed in most cases to avoid these issues.

Moreover, this approach allows for a simple manner to quantify the aleatoric uncertainty in our data while also being useful for model selection, since now we will have available measures to establish how confidently a model is fitting the data available. Also in this regard, encoding information in the prior distribution can help us get the most information on cases where there are not many data points to train the model, since constructing an adequate prior will allow us to make the most out of cases with small data samples. However, all of these good properties are mostly dependent on the usage of Bayes' theorem to perform inference, and in many cases the calculations needed are intractable (Gelman et al. 2013). This is a common problem, and therefore *approximate techniques* have to be employed. In this chapter we will focus on all of these concepts, from basic Bayesian formulation to some of the most relevant approximate inference techniques that are at the core of the main contributions of this thesis.

3.2 Bayesian Learning

Bayesian statistics is built from the Bayes theorem, which allows us to combine prior information and observations into a posterior distribution. To formalize the Bayesian approach, let us begin by assuming we have a given dataset we denote by $\mathbf{X}, \mathbf{Y}$, where $\mathbf{X}$ will represent the input features and $\mathbf{Y}$ the resulting output. Using θ to describe our model's parameters, Bayes theorem can be written as

$$p(\theta|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta)}{p(\mathbf{Y}|\mathbf{X})}. \tag{3.2.1}$$

The different factors in this expression convey meaning about our experiment and the expected results from our model, and therefore we will briefly introduce them separately. In the numerator of the right side of the equation we have $p(\theta)$ and $p(\mathbf{Y}|\mathbf{X}, \theta)$, where the first one corresponds to the prior probability distribution over the parameters in our model, and the second one is referred to as the *likelihood* term. This likelihood is a conditional probability distribution that informs about the probability that, given an input $\mathbf{X}$ and model's parameters θ, our model provides the output $\mathbf{Y}$. On the denominator of this part of the equation we have $p(\mathbf{Y}|\mathbf{X})$, which is usually called the *model evidence* (Bishop 2006). This factor does not depend on θ, and therefore many times is seen as a constant (and one could neglect it in some cases where it is not needed). The model evidence is actually the result of integrating the product of the likelihood and the prior in θ, and therefore can be seen as a normalizing constant that makes the whole right-hand-side of the equation behave as a proper probability distribution. Finally, on the left side of the equation we have the *posterior probability*, that is, the probability that given data $\mathbf{X}, \mathbf{Y}$, our model's parameters have the value θ. This distribution includes the parameter values more probable given our observed data, and therefore can be seen as the updated version of the prior distribution we had to set over them firstly.

To make inference we need to be able to obtain the probability distribution of new data instances once we have obtained the best possible values for θ in our model (given the training data). If the new data point is $\mathbf{x}^*$, we can obtain the estimated output by performing the following integral:

$$p(\hat{\mathbf{y}}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\hat{\mathbf{y}}^*|\mathbf{x}^*, \theta)p(\theta|\mathbf{X}, \mathbf{Y})d\theta, \tag{3.2.2}$$

where θ is being integrated on and thus dissapears from the final predictive distribution, which is called *marginalizing* over θ. The first term in the integral is the likelihood model applied to the new data instance, *i.e.* the probability distribution of obtaining output $\hat{\mathbf{y}}^*$ given input $\mathbf{x}^*$ and a set of model's parameters θ. The second integrand is directly the posterior distribution we obtained in (3.2.1). Finally, the distribution obtained informs us about the probability of observing the estimated output $\hat{\mathbf{y}}^*$ given the training data and the new data instance. The integration over the model's parameters is the part of the inference process that allow us to account for the uncertainty in the parameters and other uncertainty sources in the model, since this some of these uncertainties will be included in our posterior distribution $p(\theta|\mathbf{X}, \mathbf{Y})$. The combination of all these different parts to obtain predictions is the process is known as Bayesian inference.

Finally, another important aspect about (3.2.1) is the model evidence term. As we mentioned, this is given by integrating the numerator of the Bayes theorem over the parameters θ, that is

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta)d\theta. \tag{3.2.3}$$

Since this integral means marginalizing over θ, this term also receives the name of *marginal likelihood*. The marginalisation can be performed exactly in simple cases, for which the likelihood is the *conjugate* to the prior, which means that the posterior distribution will be in the same probability distribution family as the prior distribution (Gelman et al. 2013). However, in most relevant cases, this marginalisation calculations cannot be conducted analytically, and thus we must resort to approximate techniques that allow us to obtain an estimated value to complete the inference process. These approximate techniques do not necessarily mean obtaining an approximate value for the model evidence itself, but could rather give us a distribution that approximates the posterior or some other part of the model as well. Statistical inference techniques for approximating the likelihood represent one of the main research interests in this topic, and work is still being done to further improve the current performance level. However, before we introduce these techniques we will briefly discuss the construction of a model of key importance for the thesis: Bayesian Neural Networks.

3.2.1 Bayesian Neural Networks and Approximate Inference

As we have discussed earlier, the common models employed in DL are not able to provide detailed information about their uncertainties when providing an output. This is highly valuable information, that in some problems may be crucial for making decisions and incorporating new information into the model. Bayesian NNs (BNNs) offered a way of introducing a probabilistic interpretation to the DNNs parameters by considering distributions over the network weights and biases. The simple fact of introducing distributions to model these parameters lead to an increasing interest on extending NNs with a Bayesian perspective, one which could allow us to benefit from all of the power of Bayesian statistics. These types of models would be more robust to overfitting, could take into account uncertainties in different parts of the model, and, more importantly, give uncertainty estimations on the outputs of the network. Also, they could potentially be trained on fewer data points than regular NNs due to the usage of prior information. Therefore, this extension to NNs could expand many of their already impressive features with tools that had been widely studied and developed inside the statistics research.

The idea of mixing Bayesian statistics and ML is not something new. As an example, the first instance of combining both topics in the context of neural networks took place more than thirty years ago by Denker et al. 1987. In that work, Denker *et. al.* proposed replacing the point-like weights in the networks with distributions over the space of weights, therefore turning them into random variables and, for their case, setting a uniform distribution on a compact space as priors for them. Denoting all the inputs as $\{\mathbf{x_1}, ..., \mathbf{x_N}\}$ they mapped each weight configuration to its correspondent output $\{\mathbf{\hat{y}_1}, ..., \mathbf{\hat{y}_N}\}$, which allowed them to obtain a marginal probability for each pair $(\mathbf{x}_i, \mathbf{\hat{y}}_i)$. However, the first introduction of BNNs can be tracked to 1989 (Tishby et al. 1989), which showed that inference could be performed theoretically through the use of Bayes' theorem. Early work at the end of the 1980s and the early 1990s suggested different approaches to obtain approximated posterior distributions over the BNNs parameters (Tishby et al. 1989; MacKay 1992). The first attempt at using backpropagation to find values for the weights was introduced by LeCun et al. 1989. The mode for the parameters' distribution was found and then a Gaussian was fitted to that mode, the width of which would be given by the Hessian of the likelihood. However, this technique was limited to small datasets since it involved the inversion of a Hessian matrix, and therefore would scale poorly if the dataset is big enough. Further on, thanks to an array of different experiments, MacKay 1992 showed important properties about the *model evidence*. They characterized the correlation between model evidence and the generalization error, which meant that in some cases this factor could be used to conduct model selection and choose the model's size. Moreover, they showed that misspecification of the models could lead to situations on which that correlation is broken and model evidence is not indicative of the model generalization, which could mean that low probability could be given to models unjustifiably.

In the early 1990s, Hinton and Van Camp 1993 suggested the use of minimum description length as a way for conducting model regularization and penalize accumulating high amounts of information in few of the network's weights (see Section 2.1.3). They also showed that, for the case of a single hidden layer NN, their objective could be computed analytically, which is also strongly related to the connection between NNs and GPs. In this context, however, it can be seen as one of the first instances of approximating the parameters' distributions through *variational inference* (VI), an important set of techniques that are a major part of the research in approximate Bayesian inference and that we will introduce during this chapter as well. Other approximate inference techniques have also been implemented in this context. This includes using *Markov Chain Monte Carlo* (MCMC) and *Hamilton Monte Carlo* (HMC, also called *Hybrid Monte Carlo*) for BNNs, as proposed by Neal 2012. These approaches for approximate inference do not rely on prior assumptions about the form of the posterior distribution, which as we will see is a core concept for VI-based approaches. In addition to this, Neal 2012 also tried to reproduce some of the results of the Laplace's approximations by MacKay 1992, and showed that in the limit of a very high number of units, the model would converge to various stable processes which would depend on the prior specified (see Section 2.3.2.1).

As a final remark, Barber and Bishop 1998 extended the formulation introduced by Hinton and Van Camp 1993 using a fully VI interpretation and employing full covariance matrices. They highlighted that the obtained objective forms a lower bound to the model evidence, and also performed VI with free-form variational distributions over the hyper-parameters. From this point forward it was soundly established that approximate techniques were one of the most important approaches to deal with the intractability problems that arise when formulating Bayesian inference in ML (not restricted solely to BNNs). This way we should be able to obtain an useful distribution which we can employ to perform our desired regression or classification tasks while conserving the information about the system and its uncertainty. Techniques such as these are a core concept of our work, which is mostly centered on being able to provide better approximations to the distributions needed in Bayesian ML. Ideally, this would provide more flexible and versatile predictive distributions, while also updating the priors in a sensible manner according to the data present. This would also maintain all of the important properties of the Bayesian approach, which are broadly sought-for in ML research. Since this topic is of crucial importance for this thesis, we will introduce here some of the most resorted techniques for approximate inference, including both *optimization-based* and *sampling based* approaches.

3.3 Optimization Based Methods

The main contributions of this thesis revolve around concepts derived from optimization-based methods for approximate inference. These approaches consist on obtaining a new distribution that approximates the target as closely as possible, but that is also more manageable and that allows us to successfully obtain what we may need to conduct inference. Optimization-based methods can also be referred to as *deterministic* in most cases, since given a target distribution and a set of characteristics of the approximating distribution, there is a fixed set of values for the approximation's parameters so that it resembles as closely as possible the objective distribution (albeit in some cases finding that set of values is a complicated task, and in some cases we only arrive to a close estimation of it). We will review some of the most popular methods, although there are many other to choose from depending on what we may be interested on given a particular task.

3.3.1 Variational Inference

Variational inference (VI) is one of the most common approximations for the posterior distributions in modern research in approximate inference. The key idea behind VI is that, instead of calculating the true posterior distribution for the parameters in the model, we will define an approximate *variational* distribution $q_\theta(\mathbf{z})$, parametrized by θ, and whose structure is easy to evaluate. q_θ is usually set to belong to a certain family of distributions, *e.g.* the exponential family, and in some of these cases the computations involved will have analytical solutions. Once the family of distributions is selected, we will make sure that this variational distribution approximates $p(\mathbf{z}|\mathbf{X}, \mathbf{Y})$ closely. To that end, we perform the following decomposition of the logarithm of the marginal likelihood of the probability of the observed data

$$\log p(\mathcal{D}) = \mathcal{L}(q) + \mathrm{KL}(q|p), \tag{3.3.1}$$

where q represents the parametric distribution we are looking to use as approximation for $p(\theta|\mathcal{D})$, $\mathcal{D}$ represents the observed data $(\mathbf{X}, \mathbf{Y})$ and where we have defined

$$\mathcal{L}(q) = \int q_\theta(\mathbf{z}) \log\left(\frac{p(\mathcal{D}, \mathbf{z})}{q_\theta(\mathbf{z})}\right) d\mathbf{z}, \tag{3.3.2}$$

$$\mathrm{KL}(q|p) = \int q_\theta(\mathbf{z}) \log\left(\frac{q_\theta(\mathbf{z})}{p(\mathbf{z}|\mathcal{D})}\right) d\mathbf{z}. \tag{3.3.3}$$

The second term defined here, $\mathrm{KL}(q|p)$, is the Kullback-Leibler (KL) divergence between the distributions $q_\theta(\mathbf{z})$ and $p(\mathbf{z}|\mathcal{D})$. The KL divergence is not a proper distance measure since it lacks the symmetry condition, although it allows us to quantify how dissimilar both distributions are: it is only 0 when both of distributions are the same, and takes values ≥ 0 otherwise. From this fact, it follows from (3.3.1) that $\mathcal{L}(q)$ is a lower bound of $\log p(\mathcal{D})$, since $\log p(\mathcal{D}) \geq \mathcal{L}(q)$. Moreover, since $\log p(\mathcal{D})$ is independent of the choice of approximation q, we will have that minimizing the KL term is the same as maximizing this lower bound, since both of them have to add up to the constant value given by $\log p(\mathcal{D})$, as illustrated in Figure 3.1.

The decomposition performed in (3.3.1) can be done for any possible approximating distribution q. The same holds for the rest of the arguments we have used thus far, namely the fact that choosing the best approximating distribution q is equivalent to minimizing the KL divergence. In practice, this is done by maximizing the lower bound term of (3.3.2) ($\mathcal{L}(q)$), also named the *Evidence Lower Bound* (ELBO). Using the parametric expression for q, this can be written in terms of θ as

$$\mathcal{L}(\theta) = \int q_\theta(\mathbf{z}) \log p(\mathbf{Y}|\mathbf{X}, \mathbf{z}) d\mathbf{z} - \mathrm{KL}(q_\theta(\mathbf{z})|p(\mathbf{z})) \tag{3.3.4}$$

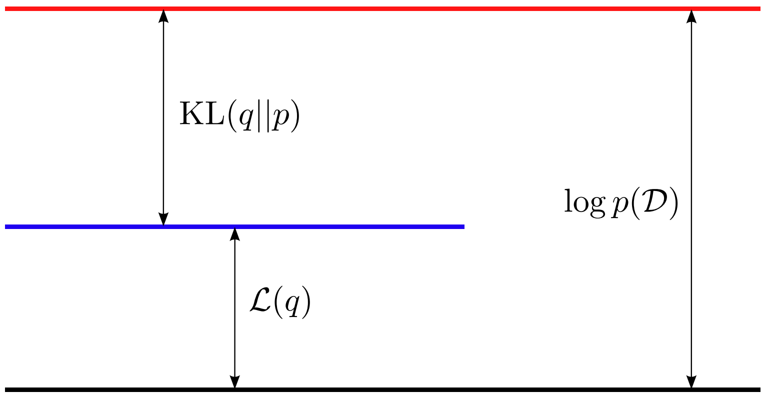


Figure 3.1: Decomposition of the $\log p(\mathcal{D})$ in terms of the $\mathcal{L}(q)$ and $\mathrm{KL}(q|p)$, where the sum of the latter two terms provide the value for the first term. Adapted from Bishop 2006.

which is the objective function to which we will refer here. The first term is usually referred to as the *expected log-likelihood* of the ELBO, while the second term ensures that our approximate distribution does not stray too far away from the prior set on the model since maximizing the ELBO will imply minimizing this term (and thus can be said to work like a regularization term for the approximation). Maximizing this function with respect to θ is what commonly is called as *variational inference*, replacing the calculation of the Bayesian model marginalizations with optimization problems. This formalism allows us to preserve many of the advantages of Bayesian modelling, capturing the model uncertainty in the output, while also providing an easy solution to the approximation problem. Inside this framework, once we have obtained the minimum of the KL divergence by optimizing the parameters in our approximating distribution, we can also write the approximate predictive distribution as

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) \approx \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{z}) q^*_\theta(\mathbf{z}) d\mathbf{z} \equiv q^*_\theta(\mathbf{y}^*|\mathbf{x}^*). \tag{3.3.5}$$

Therefore, once the approximation is optimized, the rest of the calculations can be conducted in an easy manner.

The usage of a parametric approximating distribution is at the same time, one of the most important features of VI and one of its main drawbacks as well. The choice to have q restricted inside a parametric family of distributions allows for most of the calculations to have closed-form solutions. The exact shape of the resulting distribution will depend exactly on what type of restrictions we impose over q. However, since q is forced to belong to the selected family, in some cases this can incur in a bad fit to the original distribution due to the limited flexibility of the family itself. A good example for this is performing VI with the exponential family of distribution in a case where the target distribution has more than one mode. Since the distributions to choose from will be unimodal, depending on how we treat the problem, the resulting approximate distribution can have its mean among the modes of the original distribution (where the probability allocated by the latter is rather low), or it can result in a mode-selection model, where the approximation focuses on one mode and forgets about the others. In general, since VI makes use of the KL divergence in the shape of (3.3.3), the mode-selecting behavior will be more common (Bishop 2006), whereas an alternative use of this divergence can show a *mode-covering* behavior (for more information, see Section 3.5).

For VI techniques, some modern approaches follows Hinton and Van Camp 1993, where they assume a fully factorized approximation to the real posterior distribution and performing optimization then in what has been named *variational mean field* (Jaakkola 2001)

$$q_\theta(\mathbf{z}) = \prod_{i=1}^{K} q_\theta(\mathbf{z}_i), \tag{3.3.6}$$

where $\mathbf{z}$ are assumed to factorize into K disjoint groups of parameters. In the context of NNs, if $q_\theta(\mathbf{z}) = q_\theta(\mathbf{w})$, where now the parameters for the model are represented as $\mathbf{w}$, referring to both the weights and biases. The approximation here is defined to factorize over the weights for each unit in each layer, which could be expressed as

$$q_\theta(\mathbf{w}) = \prod_{i=1}^{L} q_\theta(\mathbf{W}_i) = \prod_{(i,j,k)=1}^{(L,K_i,K_{i+1})} q_{m_{ijk}\sigma_{ijk}}(w_{ijk}) = \prod_{(i,j,k)=(1,1,1)}^{(L,K_i,K_{i+1})} \mathcal{N}(w_{ijk}|m_{ijk}, \sigma^2_{ijk}) \tag{3.3.7}$$

for L data inputs, being K_i and K_{i+1} the number of units in layers i and $i+1$ respectively, m_{ijk} the mean and σ^2_{ijk} the variance for each of the w_{ijk}, which have themselves the shape of a normal distribution with those parameters. Doing optimization in this problem is challenging, and therefore the early attempts only demonstrated VI could be implemented in NNs with a single hidden layer, for which the result has a closed-form expression under certain restrictions regarding the output units (Hinton and Van Camp 1993). However, this type of model is expected to have issues in its performance, since assuming the factorization for the approximating distribution makes that important information about the model's parameters correlations is lost. This can be accounted for with *ad-hoc* transformations, although that also comes with important increases in the computational cost for the method (Louizos and Welling 2017b). This is impractical for modern standards of ML, so both in the context of DL and in the rest of ML VI had to be developed further at this point.

In Graves 2011 a partial sollution to the problems above is presented through the use of *data sub-sampling techniques* in a fully factorized VI objective, allowing VI to scale to large amounts of data. Graves *et. al.* use Monte Carlo estimates for the intractable terms that appear in the expected log-likelihood (Hoffman et al. 2013). This allowed them to apply this approach to models with more parameters, such as NNs with more than one-hidden layer. Therefore, this constitutes the first instance in which a VI approximation technique was scalable for complex models and big data cases. Nevertheless, this model performed badly in practice, and it was not extended until Blundell et al. 2015b proposed re-parametrizing the expected log likelihood Monte Carlo estimates. The work by Blundell et al. 2015b is done inside the context of BNNs, where they further change the model by imposing a mixture of

Gaussian priors over each network weight and then optimizing the parameters of the mixture components. This has a much better performance than previous models, but it is again at the expense of an increase on the computational cost: the use of Gaussian distributions doubles the number of parameters in the model, since now we need to define a Gaussian distribution over each one of them. Although effective, this change would make these models complex to fit.

Most current approaches to VI are based in the work Hoffman et al. 2013, where *mini-batch optimization* for VI was introduced. This would allow to rewrite the VI objective as

$$\hat{\mathcal{L}}(\theta) \equiv -\frac{N}{M}\sum_{i\in S}\int q_\theta(\mathbf{z})\log p(\mathbf{y_i}|\mathbf{f^z}(\mathbf{x_i}))d\mathbf{z} + KL(q_\theta(\mathbf{z}|p(\mathbf{z})), \tag{3.3.8}$$

being S a random index set of size M. The data sub-sampling approximation forms an *unbiased stochastic estimator*, which implies that

$$\mathbb{E}_S[\hat{\mathcal{L}}_{VI}(\theta)] = \mathcal{L}_{VI}(\theta). \tag{3.3.9}$$

Therefore we can use an stochastic optimizer to optimize this stochastic objective and obtain a local optimum set of parameters θ^*, which would in turn be also an optimum for $\mathcal{L}_{VI}(\theta)$. The gradients to train the system can also be attained in a similar fashion, since usually noisy gradients are employed (). The remaining difficulty is then the evaluation of the expected log likelihood term, which is done through Monte Carlo integration and for which several MC estimators have been built, depending on the purpose. We will introduce some of the most relevant here.

3.3.1.1 Monte Carlo Estimators for VI

There are many ways to perform the MC estimation in VI. To simplify the notation, we will introduce these considering the general case of estimating the *integral derivative*

$$I(\theta) = \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx \tag{3.3.10}$$

whose form is analogous to the optimization of the expected log likelihood term in the ELBO objective (3.3.4). In this analogy, $f(x)$ will be a function defined on $\mathbb{R}$ and $p_\theta(x)$ a probability density function parametrized by θ. As a concrete example, if $p_\theta(x) = \mathcal{N}(x;\mu,\sigma^2)$ with $\theta = \{\mu,\theta\}$, when we differentiate Eq.(3.3.10) with respect to $\theta = \mu$ we will refer to the estimator as the *mean derivative estimator*, while if $\theta = \sigma$ we will refer to it as the *standard deviation derivative estimator.*

There are many types of MC estimators for Eq.(3.3.10), of which we will only introduce two:

1. The *score function estimator* (Paisley et al. 2012), which relies on the following identity

$$\begin{aligned}\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx &= \int f(x)\frac{\partial}{\partial\theta}p_\theta(x)dx \\ &= \int f(x)\frac{\partial\log p_\theta(x)}{\partial\theta}p_\theta(x)dx.\end{aligned} \tag{3.3.11}$$

This leads to

$$\begin{aligned}\hat{I}_1(\theta) &= f(x)\frac{\partial \log p_\theta(x)}{\partial\theta} \\ \mathbb{E}_{p_\theta(x)}[\hat{I}_1(\theta)] &= I(\theta).\end{aligned} \tag{3.3.12}$$

2. The *pathwise derivative estimator*, otherwise known as the *reparametrization trick* (described in Glasserman 2013, among others). For this, assume that $p_\theta(x)$ can be reparametrized as a parameter-free distribution, $p(\epsilon)$, separating the stochastic component from the parameters of the model such that $x = g(\theta, \epsilon)$, being $g(\cdot,\cdot)$ a deterministic differentiable bivariate distribution (for example, for $p_\theta(x) = \mathcal{N}(x; \mu, \sigma^2)$, $g(\theta, \epsilon) = \mu + \sigma\epsilon$ and $p(\epsilon) = \mathcal{N}(\epsilon; 0, I)$). The estimator in this case will be

$$\hat{I}_2(\theta) = f'(g(\theta,\epsilon))\frac{\partial}{\partial\theta}g(\theta,\epsilon), \tag{3.3.13}$$

with $\mathbb{E}_{p(\epsilon)}[\hat{I}_2(\theta)] = I(\theta)$, being $f'(\cdot) = \partial f(\cdot)/\partial\theta$.

The usage of these techniques is not exclusive to BNNs, although extensive work has been conducted into applying them to these models. As an example for this, in (Gal 2016) they use the results of Graves 2011 with the pathwise derivative estimator, which we can employ non-Gaussian approximating distributions, trying to achieve a more flexible model. Furthermore, Gal 2016 factorize the distribution for the weights for each row $\mathbf{w_{l,i}}$ in each $\mathbf{W}_l$ rather than over each weight alone. This is done hoping this decomposition behaves better at preserving the correlations between different parameters in the model. Here, a re-parametrization for each $q_{\theta_{l,i}}(\mathbf{w}_{l,i})$ as $\mathbf{w}_{l,i} = g(\theta_{l,i}, \epsilon_{l,i})$ is needed, specifying some distribution over the random variable $\boldsymbol{\epsilon}$ given by $p(\epsilon_{l,i})$. Reparametrizing Eq. (3.3.8) with respect to $p(\boldsymbol{\epsilon}) = \prod_{l,i} p(\epsilon_{l,i})$ and replacing each expected log likelihood term with the corresponding pathwise estimator results in a new MC estimator

$$\mathcal{L}_{MC}(\theta) = -\frac{N}{M}\sum_{i\in S}\log p(\mathbf{y_i}|\mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x_i})) + KL(q_\theta(\mathbf{w})|p(\mathbf{w})) \tag{3.3.14}$$

where $\mathbf{w} = g(\theta, \boldsymbol{\epsilon})$, such that $\mathbb{E}_{S,\boldsymbol{\epsilon}}[\hat{\mathcal{L}}_{MC}(\theta)] = Ł_{VI}(\theta)$. Therefore, optimizing $\hat{\mathcal{L}}_{MC}(\theta)$ with respect to θ will converge to the same optimum as optimizing the original objective, $\mathcal{L}_{VI}(\theta)$. This is what is used in (Gal 2016), by calculating the stochastic derivative estimator with respect to θ, $\hat{\Delta\theta}$ and then updating the parameters by doing $\theta \leftarrow \theta + \eta\hat{\Delta\theta}$ for some learning rate η until a previously set convergence criteria is met.

Using the previous calculations, the actual posterior distribution on the weights can be approximated by the optimized approximate posterior. This allows also to approximate the predictive distribution from Eq.3.3.5 with MC integration as well

$$\tilde{q}_\theta(\mathbf{y}^*|\mathbf{x}^*) \equiv \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \hat{\mathbf{w}}_\mathbf{t}) \approx p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}), \tag{3.3.15}$$

where $\hat{\mathbf{w}}_t \sim q_\theta(\mathbf{w})$. All of these procedures have been developed and researched extensively in the literature (Gal and Ghahramani 2015; Gal 2016; Gal and Ghahramani 2016).

As a final note regarding VI, it is relevant here to mention that it has a close connection to *Stochastic Regularization Techniques* such as dropout. As mentioned in Section 2.1.3, under certain circumstances, VI can be said to provide the same results as dropout. This is true when certain restrictions for $q_\theta(\mathbf{w})$ and $p(\mathbf{w})$ are in place, as shown by Gal 2016.

3.3.1.2 VFE Approximation for Sparse GPs

As we mentioned in the previous chapter, one of the possible solutions to the scalability problems of GPs is the introduction of a set of new parameters, namely the *inducing points*, which lead to the calculations needed being performed in lower-ranked matrices. Sparse GPs allowed for an important increase in the scalability of GPs since the complexity of the model now followed, at most, an $\mathcal{O}(M^2N)$, where M is the number of inducing points, N the number of data points, and usually $M \ll N$.

In Section 2.3.3 we introduced the FITC approximation for sparse GPs. This assumed that the conditional distribution of the functions sampled from the GP and evaluated in each training point, (f_i) were fully independent between them given the evaluation of these functions on the locations of inducing points $(\mathbf{u})$. This allowed for a decomposition between terms that allowed for a simple solution to the sparse GP approach by readjusting slightly the prior model from the one proposed initially. However, now that we have introduced VI, it can be useful for later discussion to introduce another type of approximation for sparse GPs: the *variational free energy* approach (VFE) (Titsias 2009).

Using the same notation as in Section 2.3.3, consider the following decomposition of the log-likelihood

$$\log p(\mathbf{y}|\theta) = \log \int p(\mathbf{y}, \mathbf{f}, \mathbf{u}|\theta) d\mathbf{f} d\mathbf{u}, \tag{3.3.16}$$

where θ represents the parameters in our model. We will now introduce a joint distribution $q(\mathbf{f}, \mathbf{u})$ as an approximating distribution to the true posterior $p(\mathbf{f}, \mathbf{u}|\mathbf{y})$. We write the log-likelihood then as

$$\begin{aligned}\log p(\mathbf{y}|\theta) &= \log \int p(\mathbf{y}, \mathbf{f}, \mathbf{u}|\theta) \frac{q(\mathbf{f}, \mathbf{u})}{q(\mathbf{f}, \mathbf{u})} d\mathbf{f} d\mathbf{u} \\ &\geq \int q(\mathbf{f}, \mathbf{u}) \log \left(\frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u}|\theta)}{q(\mathbf{f}, \mathbf{u})} \right) d\mathbf{f} d\mathbf{u} \equiv \mathcal{L}(q, \theta),\end{aligned} \tag{3.3.17}$$

where $\mathcal{L}(q, \theta)$ is an ELBO function such as the one described in (3.3.4)

$$\log p(\mathbf{y}|\theta) = \mathcal{L}(q, \theta) + \mathrm{KL}(q(\mathbf{f}, \mathbf{u})|p(\mathbf{f}, \mathbf{u}|\mathbf{y})). \tag{3.3.18}$$

By optimizing this distribution, we will make that q resembles the exact posterior as closely as possible, which is the same objective of the original VI formulation. However, here we will not use the same type of parametric solution that is often employed at this point. Rather, consider the following decomposition for the approximate distribution:

$$q(\mathbf{f},\mathbf{u}) = p(\mathbf{f}|\mathbf{u})q(\mathbf{u}) = p(\mathbf{f}|\mathbf{u})\mathcal{N}(\mathbf{u}|\mathbf{m}.\mathbf{\Sigma}) \tag{3.3.19}$$

Here, the $q(\mathbf{u})$ distribution is being approximated by a multivariate normal distribution with means $\mathbf{m}$ and covariance matrix $\mathbf{\Sigma}$, and the conditional $p(\mathbf{f}|\mathbf{u})$ is kept fixed depending on the value of $\mathbf{u}$. This approximate solution makes $q(\mathbf{u})$ the only tunable part of the approximating distribution, being the inducing points parameters of this contribution exclusively as well. The whole approximation $q(\mathbf{f},\mathbf{u})$ can be seen as a way to separate the contributions from the inducing points and the functions evaluated at the original training inputs. Optimizing (3.3.17) according to this decomposition will provide a more accurate approximation to the original GP than what is obtained through the FITC approximation, which requires a change of priors, while here we are just obtaining a different approximation to the exact GP posterior and fitting it performing VI.

Minimizing the KL distribution between the exact GP posterior and the approximation of (3.3.19) requires maximizing the ELBO, as we have seen before. If we introduce the new decomposed approximation $q(\mathbf{f},\mathbf{u})$ we will have the objective function behave as

$$\begin{aligned}
\mathcal{L}(q,\theta) &= \int q(\mathbf{f},\mathbf{u}) \log\left(\frac{p(\mathbf{y},\mathbf{f},\mathbf{u}|\theta)}{q(\mathbf{f},\mathbf{u})}\right) d\mathbf{f}d\mathbf{u} \\
&= \int p(\mathbf{f}|\mathbf{u})q(\mathbf{u}) \log\left(\frac{p(\mathbf{y}|\mathbf{f},\theta)\cancel{p(\mathbf{f}|\mathbf{u})}p(\mathbf{u})}{\cancel{p(\mathbf{f}|\mathbf{u})}q(\mathbf{u})}\right) d\mathbf{f}d\mathbf{u} \\
&= \mathbb{E}_{q(\mathbf{f})}[\log p(\mathbf{y}|\mathbf{f},\theta)] - \mathrm{KL}\ (q(\mathbf{u})|p(\mathbf{u})).
\end{aligned} \tag{3.3.20}$$

This objective function is analogous to the original VI ELBO, but in this case it serves us to obtain an approximation to the exact GP posterior without having to make changes in the model. The first term in (3.3.20) depends on the mean squared prediction error of the model, while the KL divergence is calculated between two Gaussian distributions and thus has a closed-form solution. The main part of the calculations is dominated again by the inversion of the covariance matrices, but since in the KL divergence we are using only the $q(\mathbf{u})$ part of the approximation, here the scalability will also be $\mathcal{O}(M^2N)$ at most (the same as for the FITC approximation). Finally, predictions are obtained marginalizing $p(\mathbf{f}^*|\mathbf{u})q(\mathbf{u})$ over $\mathbf{u}$ (Hensman et al. 2013b).

VFE in general provides more accurate predictions than FITC, which is closely related to the fact that it does not need to change the model itself. However, in general they are considered to be more difficult to optimize than the FITC approach due to the complexity of the optimization space. In practice, VFE may be convenient since we do not have to change the GP model to obtain the benefits of the inducing points approach, and the performance outweights the intricacies of the optimization problem (Quinonero-Candela and Rasmussen 2005; Bauer et al. 2016; Bui et al. 2017).

3.3.2 Expectation Propagation

Asides from VI, there are other optimization-based approaches to approximate inference that have an important role in research as well. One of the main alternatives here is known as *expectation propagation* (EP), introduced mainly by the works of Minka 2001b; Minka 2001a. This algorithm is constructed around the minimization of the opposite KL divergence to the one being used in VI, which is given by

$$\mathrm{KL}(p|q) = \int p(\mathbf{z}) \log \left(\frac{p(\mathbf{z})}{q(\mathbf{z})} \right) d\mathbf{z}, \tag{3.3.21}$$

being $q(z)$ the approximating distribution to the real one, $p(z)$. Usually $q(z)$ is assumed to belong to the exponential family of functions, as is done in VI as well since this eases many of the calculations involved. If this assumption is taken, then we can write q as

$$q_\eta(\mathbf{z}) = \exp(\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z}) - g(\boldsymbol{\eta})), \tag{3.3.22}$$

where η is a vector of *natural parameters* of q inside this family of functions, $\mathbf{u}(\mathbf{z})$ is some vector function of $\mathbf{z}$ known as the *sufficient statistics*, and $g(\boldsymbol{\eta})$ is the logarithm of a partition function that acts as a normalizing constant, ensuring that q integrates to one and is therefore a well-defined probability distribution. If this approximation is introduced, minimizing the KL divergence can be done using closed-form expressions. Thanks to this, it can be obtained that

$$\mathbb{E}_{p(z)}[\mathbf{u}(\mathbf{z})] = \mathbb{E}_{q(z)}[\mathbf{u}(\mathbf{z})], \tag{3.3.23}$$

where we will write $q_\eta(\mathbf{z}) = q(\mathbf{z})$ to simplify the notation. Thus, the optimal solution for q is achieved by matching the expected sufficient statistics from q and p (which is sometimes called as *moment matching*).

In a more general sense, in (Minka 2001b; Minka 2001a) it is assumed that, for many probabilistic models, the joint distribution of the data $\mathcal{D}$ and hidden variables $\mathbf{z}$ can be factorized as

$$p(\mathcal{D}, \mathbf{z}) = \prod_i f_i(\mathbf{z}). \tag{3.3.24}$$

The basic EP algorithm approximates the joint distribution of $\mathcal{D}$ and $\mathbf{z}$ as a product of simple factors

$$p(\mathcal{D}, \mathbf{z}) \approx \prod_i \tilde{f}_i(\mathbf{z}), \tag{3.3.25}$$

where each factor $\tilde{f}_i$ in the approximation corresponds to one of the factors in the true joint distribution and where $\tilde{f}_i$ is restricted to have a similar form: they will belong to the exponential family of distributions, but do not need to be normalized. Then, the posterior approximation for $\mathbf{z}$ is simply

$$q(\mathbf{z}) = \frac{1}{Z} \prod_i \tilde{f}_i(\mathbf{z}) \tag{3.3.26}$$

where Z is a normalizing factor given by

$$Z = \int \prod_i \tilde{f}_i(\mathbf{z}) d\mathbf{z}. \tag{3.3.27}$$

To obtain the distribution that minimizes the KL divergence, in EP we must define a *cavity* distribution from our approximation q, which will be given as

$$q^{\backslash j}(\mathbf{z}) \propto \frac{q(\mathbf{z})}{\tilde{f}_j(\mathbf{z})}. \tag{3.3.28}$$

This distribution can be interpreted as the original approximating distribution but with the j-th factor removed from it and renormalized. We then compute the updated posterior distribution $\hat{p}$ as

$$\hat{p}(\mathbf{z}) = \frac{1}{Z_j} f_j(\mathbf{z}) q^{\backslash j}(\mathbf{z}) \tag{3.3.29}$$

where Z_j is the normalization constant to ensure integration to 1. With this, the solution of the KL minimization will provide us with the new approximation distribution q^{new}:

$$q^{new} = \arg\min_q \mathrm{KL}\left(\hat{p}(\mathbf{z}) | q(\mathbf{z})\right). \tag{3.3.30}$$

This is again solved by meeting the sufficient statistics, as we mentioned earlier. With this, the approximating factor $\tilde{f}_j(\mathbf{z})$ will be updated to

$$\tilde{f}_j(\mathbf{z}) = Z_i \frac{q^{new}(\mathbf{z})}{q^{\backslash j}(\mathbf{z})} \quad \text{with} \quad Z_i = \int \tilde{f}_j(\mathbf{z}) q^{\backslash j}(\mathbf{z}) d\mathbf{z}. \tag{3.3.31}$$

Applying this procedure, each factor is approximated in a *one at a time* basis through a cyclic algorithm. This is done in (Vehtari et al. 2020), where this cyclic approach is followed. However, this requires to keep in memory several approximate factors for each data point, which in a large scale data base is not feasible. As a response to the low scalability of basic EP there has been two recent developments in the form of *Probabilistic Backpropagation* (PBP) (Hernández-Lobato and Adams 2015) and *expectation backpropagation* (EBP) (Soudry et al. 2014). The latter case is an online extension of EP for NNs with *sign* activation functions and binary weights, and as in the case for PBP, it includes a forward propagation of gradients. However, EBP seems to lack some of the characteristics of PBP like the fact that it can only be used to model data with binary targets and that EBP with continuouss weights only updates the mean parameters of the Gaussian posterior approximations. In the other side, PBP is thought-out to be scalable in larger data sets where only a few passes over the data are possible (possibly due to the cost of evaluating each of the data values). For the case of (Hernández-Lobato and Adams 2015), PBP improves the results of VI approaches both in *root square mean error* and uncertainty estimation. Moreover, the approximating distributions for this papers are rather simple, as is the case for (Depeweg et al. 2017; Hernández-Lobato et al. 2016). The motivation behind these choices is fitting a more dispersed distribution than the ones in VI techniques, which although seems to sacrifice their estimations of the dominant modes of the posterior, they do achieve a correct search and exploration of the modes, which is usually what we are interested on when performing prediction tasks. Finally, it is interesting to remark the recent work done by Bui et al. 2016b, where deep Gaussian processes are employed along with EP using the previous developments in order to try to reduce the memory demand of EP. For this they dispose an approximation to the factorized approximating function in terms of a *average factor* of the different $f_i(\theta)$'s, significantly reducing the memory usage for EP by a factor of N (the number of factors that had to be considered this far). This is also strongly related to the work done by (Li et al. 2015). Their proposed method is *Stochastic EP* (SEP), in which the approximating factors of EP can be interpreted to parametrize a global factor that captures the average effect of the likelihood on the posterior. This extension to the usual formulation of EP improves significantly the scalability of EP when applied to large datasets.

As final remarks, the performance of EP may depend strongly on the case. It has been shown that it does not always converge to a solution (Minka 2001a), although certain modifications to the basic approach have been proposed in order to ensure convergence (Heskes and Zoeter 2012). Moreover, the dependence of EP on the moment-matching procedure can lead to poor performing results if the data distribution is complex enough, *e.g.* when the data distribution is multimodal. If the approximating distribution is selected within the exponential family in a case with bimodal data, the mean of the predictive distribution can be placed between the two modes of the true posterior, which may be a region with very low probability density. This is a problem that is also faced in VI, and it represents a major challenge for obtaining more flexible predictive distributions

3.3.3 Adversarial Variational Bayes

As we mentioned, the fact that VI uses a parametric family of distributions to find the best approximation for a distribution has important features, but also comes at the expense of limiting the behavior of the resulting distribution. The performance achieved by the basic approach to VI is strongly dependent on the choice of family of distributions for the approximation. Given certain data, VI can only choose the distribution inside that family that most resembles the exact posterior distribution we are looking for. This means that, if the exact posterior is far from the distributions of the selected family from which we can choose from, we will obtain a loosely precise approximating distribution, strongly biased by our selection of the family. This could imply losing crucial information if the divergence between the exact posterior and the approximation is large enough (Bishop 2006). For example, if we resort to the exponential family of distributions (a very common assumption), we will miss features such as multimodal behavior, heavy tails, skeweness, etc. Some different techniques have been proposed to deal with this issue, attempting to remove this restriction regarding the choice of family of the approximation function, and some of the most successful ones are those relying on *implicit models* (Salimans et al. 2015;

Rezende and Mohamed 2015; Li and Liu 2016; Mescheder et al. 2017).

An implicit distribution is usually referred to as a model from whom it is easy to draw samples, although we may not have a closed-form expression for the p.d.f. of said distribution. The *Adversarial Variational Bayes* approach makes extensive use of this concept to obtain a more flexible approximating distribution than what the regular VI approach is capable of producing (Mescheder et al. 2017). An example for an implicit model can be a source of standard Gaussian noise that is non-linearly transformed by a neural network. In this case, we can write the approximating distribution as

$$q_\phi(\mathbf{z}) = \int \delta\left(\mathbf{z} - \mathbf{f}_\phi(\boldsymbol{\epsilon})\right) \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I}) d\boldsymbol{\epsilon}\,, \tag{3.3.32}$$

where $\mathbf{f}_\phi(\boldsymbol{\epsilon})$ is the output of the transformation of the Gaussian noise, given by $\boldsymbol{\epsilon}$, ϕ represents the transformation's parameters, and $\delta(\cdot)$ is the delta function. We will mostly apply this for NNs, and therefore during this discussion we will use $\mathbf{w}$ instead of $\mathbf{z}$ to refer more precisely to a NN parameters, although the approach can be generalized to make use of other methods as well.

In general, the integral in (3.3.32) is intractable. This is caused by the fact that, in order to obtain a fairly general approach, the original Gaussian noise can be transformed through strong non-linearities such as the ones present in NNs. This is the case for most instances since, in order to be able to transform that noise into a sample from any given function, we may need an important degree of flexibility on the transformation. However, even though we will not dispose of an explicit expression for q, it is very easy to generate samples $\mathbf{w}$ from it ($\mathbf{w} \sim q_\phi$). For this, one only has to generate $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to then compute $\mathbf{w} = \mathbf{f}_\phi(\boldsymbol{\epsilon})$, which consists in simply passing the noise input through the transformations themselves. If the noise dimension is large enough and $\mathbf{f}_\phi(\cdot)$ is flexible enough, any probability distribution can be described using these method, which makes NNs especially interesting in this case since they can be used as universal approximators. Therefore, methods based on this approach attempt to alleviate the approximation bias of VI by substituting the parametric distributions q with an implicit model for it.

Using an implicit distribution inside the VI framework is challenging because the lower bound of (3.3.4) cannot be easily evaluated nor optimized in the usual manner. The KL divergence between the approximate distribution q and the prior on (3.3.3) usually requires the p.d.f. of q to be able to estimate the value of the integral. This is not the case here, since now we only have samples from the distribution but not its distribution. AVB provides an elegant solution to this problem by just employing samples from the approximate solution and the prior. In AVB, this term is written as

$$\begin{aligned} \mathrm{KL}(q(\mathbf{w})|p(\mathbf{w})) &= \mathbb{E}_{q_\phi(\mathbf{w})}\left[\log q_\phi(\mathbf{w}) - \log p(\mathbf{w})\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{w})}\left[T(\mathbf{w})\right], \end{aligned} \tag{3.3.33}$$

where $T(\mathbf{w})$ is simply the log-ratio between q_ϕ and the prior. In AVB this log-ratio is approximated as the output of an auxiliary problem: another neural network that discriminates between samples of $\mathbf{w}$ generated from q_ϕ and from the prior $p(\mathbf{w})$ (Mescheder et al. 2017). Let us consider $T_\omega(\cdot)$ as the output of the discriminator. Assuming $q_\phi(\mathbf{w})$ is fixed, the objective function for the discriminator will be:

$$\max_{\omega} \ \mathbb{E}_{q_\phi(\mathbf{w})} \log\left(\sigma(T_\omega(\mathbf{w}))\right) + \mathbb{E}_{p(\mathbf{w})} \log\left(1 - \sigma(T_\omega(\mathbf{w}))\right), \tag{3.3.34}$$

where $\sigma(\cdot)$ is the sigmoid function of (2.1.5). Roughly speaking, this objective tries to make the discriminator differentiate between samples generated from $q_\phi(\mathbf{w})$ and from the prior $p(\mathbf{w})$.

If the discriminator T_ω is considered flexible enough to represent any function of $\mathbf{w}$, it is possible to prove that the optimal discriminator behaves as expected (Mescheder et al. 2017). If we rewrite (3.3.34) using the explicit form for the expected values we obtain

$$\max_{\omega} \int \left[q_\phi(\mathbf{w}) \log \sigma(T_\omega(\mathbf{w})) + p(\mathbf{w}) \log(1 - \sigma(T_\omega(\mathbf{w}))\right] d\mathbf{w}\,. \tag{3.3.35}$$

This integral is maximal for $T_\omega(\mathbf{w})$ if and only if the integrand is maximal for every $\mathbf{w}$ value. The shape of the integrand is equivalent to

$$a \log t + b \log(1 - t), \tag{3.3.36}$$

for $a = q_\phi(\mathbf{w})$, $b = p(\mathbf{w})$, and $t = T_\omega(\mathbf{w})$. Thanks to this decomposition, it can be shown that the maximum value for the integrand is attained at $t = \frac{a}{a+b}$ (Mescheder et al. 2017; Goodfellow et al. 2014). Therefore, the optimal solution for (3.3.35), which we will denote as $T_{\omega^\star}$, is given by

$$\sigma(T_{\omega^\star}(\mathbf{w})) = \frac{q_\phi(\mathbf{w})}{q_\phi(\mathbf{w}) + p(\mathbf{w})}, \tag{3.3.37}$$

or equivalently,

$$T_{\omega^\star}(\mathbf{w}) = \log q_\phi(\mathbf{w}) - \log p(\mathbf{w}). \tag{3.3.38}$$

which is the result desired to correctly estimate the KL divergence between q_ϕ and the prior. In particular, the discriminator can be plugged in (3.3.33) and the expectation can be approximated simply by a Monte Carlo average by generating samples from q_ϕ.

Given $T_{\omega^\star}$ the lower bound employed in AVB is obtained by re-writing the evaluation of the KL divergence between q_ϕ and the prior:

$$\mathcal{L}(\phi) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(\mathbf{w})}[p(y_i|\mathbf{w}, \mathbf{x}_i)] - \mathbb{E}_{q_\phi(\mathbf{w})}[T_{\omega^\star}(\mathbf{w})]\,. \tag{3.3.39}$$

Note that all the required expectations can be simply approximated by generating samples from q_ϕ and the sum across the training data can be approximated using a mini-batch. This lower bound can be hence easily maximized w.r.t. ϕ using stochastic optimization techniques. For this, however, we need to differentiate the stochastic estimate with respect to ϕ. This may seem complicated since $T_{\omega^\star}(\mathbf{w})$ is defined as the solution of an auxiliary optimization problem that depends on ϕ. However, due to the expression for the optimal discriminator, it can be showed that $\mathbb{E}_{q_\phi(\mathbf{w})}\left(\nabla_\phi T_{\omega^\star}(\mathbf{w})\right) = 0$. Therefore the dependence of $T_{\omega^\star}(\mathbf{w})$ w.r.t ϕ can be ignored (see Mescheder et al. 2017 for further details). In practice, both q_ϕ and the discriminator $T_\omega(\mathbf{w})$ are trained simultaneously. However, q_ϕ is updated by maximizing (3.3.39) using a smaller learning rate than the one used to update the discriminator T_ω, which considers the objective in (3.3.34). This guarantees that T_ω is an accurate estimator of the log-ratio between q_ϕ and the prior, and that the KL divergence is correctly estimated when updating q_ϕ.

The AVB approach serves as a first approximation of what can be done to obtain more flexible predictive distributions than those present in VI and EP. Although it involves introducing the auxiliary discriminator problem, the performance of methods based on this approach have shown that this framework can be truly productive for conducting approximate Bayesian inference in ML and not only restricted to BNNs. This ideas are one of the key concepts of some of the main contributions of this thesis, and we will see how to expand on them later.

3.4 Sampling Based Methods

On the other hand, *sampling based methods* are a strong alternative to the optimization-based approaches to approximate inference. Here, the approximations are usually constructed around the concept of a *Markov Chain*. Instead of proposing an approximating distribution for the posterior, these approaches look for approximating it through a series of samples from it $\mathcal{S} = \{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N\}$, where $\boldsymbol{\theta}_i$ represents a vector containing all of the model's parameters (Bishop 2006). Then, applying the law of large numbers, an approximation to the predictive distribution can be written as

$$p(\hat{\mathbf{y}}^*|\mathbf{x}^*, \mathcal{D}) \approx \frac{1}{N}\sum_{i=1}^{N} p(\hat{\mathbf{y}}^*|\mathbf{x}^*, \boldsymbol{\theta}_i, \mathcal{D}), \tag{3.4.1}$$

where $\mathcal{D}$ represents the training dataset $(\mathbf{X}, \mathbf{Y})$, and the samples $\boldsymbol{\theta}_i$ are obtained from a Markov Chain where the stationary distribution coincides with the exact posterior we are looking to approximate (and which can be obtained without having to solve (3.2.3)). Sampling methods converge to the exact values, although in practice they have to run very long chains, which can be computationally demanding. A lot of research has been put into accelerating and applying these approaches, and here we will briefly introduce some of the basic concepts that we will later use as well.

3.4.1 Markov Chain Monte Carlo

Most sampling based approaches are based on Markov Chain Monte Carlo (MCMC), a general framework that can be seen as a simple method for obtaining samples out of a desired distribution, even if we do not know the exact shape this distribution may have (Bishop 2006; MacKay 2003) . Markov Chains can be defined using a set of states $\mathcal{S}$, where certain random variable $\mathbf{z}^{(n)} \in \mathcal{S}$ represents a fixed state at stage n and where n can be interpreted as a position in a succession of states (chain). The sequence $\mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(n-1)}$ is referred to as a *Markov chain of first order* if

$$p(\mathbf{z}^{(n)}|\mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(n-1)}) = p(\mathbf{z}^{(n)}|\mathbf{z}^{(n-1)}), \tag{3.4.2}$$

for any possible $n \geq 2$. This means that the value of the fixed state at position n in the chain will depend *only* on the previous state of the chain $(n-1)$ and not on the chain's history, which is usually referred to as the *memoryless property.* With this description, the Markov Chain can be fully specified given the initial probability distribution of states $p(\mathbf{z}^{(1)})$ and the matrix of transition probabilities, written as $p(\mathbf{z}^{(i+1)}|\mathbf{z}^{(i)})$. The Markov Chain is also said to be *homogeneous* if these transition probabilities do not depend on the position of a state in the chain.

The marginal probability at stage n in the chain can be easily obtained from the decomposition in (3.4.2), since the memoryless property allows us to decompose so that it only depends on the previous state

$$p(\mathbf{z}^{(n)}) = \sum_{\mathbf{z}^{(n-1)}} p(\mathbf{z}^{(n)}|\mathbf{z}^{(n-1)})p(\mathbf{z}^{(n-1)}), \tag{3.4.3}$$

where the summation is done across all possible values for the previous state in the chain. A distribution is said to be stationary or invariant with respect to a Markov Chain if each step in the chain leaves this previous marginal probability invariant. The stationary distribution of a Markov Chain can be defined as

$$p^*(\mathbf{z}) = \sum_{\mathbf{z}' \in \mathcal{S}} p^*(\mathbf{z}')p(\mathbf{z}|\mathbf{z}'). \tag{3.4.4}$$

The stationary distribution is important since it let us define the probability for every given state at any random position in the chain, once it is reached. A sufficient (although not necessary) condition that ensures a probability distribution is a stationary distribution is that it follows the *detailed balance* property

$$p^*(\mathbf{z})p(\mathbf{z}'|\mathbf{z}) = p^*(\mathbf{z}')p(\mathbf{z}|\mathbf{z}'), \tag{3.4.5}$$

for a given distribution $p^*(\mathbf{z})$. From this equation we could obtain again the stationary distribution of (3.4.4) marginalizing $\mathbf{z}'$. Moreover, a Markov chain that follows the detailed balance condition is said to be *reversible.*

The main concept behind the usage of Markov chains is that of constructing a chain whose stationary distribution is precisely the distribution we are trying to approximate. If done so, we would need to run the chain until it reached convergence, and after that we would sample from the distribution itself, approximating it by a Monte Carlo approach. Therefore, the very few components that shape the Markov chain and the stochastic evolution of its states are the only things we may need for these techniques. In the stationary distribution, the chain is expected to spend more time in states with higher probability, therefore conducting the sampling proportionately to the probability of that distribution. In practice, several chains are run at once for a long time. The first part of their chains is always discarded, since it is assumed that it does not correspond to samples from the stationary distribution, and afterwards samples are obtained from each chain to approximate the distribution, where the more samples we have, the more precise our estimates will be. However, ideally we would only use *independent* samples for the estimate. In order to consider the values from a chain as *independent* samples, we have to be aware that strong correlations may arise due to two states being close in the chain. Therefore, in order to obtain more independent samples from the target distribution, we will need to run the chain for a long time and afterwards keep only a given small portion of all positions sampled, where the precise number will depend on each case. This is a difficult problem to deal with, and in most cases it means that chains need to be run for long before their samples can be useful at all.

Many successful approaches have been constructed based on the MCMC framework. Among them, the most famous instances may be the *Metropolis-Hastings* and the *Gibbs sampling* methods. In both approaches, a rule to the evolution of states in the chain is set, which is usually referred to an *acceptance rate*. Depending on this acceptance rate, the state evolves to a newly proposed value or it does not, and if the evolution is rejected, the chain remains in the previous state for one more step and a new candidate is proposed. These candidates for the evolution are obtained in a random fashion following certain probability distribution that must be established in beforehand. The Metropolis-Hastings algorithm builds a Markov chain whose stationary distribution is the normalized version $p(\mathbf{z})$ of another unnormalized one $\tilde{p}(\mathbf{z})$

$$p(\mathbf{z}) = \frac{1}{Z}\tilde{p}(\mathbf{z}), \tag{3.4.6}$$

with Z the normalization constant. Given the current state $\mathbf{z}^(n)$ in the n-th step of a Markov chain, a candidate state $\mathbf{z}^*$ is generated from a proposal probability distribution $q(\mathbf{z}|\mathbf{z}')$. The chain transitions to the proposed state with probability given by

$$A(\mathbf{z}^*|\mathbf{z}^{(n)}) = \min\left(1, \frac{p(\mathbf{z}^*)q(\mathbf{z}^{(n)}|\mathbf{z}^*)}{p(\mathbf{z}^{(n)})q(\mathbf{z}^*|\mathbf{z}^{(n)})}\right). \tag{3.4.7}$$

If the new state is accepted, then we set $\mathbf{z}^{(n+1)} = \mathbf{z}^*$, and otherwise we set $\mathbf{z}^{(n+1)} = \mathbf{z}^{(n)}$, causing a repetition in the chain. From this expression, it can be seen that, if q is symmetric, it can be ignored and the probability of acceptance of a new state will only depend on the ratio of probabilities between the newly proposed one and the current state of the chain. The Metropolis-Hastings algorithm must balance between the number of states explored and sampling correctly regions with high probability in the distribution, However, when done correctly, it can provide useful samples and allow us to approximate the distribution we may need (Bishop 2006).

On the other hand, the Gibbs sampling method can be seen as a particular case of the Metropolis-Hastings algorithm on which the acceptance probability of a new candidate is always fixed to 1, and therefore it is encouraged that the chain of states explores the whole range of states possible. In Gibbs sampling, however, the sampling is done in a cyclical manner across the different possible parameters in the probability distribution we are looking to approximate. Sampling for each parameter one at a time, leaving the rest fixed meanwhile, simplifies the sampling procedure, and therefore allows to sample and approximate much more complex distributions than originally possible with Metropolis-Hastings in its basic form (Bishop 2006; MacKay 2003).

Although both the Metropolis-Hastings algorithm and Gibbs sampling are important approaches, we will not describe them into further detail since we do not have much use for them in later chapters. However, they are important techniques that are still very much relevant to current research. We will, however, introduce another sampling-based method that we will use later on: *Hamilton Monte Carlo*.

3.4.2 Hamilton Monte Carlo

The random walk model used in the previous MCMC models, although it works in the asymptotic limit, can pose a drawback in terms of the convergence time of the algorithm. This issue cannot be easily solved inside those approaches by simply increasing the size of the steps taken, since this may incur in worse samples and having to run the Markov chain for longer. Hamilton Monte Carlo can be seen as a more sophisticated approach that makes use of classical physics concepts such as Hamiltonian dynamics to make a more efficient sampler (Bishop 2006; MacKay 2003; Neal 2011).

Hamiltonian dynamics are an extension to the classical Lagrangian mechanics, on which a classical physical system is completely defined through few components such as the position and momentum of particles and an energy function. The energy function encodes information about the different forces being applied to the particles in the system, and the location and momentum of the particles inside that system evolve according to Hamilton's equations. The position of a given particle can be written as $\mathbf{z}$, and its momentum $\mathbf{p}$ is given element-wise by

$$p_i = m\frac{dz_i}{dt} \tag{3.4.8}$$

where m is the mass of said particle (for our purposes we can assume $m = 1$). Here, for each time t, the state of the particle is completely determined by $(\mathbf{z}, \mathbf{p})$, given an energy function that allow us to evolve the system through time.

The energies applying in a classical system such as this one could potentially be just the *kinetic* and *potential* energy, and if this is the case, the sum of both of them will provide the complete energy of a given particle for any given position and momentum. Assume that $p(\mathbf{z})$ is the probability distribution we are interested in sampling from. This distribution can be interpreted as the probability distribution around the location of the particle. In general, we can write this to be

$$p(\mathbf{z}) = \frac{1}{Z_p}\exp(-E(\mathbf{z})), \tag{3.4.9}$$

where $E(\mathbf{z})$ represents the potential energy of the system when it is on state $\mathbf{z}$. This holds without loss of generality, and it reflects that, as in physical systems, particles subject to a potential energy tend to locate themselves in places where this energy is minimized (which are positions with maximum probability here). Due to this fact, whenever the system is *not* at its configuration of minimum energy, the resulting effect is a *force* that appears in the opposite direction of the gradient of the potential energy, tending to minimize it. This can be written as

$$\frac{dp_i}{dt} = -\frac{\partial E(\mathbf{z})}{\partial z}, \tag{3.4.10}$$

and it only depends on the location of the particle inside the system. On the opposite side of this we have the kinetic energy term. This can be defined as

$$K(\mathbf{p}) = \frac{1}{2}||\mathbf{p}||^2 = \frac{1}{2}\sum_i p_i^2, \tag{3.4.11}$$

which is exclusively dependent on the momentum of the particle.

The sum of both energies allow us to define the Hamiltonian of the system as its total energy

$$H(\mathbf{z}, \mathbf{p}) = E(\mathbf{z}) + K(\mathbf{p}) \tag{3.4.12}$$

where H is now the Hamiltonian of the system. The whole motivation for the construction of the Hamiltonian is the fact that in this point we can make use of Hamilton's equations to study the evolution of the dynamic system in time

$$\frac{dz_i}{dt} = \frac{\partial H}{\partial p_i} \tag{3.4.13}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial z_i}. \tag{3.4.14}$$

Therefore, using this description we could obtain the position and momentum of a given particle at any time t.

HMC makes use of all of the previous description, where in the case of the sampling algorithm we will be interested in the position of the particle at each time t, while not so much on its momentum. The joint probability for the system to be at state $\mathbf{z}$ and with momentum $\mathbf{p}$ is

$$\begin{aligned} p(\mathbf{z}, \mathbf{p}) &= \frac{1}{Z_H} \exp(-H(\mathbf{z}, \mathbf{p})) \\ &= \frac{1}{Z_H} \exp(-E(\mathbf{z}) - K(\mathbf{p})). \end{aligned} \tag{3.4.15}$$

Since this distribution factorizes $\mathbf{z}$ and $\mathbf{p}$, to obtain samples from $p(\mathbf{z})$ we can simply sample from it and discard the momentum variables afterwards (which are easily sampled from a Gaussian distribution). Moreover, due to the structure of the kinetic energy function, the conditional probability for the momenta given $\mathbf{z}$ is simply

$$p(\mathbf{p}|\mathbf{z}) = \exp\left(-\frac{1}{2}\mathbf{p}^T\mathbf{p}\right). \tag{3.4.16}$$

The Hamiltonian framework has some important properties that are useful in terms of sampling from a distribution. First, the value for the Hamiltonian remains fixed throughout the evolution of the system, which is a direct consequence of the *Law of conservation of energy*. Applying the chain rule to the derivatives, one could easily see that

$$\frac{dH}{dt} = 0. \tag{3.4.17}$$

Another important feature is the fact that volumes that are evolved according to Hamilton's equations are preserved, also known as *Liouville's theorem*

$$\text{div } \mathbf{V} = \text{div } \left(\frac{d\mathbf{z}}{dt}, \frac{d\mathbf{p}}{dt} \right) = 0. \tag{3.4.18}$$

These two conservation rules are important for our case since, from both of them, it ensures us that evolving the probability distribution $p(\mathbf{z}, \mathbf{p})$ over time according to these equations will also provide us with a well-defined probability distribution (Bishop 2006). Moreover, integrating Hamilton equations we can make large changes in $\mathbf{z}$ by selecting different times t, which is useful when trying to obtain samples from $p(\mathbf{z})$.

The HMC algorithm works by initially choosing a value for $\mathbf{p}$, which is usually implemented by sampling from the conditional Gaussian probability in (3.4.16). Starting from an initial position for $\mathbf{z}$, the system is evolved according to the dynamics detailed by (3.4.13) and (3.4.14). This step can produce large changes in $(\mathbf{z}, \mathbf{p})$, producing a new candidate configuration $(\mathbf{z}^*, \mathbf{p}^*)$. Since the numerical integration of Hamilton's equations is not perfect, the acceptance probability will not always be 1 (since ideally we would like the system to remain in the trajectories defined by the Hamiltonian dynamics). To correct for this fact, an acceptance rule is set similar to Eq.3.4.7

$$A((\mathbf{z}^*, \mathbf{p}^*), (\mathbf{z}, \mathbf{p})) = \min \left(1, \frac{p(\mathbf{z}^*, \mathbf{p}^*)}{p(\mathbf{z}, \mathbf{p})} \right) = \min(1, \exp(-\Delta H)), \tag{3.4.19}$$

where ΔH quantifies the change in the estimation of H between configurations. This case differs from those seen previously since it does not require a proposal distribution, which is due to the fact of this system being reversible.

Among all of the sampling-based methods presented here, HMC is usually the one used as a gold-standard for approximate inference. The approximate solution that it converges to is actually the exact distribution, and it converges to it in much fewer iterations than those other methods previously mentioned. HMC does suffer from some of the same drawback, namely the need to run long chains to obtain independent samples, the computational cost, etc. However, this is usually used as a benchmark due to its performance, specially in datasets of manageable size and given that the model choice for fitting the data is at least somewhat adequate.

3.5 Alpha Divergence Minimization

As far as we have seen, the optimization-based approaches to approximate Bayesian inference are mainly based on the idea of minimizing the KL divergence between the exact posterior distribution of the parameters in our model p and a chosen approximation q, who may be more tractable and convenient for our use. Finding the best approximation usually implies maximizing the lower bound described in (3.3.2), which has the effect of minimizing said divergence (as shown in Figure 3.1). The object of the KL divergence has been widely studied in this context, since it plays a key role inside approximate inference and its properties directly affect the resulting distributions. The KL divergence can be extended to a more general class of divergences called α-divergences, also called *Renyi divergences*. These have important properties of which we will make extensive use, and therefore we will introduce them here.

To discuss this in a fairly general language, we will denote as p and q two given distributions defined over the vector $\boldsymbol{\theta}$. In general, the α-divergence between p and q is a non-negative function that is only equal to zero if $p = q$ (Amari 2012; Van Erven and Harremos 2014). The corresponding expression for the α-divergence between both distributions is

$$D_\alpha[p|q] = \frac{1}{\alpha(1-\alpha)}\left(1 - \int p(\boldsymbol{\theta})^\alpha q(\boldsymbol{\theta})^{1-\alpha} d\boldsymbol{\theta}\right) . \tag{3.5.1}$$

where α is a parameter defined such that $\alpha \in \mathbb{R} \setminus \{0, 1\}$. Depending on the value of α, different divergences between the distributions can be recovered. This is the case for some well-known measures, such as

$$D_1[p|q] = \lim_{\alpha\to 1} D_\alpha[p|q] = \mathrm{KL}(p|q) , \tag{3.5.2}$$

$$D_0[p|q] = \lim_{\alpha\to 0} D_\alpha[p|q] = \mathrm{KL}(q|p) , \tag{3.5.3}$$

$$D_{\frac{1}{2}}[p|q] = 2\int \left(\sqrt{p(\boldsymbol{\theta})} - \sqrt{q(\boldsymbol{\theta})}\right)^2 d\boldsymbol{\theta} = 4\,\mathrm{Hel}^2[p|q] . \tag{3.5.4}$$

The first two limiting cases, when $\alpha \to 1$ and when $\alpha \to 0$, represent the two different KL divergences between p and q, being the first the KL divergence used in EP (3.3.21), and the latter the one used in VI (3.3.3). Also, some other relevant functions can be obtained in terms of α, as is the case for (3.5.4), which is known as the *Hellinger distance*, which is the only instance in the family of α-divergences where the function is symmetric w.r.t. both inputs (unlike the KL divergences).

The value of the α parameter in the α-divergence has a strong impact in the inference results. To further understand its effect lets consider a toy problem in which we try to approximate a *slightly complex* distribution p with a more simple one, q. If we considered for example p as a bimodal distribution and q as a simple Gaussian distribution we would obtain the results displayed in Figure 3.2 (reproduced from Minka 2005). In this figure, the resulting (unnormalized) approximating distributions exhibit different behaviors. First of all, in the limit of $\alpha \to -\infty$, q (here represented

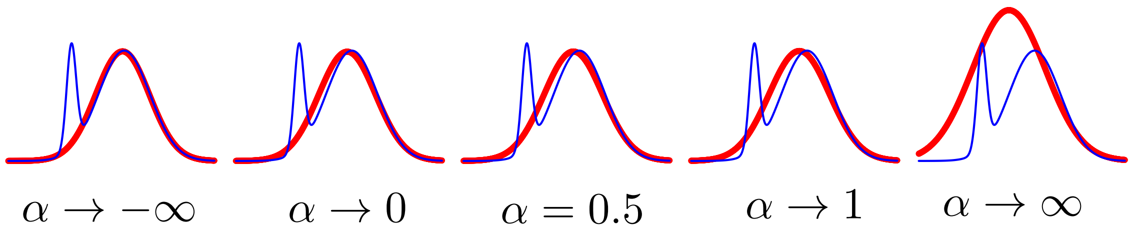


Figure 3.2: Changes on the approximate distribution q (in red) when trying to approximate it to the original distribution p (in blue) using different values for α in the α-divergence

in red) tends to cover only the mode with the larger mass of the two present in p. By contrast, when $\alpha \to \infty$, q tends to cover the whole p distribution, overlaying the latter completely. This can be seen in terms of the form of the α-divergence. More precisely, for $\alpha \leq 0$, the α-divergence emphasizes q to be small whenever p is small (thus it could be considered as zero-forcing). On the other hand, when $\alpha \geq 1$, it can be said that the divergence is inclusive, following the terminology of Frey et al. 2001. In this case, the divergence enforces $q > 0$ wherever $p > 0$, hence avoiding not having probability density in regions of the input space in which p takes large values.

In the remaining cases, α lays inside the interval $(0, 1)$. The behavior of q is intermediate between the two extreme possibilities that we have seen so far. In Figure 3.2 we can see that when $\alpha \to 0$ the q distribution is more centered in the main mode of p, whereas in $\alpha \to 1$ it begins to open to account for some of the mass of the secondary peak of p. This type of changes happen also when the distributions being considered are more complex than these ones, and therefore one should be careful when choosing a value for α if an α-divergence is used as dissimilarity measure. In particular, the optimal value of α may depend on the task at hand and the particular model one is working with. As it has been pointed out before, when α is restricted to be in the interval $(0, 1)$ we can obtain two notable results at the extremes, $D_\alpha = \text{KL}(q|p)$ for $\alpha \to 0$ and $D_\alpha = \text{KL}(q|p)$ for $\alpha \to 1$. This change of behavior is strongly related to the output differences between VI and EP, which we have mentioned before. Moreover, α-divergences have certain important properties, some of which could be interesting in order to exploit them in the approximate inference context (Van Erven and Harremos 2014).

3.5.1 Power Expectation Propagation

We will briefly introduce here the *power expectation propagation* algorithm (PEP) (Minka 2004), the first approximate inference method based on optimizing α-divergences constructed around the original EP algorithm. PEP is limited to use parametric approximate distributions such as a Gaussian distribution, and does not scale well to large datasets. However, it can be useful to better understand related methods such as black-box-α (Hernández-Lobato et al. 2016), which we will see later. In particular, we will restrict the description here to its objective function, since PEP allows for minimizing α-divergences in an approximate way. However, it suffers from the constrains such as choosing the approximate distribution $q(\mathbf{w})$ from inside a family of distributions of exponential distributions, as we saw in other optimization-based approximate inference approaches.

In general, the global minimization of the α-divergence is intractable except when $\alpha \to 0$ (see Minka 2005 for further details). Here, as we did in EP, we will let the unnormalized target distribution be described by the product of several factors, *i.e.*, $p \propto \prod_i f_i$. If we have i.i.d. data this is always the case, since the likelihood factorizes. The approximation for $p(\mathbf{w}|\mathcal{D})$ will be $q(\mathbf{w})$, which is again written as a product of simple factors $q \propto \prod_i \tilde{f}_i$. Each $\tilde{f}_i$ belongs to the exponential family (*e.g.* a Gaussian factor) and approximates the corresponding exact factor f_i, as in EP. However, PEP minimizes the α-divergence locally, instead of globally, between the *tilted* distributions of the model, and the approximate distribution $q(\mathbf{w})$. Therefore, PEP minimizes $D_\alpha[p^{\setminus j}|q]$ for all j. In general, it is expected that a local minimization of the α-divergence gives similar results to a global minimization while being a much simpler problem, as indicated by Minka 2005.

In the implementation of PEP, it is shown that the minimization of α-divergences is equivalent to the minimization of the KL divergence if certain factors are raised to the power of α (Minka 2005). In particular, we have to modify slightly the definitions we made for EP, especially the *cavity distribution* in (3.3.28) and the updated posterior distribution in (3.3.29). Now, they will be

$$q^{\setminus j}(\mathbf{z}) \propto \frac{q(\mathbf{z})}{\tilde{f}_j(\mathbf{z})^\alpha}, \qquad \hat{p}(\mathbf{z}) = \frac{1}{Z_j} f_j(\mathbf{z})^\alpha q^{\setminus j}(\mathbf{z}). \tag{3.5.5}$$

With these small changes, minimizing the KL divergence between the modified updated posterior and the approximating distribution will be analogous to minimizing the α-divergence. In order to conduct the local minimization of the α-divergence, PEP finally maximizes the following objective function:

$$\mathcal{L}(\phi, \{\theta_i\}_{i=1}^N) = \log Z_q - \log Z_{p(\mathbf{w})} + \frac{1}{\alpha}\sum_{i=1}^N \log \mathbb{E}_{q_\phi(\mathbf{w})}\left[\left(\frac{p(y_i|\mathbf{w},\mathbf{x}_i)}{\tilde{f}_i(\mathbf{w})}\right)^\alpha\right], \tag{3.5.6}$$

where Z_q is the normalization constant of q_ϕ, $Z_{p(\mathbf{w})}$ is the normalization constant of the prior, ϕ are the parameters of $q(\mathbf{w})$ and $\{\theta_i\}_{i=1}^N$ are the parameters of the approximate factors $\tilde{f}_i$. In this case, we have assumed that the prior distribution need not be approximated and already belongs to the exponential family (*i.e.*, it is a Gaussian prior). When $\alpha \to 0$, (3.5.6) converges to the lower bound of VI (Minka 2005). Therefore, a local minimization of the KL divergence employed in VI is equivalent to a global minimization.

In practice, PEP solves the problem $\max_\phi \min_{\{\theta_i\}_{i=1}^N} \mathcal{L}(\phi, \{\theta_i\}_{i=1}^N)$, which is a complicated task since it requires a slow double loop algorithm (Heskes and Zoeter 2002). Furthermore, PEP does not scale to big data since it maintains an approximate factor associated to each likelihood factor, which results in a space complexity of $\mathcal{O}(N)$. As we will see now, the Black-box-α approach provides a simple way of addressing these problems.

3.6 Approximate Inference with Implicit Processes

In the previous chapter we have seen that GPs are able to obtain good predictions based on the idea of placing a prior distribution over the space of functions such that functions sampled from it will behave in a Gaussian-like manner. This fact allowed many of the calculations involved in the training of GPs to be solved analytically, and in some cases it also enabled approximations that solved some of their scalability issues, which introduced us to the concept of *inducing points* used in Sparse GPs. These are important features from the GP model, but they come at the expense of only being able to provide normally-distributed predictions. In many real-world cases, this could be seen as a major drawback, since the data may behave in manners that differ (sometimes strongly) from the Gaussian behavior that GPs are acquainted with. Here we will introduce *implicit processes* (IPs), a new set of models that generalize over the concepts that GPs use in order to be able to conduct approximate inference inside the space of functions while avoiding restricting our predictions to behave in a Normal-like fashion (Ma et al. 2019).

The use of implicit processes (IPs) in approximate inference is specially interesting due to the fact that they allow to avoid many issues related to the optimization in the weight-parameter space. As well as GPs, IPs are optimized using the function-space, and although this may complicate a bit the mathematical formulation behind them, it also carries important features and advances that may simplify and improve the approximate inference methods we have introduced thus far. To be more specific, as seen in (Sun et al. 2019), since the weight space is very large, achieving a final good optimal value for the parameters will depend on the sheer number of parameters and on the complexity of the model. When systems based on parameter-space optimization, such as BNNs, increase in complexity, they are subject to the appearance of undesirable behaviors in the final predictions. This behavior can be explained by the model's inability to properly balance between the increasing relative weight of the prior in comparison to the data-provided information in the objective function: using the language of VI, since the model's size is increased, more parameters are introduced (*e.g.* in a BNN, more layers are employed), which makes the KL contribution to the ELBO (3.3.4) much bigger than the data-dependent term. Although for simple setups, the model can comprehend the data and properly fit it, raising its complexity makes it increasingly unable to do the same (unless many extra data points are available, that is). This behavior has strong connections to the fact that the parameter space has a great number of symmetric modes and strong correlations between different parameters of the model, which translates in bad final results in the optimization procedure since the global minima is not being clearly defined. Among all of the different ways to deal with these issues, one of the most relevant for current research in approximate Bayesian inference is using IPs to perform inference in the function-space rather than in the weight-space, as recent works such as Sun et al. 2019; Ma et al. 2019 have proposed.

An implicit process is defined as a collection of random variables $f(\cdot)$, such that any finite collection $(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$ has a joint distribution defined by the generative process:

$$\mathbf{z} \sim p(\mathbf{z}), \qquad f(\mathbf{x}_n) = g_\theta(\mathbf{x}_n, \mathbf{z}) \qquad \forall \mathbf{x}_n \in \mathbf{X}. \tag{3.6.1}$$

where θ represent the hyperparameters of the distribution over $\mathbf{z}$, $p(\mathbf{z})$, and g is a non-linear function. We use the notation $f(\cdot) \sim \mathcal{IP}(g_\theta(\cdot,\cdot), p_\mathbf{z}) = \mathcal{IP}_\theta(\cdot)$ to indicate that f is sampled from the corresponding IP with parameters θ.This simple description provides a wide framework from which we can define many other models: as an example, a Bayesian neural network can actually be expressed in terms of an IP, since it can be considered to define a prior distribution over functions

$$\mathbf{W} \sim \mathcal{N}(\mathbf{W}|\mathbf{0}, \mathbf{I}), f(\mathbf{x}) = g_\theta(\mathbf{W}, \mathbf{x}), \tag{3.6.2}$$

where now the hyper-parameters θ are the means and variances of the weights contained in $\mathbf{W}$. Another example here are Neural Samplers (NS), where in this case $g_\theta(\cdot,\cdot)$ is a NN with weights θ, *i.e.* $NN_\theta(\cdot,\cdot)$, and $p(\mathbf{z}) \sim \mathcal{U}([-a,a]^d)$. In this case the uniform noise is processed alongside the NN as well as the input data, and the final output combines the result of both. The usage of NNs is important here since using their strong non-linearities, the functions sampled from an IP can have as much flexibility as we may need in any certain given issue, and thus we can consider this type of networks as way of obtaining samples from the implicit distribution of the implicit process. This therefore conforms an implicit model setup such as the one we established in Section 3.3.3, where we will have a way to sample from a given distribution for which we do not have a closed-form p.d.f.

Using implicit processes priors in any method would allow for an increment in the flexibility of the model in comparison to most of the current state of the art. There has been an increasing amount of work in this topic to try to provide a comprehensive approach to approximate inference with IPs, and we will focus here on two of the most popular methods thus far: *Variational Implicit Processes* (VIPs, introduced by Ma et al. 2019, and *Functional Bayesian Neural Networks* (FBNNs), by Sun et al. 2019.

3.6.1 Variational Implicit Processes

Dealing with IPs in the formulation of a method leads to several theoretical difficulties. As a first attempt at providing a more general approach for the usage of IPs, *Variational Implicit Processes* were introduced in (Ma et al. 2019). The VIP approach proposes the usage of an IP prior so that the flexibility of IPs can be of use for the model. However, in order to keep all parts of the model trainable, they resort to a series of approximations in the posterior that could in principle hinder the final flexibility of the predictive distribution. VIPs approximate the marginal likelihood of the IP by the marginal likelihood of a GP with an empirical covariance function, whose mean and covariance function are estimated by generating samples from the prior IP. The main part of the VIP approach consists in a slight modification to the wake-sleep algorithm that works as follows:

1. Generate S samples from the implicit process $f_s^\theta(\mathbf{x}) \sim \mathcal{IP}(g_\theta(\mathbf{x}, \mathbf{z}), p_\mathbf{z})$, where $s \in \{1, \cdots, S\}$ and $\mathbf{z} \sim p(\mathbf{z})$.

2. Compute the moments of the samples using

$$m^\star_{\text{MLE}}(\mathbf{x}) = \frac{1}{S}\sum_{s=1}^{S} f_s^\theta(\mathbf{x}), \quad K^\star_{\text{MLE}}(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{S}\sum_{s=1}^{S} \Delta_s^\theta(\mathbf{x}_1)\Delta_s^\theta(\mathbf{x}_2), \tag{3.6.3}$$

   where $\Delta_s(\mathbf{x}) = f_s^\theta(\mathbf{x}) - m^\star_{\text{MLE}}(\mathbf{x})$.

3. Approximate $p(\mathbf{y}|\mathbf{X}, \theta)$ by the marginal likelihood of a GP with the previously defined moments, denoted by $q_{\text{GP}}(\mathbf{y}|\mathbf{X}, \theta)$.

4. Maximize $q_{\text{GP}}(\mathbf{y}|\mathbf{X}, \theta)$ expecting that it will increase $p(\mathbf{y}|\mathbf{X}, \theta)$.

This approach, although simple, implies that the final approximate distribution is here a standard Gaussian process (GP), which will approximate the intractable true posterior given by the IP prior. Therefore, although VIP is able to employ IPs as prior model, which may benefit the approach overall, it will also suffer from the same type of issues that we saw before for the GPs, mainly the memory requirements and the Gaussian-like predictions. The estimated values here will be those given by (3.6.3). Then, the predictive distribution is approximated by the GP predictive distribution as well

$$\mathbb{E}[f(\mathbf{x}_*)] = m^\star_{MLE}(\mathbf{x}_*) + \mathbf{K}_{*,\mathbf{f}}(\mathbf{K}_{\mathbf{f},\mathbf{f}} + \mathbf{I}\sigma^2)^{-1}(\mathbf{y} - m^\star_{\text{MLE}}(\mathbf{X})), \tag{3.6.4}$$

$$\text{Var}(f(\mathbf{x}_*)) = \mathbf{K}_{*,*} - \mathbf{K}_{*,\mathbf{f}}(\mathbf{K}_{\mathbf{f},\mathbf{f}} + \mathbf{I}\sigma^2)^{-1}\mathbf{K}_{\mathbf{f},*}\,, \tag{3.6.5}$$

which can be computed efficiently given that the covariance matrices have rank S. It is important to remark here that the covariance matrices are empirically estimated, as indicated before in (3.6.3).

To guarantee scalability and avoid the cubic cost of the GP they further approximate the GP using a linear model with the same mean and covariances. The linear model is efficiently tuned by optimizing the α-energy function performing gradient descent w.r.t. θ via the method described in (Hernández-Lobato et al. 2016). The main issue with VIPs is therefore the need to resort to GPs as a final approximation for the posterior distribution, and therefore restricting the intrinsic flexibility present in the formulation of the IPs used as priors. Although they seem to give sensible results, there is some room for improvement in the latter steps of the approach.

3.6.2 Functional Bayesian Neural Networks

Using IPs for all of the components of the Bayesian approach is a challenging task, which explains the approximations made in (Ma et al. 2019) for VIP. The posterior distribution that is produced by using a prior IP in the model is complex to deal with, and therefore some sort of approximate solution needs to be set. In contrast to what is proposed VIPs, *functional BNNs* (FBNNs) rely on using VI and another IP to approximate the posterior distribution of the IP specified in the prior (Sun et al. 2019). However, when evaluating the ELBO (3.3.4), the KL term now needs to be estimated between two stochastic processes, which makes it intractable again. To solve this issue, they resort to evaluating the KL over a finite set of points $\mathbf{X}$. This reduces the KL divergence to the following expression

$$\mathrm{KL}(p|q) = \sup_{n\in\mathbb{N},\, \mathbf{X}\in\mathcal{X}^n} \mathrm{KL}(p(\mathbf{f}^{\mathbf{X}}|q(\mathbf{f}^{\mathbf{X}})), \tag{3.6.6}$$

where the *measurement set* is defined as $\mathbf{X} \in \mathcal{X}^n$, with $\mathcal{X}^n$ a finite measurement set of n data instances, and $\mathbf{f}^{\mathbf{X}}$ represents the functions sampled from the IP and evaluated on each value of said set. The details of this identity are further explained in (Sun et al. 2019). Once we have this approximation, we can rewrite the ELBO expression so that it becomes

$$\mathcal{L}_{\mathbf{X}}(q) = \inf_{n\in\mathbb{N},\, \mathbf{X}\in\mathcal{X}^n} \sum_{i=1}^{N} \mathbb{E}_q[\log p(y_i|f(\mathbf{x}_i))] - \mathrm{KL}(q(\mathbf{f}^{\mathbf{X}})|p(\mathbf{f}^{\mathbf{X}})), \tag{3.6.7}$$

where now the inf appears due to the sign of the KL in the objective function.

The crucial aspect in FBNN is how to choose the measurement set to estimate the KL divergence, and also how to estimate the gradients of this term to be able to train the model appropriately. Regarding the first point, it is important for $\mathbf{X}$ to be sampled from the complete possible space of inputs, considering both the training and testing points. This is a critical aspect of the method, since covering the test region is relevant for the method to make correct predictions and interpolations. However, this can be a difficult task, since in many cases we will not have this information available in beforehand. Moreover, regarding the choice of the measurement set positions, several different approaches are proposed here, showing that the only strict lower bound on the log-marginal likelihood is using all training inputs as the measurement set. This may lead to scalability issues if the dataset is large, so other methods for choosing this set are proposed. The one with the best performance seems the sampling-based approach, from which the measurement points are randomly sampled from the training inputs and from the regions where one might be interested in making predictions (incurring on the mentioned issue about the information of the test regions).

Once there is a choice for measurement set, the KL term can be approximated and therefore we can evaluate the objective function. In this point we still need to estimate the gradients of the objective function to train the system, and in FBNN this is done by the Stein Gradient Estimator and the Nymströn method. Here, applying Monte Carlo and truncating several summation terms, an approximation to the real gradient can be obtained (a more detailed discussion can be found in Sun et al. 2019).

With FBNNs it can be shown that some of the pathologies of the weight-space optimization are removed altogether thanks to the function-space-based optimization. In this case, increasing the complexity of the model does not come at the expense of progressively losing the quality of the predictions already achieved in simpler models.

Hence, this approach allows a better balance more between the prior information and the observed data distribution, compensating the behavior that appears in approaches based in weight-space optimization. However, there are several important drawbacks to this approach that can be of relevance: mainly, the need to resort to a finite measurement set evaluation to approximate the KL term, and also the inability for FBNN to train its prior model. The sampling-based system to estimate the KL may be prompt to issues depending on the type of dataset we are dealing with (*curse of dimensionality*), while also requiring information about the regions on which the predictions will be set. On the other hand, due to the approximations needed to estimate the gradients, FBNN is incapable of training the prior and the rest of the model simultaneously. In practice, this can be solved by using a GP as prior, training it on the data in beforehand, and only then training the rest of the system, maintaining the IP approximation as a posterior (Sun et al. 2019). In general, it can be shown that the posterior distribution will preserve the behavior of the functions sampled from the prior. However, if an IP prior model is used and this is not trained in beforehand, the prior will remain untrained, which may result in an important hindrance for the performance of the model. This fact leads to (Sun et al. 2019) resorting to a sparse GP model as prior for most regression problems.

Even though the framework of IPs is promising, we have seen that both FBNNs and VIPs present several important points that could be improved further. In the following chapters we will present some proposals to solve some of these issues, especially in Chapter 5.

Chapter 4

Adversarial α-divergence Minimization

We have seen that estimating the uncertainty in the predictions of a ML algorithm is a critical aspect with important applications. To this regard, we have focused on providing Bayesian approaches to obtain posterior distributions for the parameters in the models. These distributions summarizes which parameter values are compatible with the observed data, although are often intractable and need to be approximated in practice. In this chapter we will introduce our first contribution to approximated Bayesian inference in ML. To that end, we will start by introducing *black-box α-divergence minimization*, which combines many ideas we have introduced earlier and will allow us to construct a more general approach based on minimizing α-divergences, and that allows for flexible approximate distributions. We call this method adversarial α-divergence minimization (AADM). We have evaluated AADM in the context of Bayesian neural networks. Extensive experiments show that it may lead to better results in terms of the test log-likelihood, and sometimes in terms of the squared error, in regression problems. Moreover, we show that the selection of the value of α can be exploited to achieve better results for the desired performance metric.

4.1 Motivation

In Section 2.1 we have introduced how NNs have become such popular methods for a wide variety of tasks. This is mostly related to their good empirical achievements across multiple learning problems. Specifically, DNNs trained with back-propagation have significantly improved the state-of-the-art in supervised learning tasks (LeCun et al. 2015). Moreover, variations of the simple original NN models have been specifically designed to take advantage of underlying structure on the input data. This is the case for CNNs Krizhevsky et al. 2012 or LSTMs (Hochreiter and Schmidhuber 1997), both of which represent some of the best performing models for dealing with structured data such as images and texts, respectively (Krizhevsky et al. 2012; Jozefowicz et al. 2016; Sutskever et al. 2014). NNs can be trained on Graphical Processing Units (GPUs), which significantly reduces the total training time and the effort needed to produce highly accurate results. These models can therefore be trained on huge amounts of data very quickly, showing excellent results in regression and a competitive performance also in classification tasks. There we also mentioned that, despite of these advantages, the good performance results come with concerns about over-fitting due to the high number of parameters to be adjusted, or the lack of a confidence measure on the predicted outputs associated to the input data (Gal 2016). More precisely, regular NNs only produce point-estimate predictions and do not provide any information about the certainty of such outcome. Even in multi-class problems where the results are given in terms of a soft-max function which outputs probabilities, it is important to keep in mind that the output values do not correspond to the confidence of the prediction. In particular, a high class label probability may correspond to a data instance that will be often misclassified by the network.

The problems described can be addressed by following a Bayesian approach in the training process as we described in Chapter 3, where we introduced different possible techniques to conduct approximate Bayesian inference in complex models such as NNs. One of the main features of Bayesian probabilistic models such as Bayesian neural networks (BNNs) (Neal 2012) is that they are able to capture the uncertainty in the model parameters (the network weights) and the effects it produces in the final predictions, therefore providing an estimate of the models' ignorance on the target value associated to the input data in each specific case. This extra output information can be used in different ways: for example, confronting problems in artificial intelligence safety, performing active learning, or dealing with possible adversaries which may manipulate the data (Gal 2016). Therefore, uncertainty estimates associated to the model predictions can be very important to make optimal decisions when dealing with input data that the machine learning algorithm has never seen before.

As a summary from the basic ideas presented in Chapter 3, the Bayesian approach relies on computing a posterior distribution for the model parameters given the data (Bishop 2006; Gal 2016). This posterior distribution is obtained using Bayes' rule simply by multiplying a likelihood function (which captures how well specific values of the parameters explain the observed data) and a prior distribution (which includes prior knowledge about what potential values these parameters may take). This posterior distribution summarizes which model parameters (*i.e.*, the neural network weights) are compatible with the observed data. Intuitively, if the model is rather complex, the posterior will be very broad. By contrast, if the model is fairly simple, the posterior will concentrate on a specific region of the parameters space. The information contained in the posterior distribution can be readily translated into a predictive distribution which carries information about the uncertainty on the predictions made. For this, one simply has to average the predictions of the model for each parameter configuration weighted by the corresponding posterior probability.

The need for approximate Bayesian inference arises from the fact that the posterior distribution is usually intractable for most problems. Therefore, one must resort to approximate methods, using either an optimization-based approach (Section 3.3) or a sampling-based model (Section 3.4). Here we will mostly focus on the first, on which the parameters of an approximated distribution q are tuned by minimizing a divergence between it and the exact posterior (Section 3.3). This is how methods such as VI and EP work in practice, as well as some other more sophisticated techniques such as black-box-α, which we will introduce here briefly (Hernández-Lobato and Adams 2015; Hernández-Lobato et al. 2016; Graves 2011). Although these methods are very fast and scalable, they often suffer from the lack of flexibility of the approximate distribution q, which is typically set to be a parametric distribution that cannot adequately match the exact posterior. Therefore, these methods may suffer from strong approximation bias. Importantly, a poor approximation of the exact posterior is expected to lead to a worse predictive distribution, less accurate predictions, and a worse estimate of the uncertainty in the predictions made.

Recently, several methods have been proposed to increase the flexibility of the approximate distribution q (Rezende and Mohamed 2015; Mescheder et al. 2017; Liu and Wang 2016; Salimans et al. 2015; Tran et al. 2017). One of the main paths to increase the flexibility of the approximation is the usage of implicit models (Li and Liu 2016), such as AVB (Section 3.3.3). In implicit models, if the non-linear transformations applied to the original input noise are flexible enough, almost any distribution can be reproduced and sampled from. However, in these cases q has no explicit p.d.f., which makes marginalizing the input noise intractable. This impedes most of the calculations needed in the inference process, which makes training the models a very challenging task. We showed in Section 3.3.3 that AVB was able to bypass some of these problems and successfully train the system by employing an auxiliary problem, a discriminator network that estimates the log-ratio between the posterior approximation q and the prior distribution over the model parameters (Mescheder et al. 2017). This provides one of the most relevant instances of successful usage of implicit models for approximate inference, setting a trend that will continue until today.

AVB and also other methods such as VI or EP (only locally and in the reversed way) rely on minimizing the KL divergence between the approximate distribution q and the exact posterior. The KL divergence can be understood as a regularization term in the objetive function, and new research has shown that modifying this term can provide better results (Wenzel et al. 2020). We have also seen that the KL divergence can be generalized in terms of α-divergences, recovering the VI objective when $\alpha \to 0$, and if $\alpha = 1$, the α-divergence is equivalent to the KL divergence which is locally optimized by EP (Minka 2005). Recently, it has been empirically shown that one can obtain better results, in terms of the approximate predictive distribution, by minimizing α-divergences locally using intermediate values of the α parameter in the case of parametric q (Hernández-Lobato et al. 2016). However, it is not clear if one can also obtain better results in the case of using implicit models for q, such as the one considered by AVB.

Here we will try to provide a new extension to previous work, introducing a method to locally minimize α-divergences employing an implicit model for the approximating distribution. We refer to such a method as *Adversarial α-divergence minimization* (AADM), which can be seen as a generalization of AVB that allows to optimize a more general class of divergences, resulting in flexible approximate distributions q with different properties (Rodríguez Santana and Hernández-Lobato 2020). In particular, when $\alpha \to 0$, AADM targets the same objective as AVB, and if $\alpha = 1$, its objective is similar to EP with a flexible approximate distribution q. Intermediate values of α result in different properties of the approximate distribution. We have evaluated AADM in the context of Bayesian Neural Networks and tested different values of the α parameter. The experiments carried out involve several benchmark regression and classification problems. These show that in regression problems one can obtain, in general, better prediction results than those of AVB and standard VI by using intermediate values of α. In particular, the mean squared error, test log-likelihood, and other performance metrics of the predictive distribution such as the *continuous ranked probability score* (CRPS) (Gneiting and Raftery 2007) improve when intermediate values of α are used. We have also evaluated AADM in the context of binary and multi-class classification problems. In these cases, however, we have observed that AADM gives similar results to those of AVB, both in terms of the prediction error and test log-likelihood, as well as with other performance metrics based on the *Brier* score (Gneiting and Raftery 2007).

4.2 Black-box α-divergence Minimization

Black-box-α (BB-α) (Hernández-Lobato et al. 2016) is an improvement over the power expectation propagation method for approximate inference introduced in Section 3.5.1. This approach is able to address some of the previous limitations like PEP's $\mathcal{O}(N)$ memory space requirements while allowing for approximate inference on complicated probabilistic models (Hernández-Lobato et al. 2016). In order to do so, BB-α maximizes a modified version of the objective of PEP. Namely,

$$\mathcal{L}(\phi) = \log Z_q - \log Z_{p(\mathbf{w})} + \frac{1}{\alpha}\sum_{i=1}^{N} \log \mathbb{E}_{q_\phi(\mathbf{w})}\left[\left(\frac{p(y_i|\mathbf{w},\mathbf{x}_i)}{\tilde{f}(\mathbf{w})}\right)^{\alpha}\right], \tag{4.2.1}$$

where Z_q is the normalization constant of q_ϕ, $Z_{p(\mathbf{w})}$ is the normalization constant of the prior, ϕ are the parameters of $q(\mathbf{w})$, $p(y_i|\mathbf{w},\mathbf{x}_i)$ is a likelihood factor and $\tilde{f}(\mathbf{w})$ is a global approximate likelihood factor that is replicated N times, one per each likelihood factor (*i.e.*, we will only have *one* factor N-times, instead of the N that PEP requires). This results in $q_\phi(\mathbf{w}) \propto \tilde{f}(\mathbf{w})^N p(\mathbf{w})$, which solves PEP's problem of having to store in memory the parameters of N approximate factors (as before, one per each likelihood factor). Furthermore, there is a one to one map between $\tilde{f}(\mathbf{w})$ and $q_\phi(\mathbf{w})$. This means that the max-min optimization problem of the PEP objective is transformed into just a standard maximization problem (w.r.t to the parameters of $q(\mathbf{w})$, ϕ), which can be solved using standard optimization techniques. Importantly, the expectations in (4.2.1) can be approximated via Monte Carlo sampling and the sum across the training data can be approximated using a mini-batch. The consequence is that BB-α scales to big datasets, as (4.2.1) can be optimized using stochastic techniques, and moreover, it can be applied to complicated probabilistic models (*e.g.*, Bayesian neural networks) in which the required expectations are usually intractable.

As in PEP, BB-α minimizes locally the α-divergence. In particular, it minimizes the sum of α-divergences between the approximate distribution q_ϕ and the tilted distributions, which can be written here as

$$\hat{p}_i(\mathbf{w}) \propto \tilde{f}(\mathbf{w})^{N-1} p(y_i|\mathbf{w}, \mathbf{x}_i) p(\mathbf{w}) \quad \text{for} \quad i \in \{1, \ldots, N\}. \tag{4.2.2}$$

The local minimization of α-divergences is expected to give similar results to the global case, while being much simpler to optimize (Minka 2005). As we mentioned earlier, when $\alpha \to 0$ (4.2.1) converges to the lower bound of VI (Section 3.5). On the other hand, when $\alpha = 1$, (4.2.1) becomes approximately equal to the objective optimized by EP (Hernández-Lobato et al. 2016). A limitation of BB-α is, however, that the approximate distribution $q(\mathbf{w})$ is restricted to be inside the exponential family since it must be written as the product of an approximate factor times the prior distribution and this restriction makes this a simpler task. Therefore, our approximation should be able to be decomposed as

$$q_\phi(\mathbf{w}) \propto \tilde{f}(\mathbf{w})^N p(\mathbf{w}). \tag{4.2.3}$$

This is a major limitation, since this restricts the BB-α algorithm to be used on parametric distributions with the ability to be expressed in a factorizes manner. That fact makes difficult using implicit models for $q(\mathbf{w})$ here, which we have mentioned can be important alternatives when trying to find a more flexible approximation model. Therefore, in order to be able to implement this new set of models, we will need to reformulate the BB-α objective accordingly.

4.2.1 Reparametrization of the BB-α Objective

In this section we introduce the reparametrization for the general expression of the BB-α objective that is suggested by Li and Gal 2017 for approximate distributions with a closed form expression for the p.d.f. We combine this reparametrization with the trick of AVB to estimate the log-ratio between probability distributions. This will allow to provide the first original contribution of this thesis, that is to approximately minimize α-divergences with flexible distributions $q(\mathbf{w})$ such as the ones resulting from implicit models. All of these ingredients together will provide a complete definition of our newly-proposed method, AADM, which extends the existing literature on these topics. With this goal, we first consider the following alternative expression for the BB-α objective that is described by Li and Gal 2017:

$$\mathcal{L}_\alpha(\phi) = \frac{1}{\alpha} \sum_{i=1}^{N} \log \mathbb{E}_{q_\phi(\mathbf{w})} \left[\left(\frac{p(y_i|\mathbf{x}_i, \mathbf{w}) p(\mathbf{w})^{1/N}}{q_\phi(\mathbf{w})^{1/N}} \right)^\alpha \right]. \tag{4.2.4}$$

In this expression we observe that the hypothesis that $q_\phi(\mathbf{w}) \propto \tilde{f}^N(\mathbf{w})p(\mathbf{w})$ is not required anymore, since both Z_q and $\tilde{f}(\mathbf{w})$ are removed from the expression, and $q(\mathbf{w})$ can be an arbitrary distribution. It is possible to show that (4.2.4) and (4.2.1) are equivalent expressions if $q(\mathbf{w})$ belongs to the exponential family (Li and Gal 2017). However, this expression requires the evaluation of the density $q_\phi(\mathbf{w})$, which in practice may be hard to compute when using an implicit model for that distribution. This is the first difficulty we will need to address here in order to improve on the current methods available.

To overcome these previous limitations about the imposed shape of the approximate distribution we will follow similar steps to the ones of Li and Gal 2017, reparametrizing (4.2.4) using the *cavity distribution*, that is, the distribution given by the ratio $q_\phi/\tilde{f}^\alpha$. If $\tilde{q}_\phi(\mathbf{w})$ denotes a free-form cavity distribution, the posterior approximation q_ϕ is given by:

$$q_\phi(\mathbf{w}) = \frac{1}{Z_q}\tilde{q}_\phi(\mathbf{w})\left(\frac{\tilde{q}_\phi(\mathbf{w})}{p(\mathbf{w})}\right)^{\frac{\alpha}{N-\alpha}} \tag{4.2.5}$$

where we assume $Z_q < +\infty$ is the normalizing constant to make $q(\mathbf{w})$ a valid distribution. When $\alpha/N \to 0$ we have that $q \to \tilde{q}$ (and $Z_q \to 1$ by assumption), and this is the case either if we choose $\alpha \to 0$, or when N is sufficiently large (i.e. $N \to +\infty$), see (Li and Gal 2017). We rewrite now (4.2.4) in terms of $\tilde{q}$ rather than $q(\mathbf{w})$, as in (Li and Gal 2017):

$$\mathcal{L}_\alpha(\phi) = \frac{1}{\alpha}\sum_{i=1}^N \log \int \left(\frac{1}{Z_q}\tilde{q}_\phi(\mathbf{w})\left(\frac{\tilde{q}_\phi(\mathbf{w})}{p(\mathbf{w})}\right)^{\frac{\alpha}{N-\alpha}}\right)^{1-\frac{\alpha}{N}} p(\mathbf{w})^{\frac{\alpha}{N}} p(y_i|\mathbf{w},\mathbf{x}_i)^\alpha d\mathbf{w} \tag{4.2.6}$$

$$\begin{aligned} &= -\frac{N}{\alpha}\left(1-\frac{\alpha}{N}\right)\log\int \tilde{q}_\phi(\mathbf{w})\left(\frac{\tilde{q}_\phi(\mathbf{w})}{p(\mathbf{w})}\right)^{\frac{\alpha}{N-\alpha}} d\mathbf{w} \\ &\quad + \frac{1}{\alpha}\sum_{i=1}^N \log \mathbb{E}_{\tilde{q}_\phi(\mathbf{w})}\left[p(y_i|\mathbf{x}_i,\mathbf{w})^\alpha\right] \end{aligned} \tag{4.2.7}$$

$$= \frac{1}{\alpha}\sum_{i=1}^N \log \mathbb{E}_{\tilde{q}_\phi(\mathbf{w})}\left[p(y_i|\mathbf{x}_i,\mathbf{w})^\alpha\right] - \mathrm{R}_\beta[\tilde{q}|p]\,, \tag{4.2.8}$$

where $\beta = N/(N-\alpha)$ and $\mathrm{R}_\beta[\tilde{q}|p]$ represents the *Rényi divergence* of order β Rényi 1961, which is defined as

$$\mathrm{R}_\beta[q|p] = \frac{1}{\beta-1}\log\int \tilde{q}(\mathbf{w})^\beta p(\mathbf{w})^{1-\beta} d\mathbf{w}. \tag{4.2.9}$$

Importantly, when $\alpha/N \to 0$ we recover $q \to \tilde{q}$ and $\mathcal{L}_\alpha(\phi)$ converges to the objective of VI, which can be used here as a baseline. Also, we have that $\mathrm{R}_\beta[\tilde{q}|p] \to \mathrm{KL}(\tilde{q}|p) = \mathrm{KL}(q|p)$ if $\mathrm{R}_\beta[\tilde{q}|p] < +\infty$ (which is true assuming $Z_q < +\infty$ and $\alpha/N \to 0$). Therefore, following Li and Gal 2017, when this quotient tends to zero, we can make further approximations for the BB-α energy function, as described in (4.2.4), finally obtaining

$$\mathcal{L}_\alpha(\phi) \approx \frac{1}{\alpha}\sum_{i=1}^{N} \log \mathbb{E}_{q_\phi(\mathbf{w})}[p(y_i|\mathbf{x}_i,\mathbf{w})^\alpha] - \mathrm{KL}(q_\phi(\mathbf{w})|p(\mathbf{w}))\,. \tag{4.2.10}$$

This will be the objective function that we will maximize in our approach. Note that the expectations in (4.2.10) can be estimated via Monte Carlo sampling. In particular, we have that

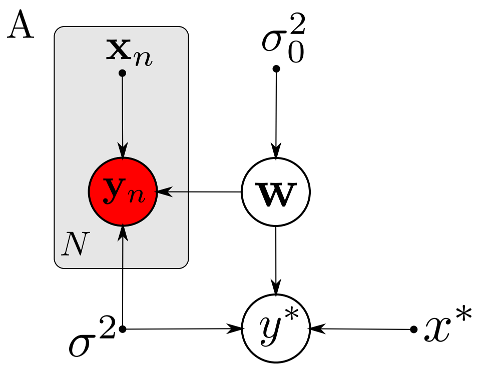
$$\log \mathbb{E}_{q_\phi(\mathbf{w})}[p(y_i|\mathbf{x}_i,\mathbf{w})^\alpha] \approx \log[K^{-1}\sum_{k=1}^{K} p(y_i|\mathbf{x}_i,\mathbf{w}_k)^\alpha] \tag{4.2.11}$$

for K samples of $\mathbf{w}$ drawn from q_ϕ. Of course, this estimate is biased, as a consequence of the non-linearity of the $\log(\cdot)$ function. However, the bias can be controlled with K. Furthermore, we expect a similar behavior as in standard BB-α, in which the bias has been shown to be very small even for $K = 10$ samples. See (Hernández-Lobato et al. 2016) for further details.

The objective in (4.2.10) has been obtained under some conditions that need not be true in practice, *e.g.* the quotient $\alpha/N \to 0$ (*i.e.*, either α is small, N is sufficiently large or a combination of both). Nevertheless, it is much simpler to estimate and maximize than the objective in (4.2.4). It is also similar to the objective functions found in the deep learning bibliography (*i.e.*, a loss function plus some regularizer, such as the KL divergence), but it still maintains the qualities of an approximate Bayesian inference algorithm. Importantly, (4.2.10) allows for implicit models for q_ϕ. The only term that is difficult to approximate is $\mathrm{KL}(q_\phi(\mathbf{w})|p(\mathbf{w}))$. However, the approach described in Section 3.3.3 for AVB can be used here for that purpose. We can simply use an independent classifier to estimate the log-ratio between $p(\mathbf{w})$ and $q_\phi(\mathbf{w})$, as in AVB. This enables using implicit distributions when maximizing the objective in (4.2.10).

By changing the α parameter of the method we will be able to interpolate between the objective function of AVB ($\alpha \to 0$) and one of an EP-like algorithm ($\alpha = 1$). Note that when $\alpha \to 0$, (4.2.10) is expected to focus on reducing the training error since the factor $\alpha^{-1}\log \mathbb{E}_{q_\phi(\mathbf{w})}[p(y_i|\mathbf{x}_i,\mathbf{w})^\alpha]$ will converge to $\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(y_i|\mathbf{x}_i,\mathbf{w})]$, with $p(y_i|\mathbf{x}_i,\mathbf{w})$ typically a Gaussian distribution with mean given by the output of the neural network and noise variance σ^2. By contrast, when $\alpha = 1$, (4.2.10) will be expected to focus more on the training log-likelihood. Intermediate values of α will trade-off between these two tasks, which may lead to better generalization properties of the predictive distribution.

The specific details of the structure of the proposed approach, AADM, are analogous to the ones described for AVB (Mescheder et al. 2017). The structure of AADM can be divided into three main components: An implicit model for q_ϕ, which takes as input Gaussian noise and outputs neural network weight samples $\mathbf{w}$ from the approximated weights posterior distribution (*i.e.*, the generator network); a discriminator, which estimates the KL term present in (4.2.10) as done in (Mescheder et al. 2017) and Section 3.3.3; and finally the main network, that uses the samples of the weights generated previously to evaluate the factor $p(y_i|\mathbf{x}_i, \mathbf{w})$. The whole system is optimized altogether. Furthermore, any potential hyper-parameter (*e.g.*, the prior variance σ_0^2 or the output noise variance σ^2) is tuned simply by maximizing the objective in (4.2.10).



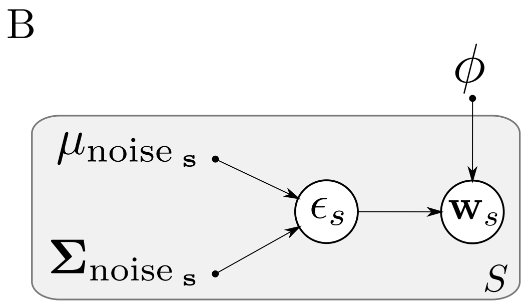


Figure 4.1: Graphical models for AADM. *Left* - Assumed probabilistic graphical model for the observed data. Point-like vertices denote deterministic variables and circular ones indicate random variables, which can either be *observed* (red) or *unobserved* (white). *Right* - Probabilistic graphical model of the implicit distribution q used to approximate the posterior. A source of S samples of Gaussian noise (we assume independence) with mean $\boldsymbol{\mu}_{\text{noise}}$ and variances $\boldsymbol{\Sigma}_{\text{noise}}$ is let through through a deep neural network with parameters ϕ to generate S samples of the weights of the main neural network. Best seen in color.

Figure 4.1 (left) shows the main probabilistic graphical model corresponding to the observed data. Point-like vertices indicate deterministic variables. Circular vertices denote random variables, which can be *observed* (red) or *unobserved* (white). Figure 4.1 (right) shows the probabilistic graphical model of the implicit approximate posterior distribution. In this case, we generate S samples of Gaussian noise in the form of $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{noise}}, \boldsymbol{\Sigma}_{\text{noise}})$, with $\boldsymbol{\Sigma}_{\text{noise}}$ a diagonal matrix. These samples are passed through a deep neural network with weights ϕ to obtain S samples for the weights of the main neural network. These weights are then used in the main network, shown in Figure 4.1 (left), to estimate the predictive distribution during training and testing.

As a last remark concerning the implementation of the proposed method, we have also included as trainable parameters both the mean and variances of the Gaussian noise which is used as input in the generator network (the implicit model for the weights, $q_\phi(\mathbf{w})$, in (3.3.32)). This allows for a more expressive implicit model for $q_\phi(\mathbf{w})$, since it increases its flexibility by enabling the tuning of the broad parameters that control its input. Using this in combination with the approximate minimization of α-divergences, the proposed method AADM is expected to reproduce to a higher degree of accuracy the original posterior distribution of the model parameters (neural network weights). Our hypothesis here is that this will lead to more accurate predictive distributions.

Finally, the proposed method AADM may suffer from convergence to bad local optima. This may happen as a consequence of the strong regularization effect of the term $\text{KL}(q_\phi(\mathbf{w})|p(\mathbf{w}))$ at the beginning of the training process. To alleviate this problem and obtain better results, we have considered also the approach suggested by Sønderby et al. 2016 that consists in adding an extra annealing parameter β that penalizes the KL term. This parameter takes value 0 at the beginning and progressively, after each epoch, it increases until it takes value 1. See the supplementary material for further details.

4.3 Adaptive Contrast

Both the performance of AADM and AVB rely on a good approximation $T_\omega(\mathbf{w})$ to the optimal discriminator, which would be the one to provide us with the correct value for the log ratio between our approximating distribution and the prior. Although in the non-parametric limit this is achieved, in practice $T_\omega(\mathbf{w})$ can fail to be sufficiently close to the optimal discriminator. This a consequence of calculating the discriminator between q_ϕ, the posterior approximation and the prior, which are often very different distributions. This results in practice in a more *relaxed* performance of the estimated discriminator, which has no problem telling apart samples from one density or the other, but that fails to correctly estimate the log-ratio between probability distributions.

Adaptive Contrast is introduced in (Mescheder et al. 2017) as a solution to this issue. It consists in using a new auxiliary conditional probability distribution $r_\alpha(\mathbf{w})$ with known density that approximates q_ϕ. This auxiliary distribution is set to be a factorizing Gaussian whose mean and variances match those of q_ϕ. Using this extra distribution, the objective defined for VI through the ELBO is rewritten as

$$\mathcal{L}(\phi) = -\text{KL}(q_\phi(\mathbf{w})|r_\alpha(\mathbf{w})) + \mathbb{E}_{q_\phi(\mathbf{w})}\left[\log p(\mathbf{y}|\mathbf{w},\mathbf{X}) + \log p(\mathbf{w}) - \log r_\alpha(\mathbf{w})\right] , \tag{4.3.1}$$

using $\mathbf{X}$ as the matrix containing all input vectors $\mathbf{x}_i$. If $r_\alpha(\mathbf{w})$ approximates well $q_\phi(\mathbf{w})$, the KL divergence between these distributions will often be much smaller than $\text{KL}(q_\phi(\mathbf{w})|p(\mathbf{w}))$, which facilitates learning the correct probability ratio.

This technique is called *adaptive contrast*, because the divergence is not being calculated between q_ϕ and the prior, but between q_ϕ and the adaptive distribution r_α. Therefore, the discriminator now estimates $\text{KL}(q_\phi(\mathbf{w})|r_\alpha(\mathbf{w}))$ and hence the log-ratio between q_ϕ and r_α. More precisely, introducing this new auxiliary distribution, the lower bound becomes

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{w})}\left[-T_\omega(\mathbf{w}) - \log r_\alpha(\mathbf{w}) + \log p(\mathbf{y}|\mathbf{w},\mathbf{X}) + \log p(\mathbf{w})\right] , \tag{4.3.2}$$

where now $T_\omega(\mathbf{w})$ approximates the optimal discriminator between samples from $r_\alpha(\mathbf{w})$ and $q_\phi(\mathbf{w})$. Moreover, the KL divergence in (4.3.1) is invariant under any change of variables. Therefore, it can be rewritten as:

$$\mathrm{KL}(q_\phi(\mathbf{w})|r_\alpha(\mathbf{w})) = \mathrm{KL}(\tilde{q}_\phi(\tilde{\mathbf{w}})|r_0(\tilde{\mathbf{w}})) \tag{4.3.3}$$

where $\tilde{q}_\phi(\tilde{\mathbf{w}})$ is the distribution of the standardized vector $\tilde{\mathbf{w}}$, whose j-th component is given by $\tilde{w}_j := \frac{w_j - \mu_j}{\sqrt{\Sigma_{j,j}}}$ (with μ_j and $\Sigma_{j,j}$ the mean and variance of w_j, respectively), and $r_0(\tilde{\mathbf{w}})$ is a standard Gaussian distribution. Therefore, the discriminator $T_\omega(\mathbf{w})$ just needs to look for differences between samples from the normalized posterior approximation and from a standard Gaussian distribution. The mean and variances of $\mathbf{w}$ under q_ϕ can simply be estimated using samples from this distribution.

4.4 Related Work

As has been mentioned earlier, obtaining uncertainty estimates associated to the predictions of machine learning algorithms is a widely spread problem. The intractabilities that arise in the inference problem and the different possible approaches for approximating the posterior have lead to several different research paths, covering many types of approximation procedures. In the sampling based approaches, the approximation is set by drawing samples from a Markov chain whose stationary distribution coincides with the target distribution. On the other hand, optimization-based methods introduce an approximate distribution $q(\mathbf{w})$ whose parameters are adjusted to match the exact posterior through the optimization of a certain objective.

Each of these approaches has, in a broad sense, a set of general advantages and disadvantages: sampling methods can be unbiased only asymptotically, but can be highly computationally expensive (Duane et al. 1987; Neal 2011; Neal 2012). Similarly, optimization-based techniques are usually limited by the definition of the approximating distribution, which is often parametric, and therefore they may lack expressiveness (Minka 2001b; Jordan et al. 1999; Graves 2011; Beal 2003; Soudry et al. 2014). The method proposed here alleviates some of the problems of these two techniques. Specifically, it allows for flexible approximate distributions and it also scales to large datasets, whereas in some of these cases, large datasets can be a burden to deal with (Hoffman 2017).

Most modern techniques for approximate inference take advantage of the speed of optimization-based methods and try to preserve the flexibility of sampling-based methods with the goal of obtaining the best results possible in terms of computational cost and accuracy of the approximation. There are, however, many different ways of combining both approaches, which is showcased by the wide variety of methods proposed. Most of them, however, rely on optimizing the KL divergence between $q(\mathbf{w})$ and the target distribution. Our approach is more general and can minimize, in an approximate way, the α-divergence, which as we pointed out before, includes the KL divergence as a particular case, as well as other divergences (*e.g.* the Hellinger distance). Recent published works suggest that modifying the KL divergence, which acts as a regularization term in the objective function, may lead to an improvement in performance, as pointed out in (Wenzel et al. 2020).

In (Titsias and Ruiz 2019) it is described how to estimate the gradient of the VI objective when using an implicit model for the approximate distribution $q(\mathbf{w})$. This gradient can then be used to maximize the objective. The method proposed there combines Markov chain Monte Carlo methods and VI. While this seems promising, its implementation is complicated since it relies on running an inner Markov chain inside the optimization process of the approximate distribution $q(\mathbf{w})$. Moreover, the parameters of the Markov chain also may need to be adjusted, depending on the probabilistic model used in practice.

Another approach for flexible approximate distributions $q(\mathbf{w})$ within the context of VI is normalizing flows (NF) (Rezende and Mohamed 2015). In NF one starts with a simple parametric approximate distribution $q(\mathbf{w})$ whose samples are modified using parametric non-linear invertible transformations that are carefully chosen. This results in a p.d.f. of the resulting distribution that can be evaluated in closed form, avoiding the problems arising from the use of implicit models for $q(\mathbf{w})$. Nevertheless, the family of transformations that can be used is limited to invertible transformations, which may constrain the flexibility of the approximate distribution $q(\mathbf{w})$.

Stein Variational Gradient Descent, proposed in (Liu and Wang 2016), is a general VI method that consists in transforming a set of *particles* to match the exact posterior distribution. The results obtained are shown to be competitive with other state-of-the-art methods, but the main drawback here is that there is a computational bottleneck on the number of particles that need to be stored to accurately represent the posterior distribution. More precisely, this method lacks a way to generate samples from the approximate distribution $q(\mathbf{w})$. The number of samples is fixed initially, and these are optimized by the method.

The work by Salimans et al. 2015 combines VI and MCMC methods to obtain flexible approximate posterior distributions. The key concept is to use a Markov chain as the approximate distribution $q(\mathbf{w})$ in VI. The parameters of this chain can then be adjusted to match the target distribution in terms of the KL divergence as close as possible. This is an interesting idea, but it is also limited by the difficulty of evaluating the p.d.f. of the approximate distribution. This is solved in (Salimans et al. 2015) by learning a backward model, that infers the p.d.f. of the initial state of the Markov chain given the generated samples. Learning this backward model accurately is a complex task and several simplifications are introduced that may affect the results.

Another approach used for approximate inference in the context of Bayesian neural networks is *Probabilistic Back-propagation* (Hernández-Lobato and Adams 2015). This method computes a forward propagation of probabilities through the neural network to then do back-propagation of the gradients. Although it has been proven to be a fast approach with high performance, it is limited by the expressiveness of the posterior approximation. In particular, the approximate distribution is restricted to be Gaussian. This means that this method will suffer from strong approximation bias, which is something we are actively trying to avoid here. The same applies to a standard application of VI in the context of Bayesian neural networks (Graves 2011; Blundell et al. 2015a).

The minimization of α-divergences in the context of Bayesian neural networks has also been addressed by Hernández-Lobato and Adams 2015. In that work it is described Black-box-α, a method for approximate inference that allows for very complex probabilistic models and that is efficient and allows for big datasets. The main limitation is, however, that the approximate distribution $q(\mathbf{w})$ must belong to the exponential family. That is, the approximate distribution has to be Gaussian, and hence this method will also suffer from approximation bias. Therefore, Black-box-α is expected to be sub-optimal when compared to the method proposed in this paper, which allows for implicit models in the approximate distribution $q(\mathbf{w})$.

The minimization of α-divergences has also been explored in the context of dropout in (Li and Gal 2017). That work considers the same objective as the one optimized by our approach in Section 4.2.1. The difference is that the approximate distribution considered by the authors of that work is limited to the approximate posterior distribution of dropout. This distribution is given by the mixture of two points of probability mass, *i.e.*, two delta functions, one of which is located at the origin (Gal and Ghahramani 2016). The flexibility of this approximate distribution is therefore very limited. By contrast, the method we propose allows for implicit approximate distributions $q(\mathbf{w})$ and is expected to give superior results.

Finally, a closely related method to ours is the one described in (Mescheder et al. 2017). This method, Adversarial Variational Bayes (AVB), which we already introduced in Section 3.3.3. One of the key ideas present in AVB is using a discriminator whose output can be used to estimate the KL divergence between the approximate distribution $q(\mathbf{w})$ and the prior. We have shown that, if the discriminator problem is constructed properly, the optimal output for the discriminator is equal to that log-ratio (see Section 3.3.3). This technique has also been considered in other works (Tran et al. 2017; Huszár 2017; Li and Liu 2016). A limitation of AVB is that the method is restricted to minimize the KL divergence between the approximate and the target distribution. Our approach, by contrast, can optimize the more general α-divergence, which includes the KL divergence as a particular case. Therefore, by changing the α parameter our method can potentially obtain better results than AVB. This hypothesis is confirmed by the experiments in the next section.

4.5 Experiments

To analyze and evaluate the performance of the proposed approach, *i.e.*, Adversarial α-divergence Minimization (AADM), we have carried out extensive experiments, both in synthetic data and on common UCI datasets (Dua and Graff 2017). Furthermore, we have compared results with previously existing methods such as VI, using a factorizing Gaussian as the approximate distribution, and AVB. AADM should give the similar results as AVB for $\alpha \to 0$. In these experiments we have also analyzed performance versus computational cost of each method on larger datasets with up to 2 million data points.

The method AADM employed in our experiments consists in the previously described three-network system. In particular, the structure we have considered for AADM (and also AVB), if not stated otherwise, is the following one: The generator network takes as an input a 100-dimensional Gaussian noise sample, with adjustable mean and diagonal covariance parameters, and passes it through 2 layers of 50 non-linear units each, outputting a sample of the weights $\mathbf{w}$. We generate 10 samples for the weights when training, and 50 samples to approximate the predictive distribution when testing. Similarly, the discriminator takes these samples of the weights as well as samples from the auxiliary distribution from *adaptive contrast*, which we also use here (see Section 4.3). and passes them through 2 layers of 50 non-linear units each to compute $T_\omega(\mathbf{w})$. Finally, the main network (*i.e.*, the model whose weights we are inferring) also consists of a 2 layer system with 50 units per layer as well. This network uses the sampled weights and the original data as input to estimate the AADM objective $\mathcal{L}_\alpha(\phi)$. Note that although the network size employed in our experiments is small, it is similar to the network size considered in recent related works (Hernández-Lobato and Adams 2015; Li and Gal 2017).

The number of training epochs and the presence (or absence) of a warm-up period depends on the dataset being used, and therefore is specified in each experiment. All non-linear units are leaky ReLU units. The code implementing the proposed approach is publicly available online [1]. All methods have been trained using stochastic optimization via ADAM (Kingma and Ba 2015). The learning rate for updating the parameters of the discriminator is set to the default value in ADAM, *i.e.*, 10^{-3}. The learning rate for updating the implicit model for q_ϕ (*i.e.*, the generator) and the model hyper-parameters (which includes the variance of the output noise and the prior) is set to 10^{-4}. Apart from this, we use the default parameter values in ADAM. The mini-batch size used is described in each experiment.

In our experiments we have evaluated 3 different performance metrics. The test log-likelihood is evaluated, as well as a metric concerning the error of the predictions (*i.e.* RMSE, in regression problems, and classification error, in binary and multi-class classification problems). We have also used a third metric in each case as well, with the goal of measuring the quality of the predictive distribution, as an alternative to the test log-likelihood. More precisely, we have used strictly proper scoring rules defined in (Gneiting and Raftery 2007). For the regression experiments we have employed the *Continuous Ranked Probability Score* (CRPS), which has a close-form expression described in (Grimit et al. 2006). On the other hand, in classification problems we have used the *Brier score*, both for binary and multi-class problems. The CRPS is the squared distance between the c.d.f. of the empirical distribution of the target variable and the c.d.f. of the predictive distribution. The Brier score is simply the squared distance between the vector of predictive probabilities for each class and a vector with a one-hot encoding of the observed class. For further information about these metrics, please see (Gneiting and Raftery 2007). In general, the smaller their value, the better, and a metric value equal to zero means a perfect predictive distribution.

[1] https://github.com/simonrsantana/AADM

4.5.1 Synthetic Experiments

To illustrate the features of the predictive distribution that the proposed approach AADM can capture, we evaluate this method on two simple regression problems extracted from (Depeweg et al. 2017). More precisely, we generate two different *toy datasets*. The first one involving a heteroscedastic predictive distribution, and the second one involving a bimodal predictive distribution.

The structure of the system employed is the one described previously. We train this system for 3000 epochs, using the first 500 epochs as the warm-up period. We repeat the experiments for different values of alpha in the $(0, 1]$. The first dataset is generated taking x uniformly distributed in the interval $[-4, 4]$ and y is obtained as $y = 7 \sin x + 3|\cos(x/2)|\epsilon$, where ϵ is normally-distributed and independent of x, *i.e.*, $\epsilon \sim \mathcal{N}(0, 1)$. Note that this dataset involves input dependent noise.The second dataset uses x uniformly distributed in the interval$[-2, 2]$ and $y = 10 \sin x + \epsilon$ with probability 0.5 and $y = 10 \cos x + \epsilon$ otherwise. The distribution of ϵ is the same as in the first dataset. Note that this other dataset involves a bimodal predictive distribution.We use 1000 data instances for training and the mini-batch size is set to 10.

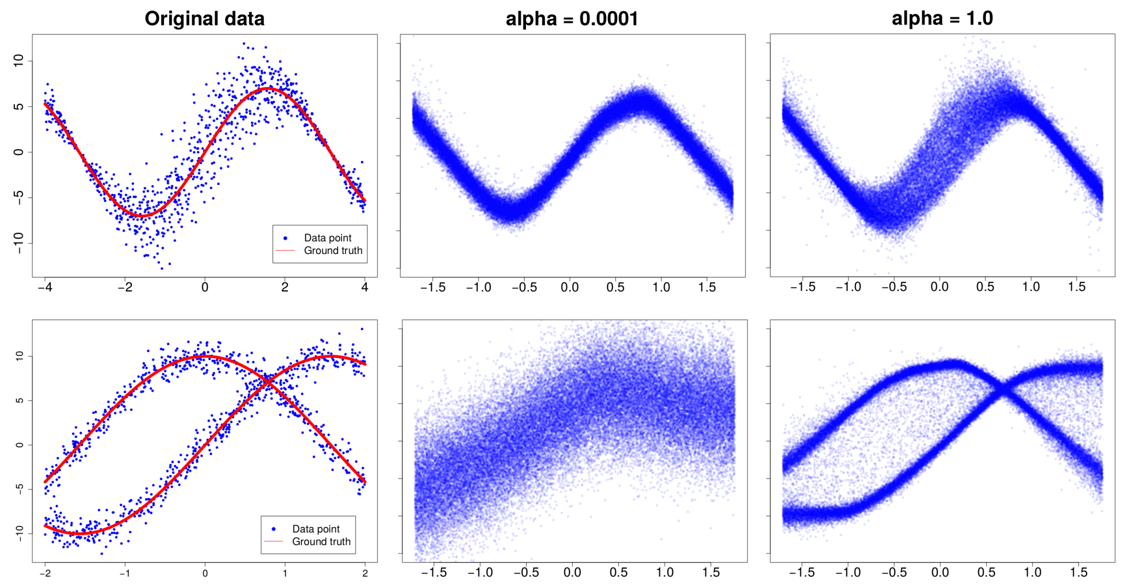


Figure 4.2: Results for the toy problems. The blue points on the left represent the original training data and the ground truth (red lines). In the middle, normalized predictions generated with $\alpha \approx 0$ (i.e. regular AVB), and in the right side are the normalized predictions with $\alpha = 1.0$.

The results obtained in the synthetic problems described are represented in Figure 4.2. The figures on at the top correspond to the problem involving the heteroscedastic

Table 4.1: Test log-likelihood, RMSE and CRPS for AADM with $\alpha = 10^{-4}$ and $\alpha = 1.0$ in both toy experiments.

	Bimodal			**Heteroscedastic**		
$\boldsymbol{\alpha}$	**L-L**	**RMSE**	**CRPS**	**L-L**	**RMSE**	**CRPS**
10^{-4}	-3.05	5.10	3.28	-2.10	1.91	1.07
1.0	-2.23	5.09	2.65	-1.91	1.94	0.97

noise and the bottom ones to the problem with a bimodal predictive distribution. On the left of the figure we show the original data we used to train AADM. In these plots, the red lines represent the *ground truth* for each dataset and the blue points are the actual samples we used as training data. The middle and right columns show normalized samples from the predictive distribution of a neural network trained using AADM, for $\alpha = 10^{-4}$ and $\alpha = 1$, respectively. The results obtained for $\alpha = 10^{-4}$ are expected to be equal to those of AVB. A low value for α is unable to capture the complex structure of predictive distribution for the target variable, ignoring features such as the heteroscedastic noise in the first task, and the bimodality of the predictive distribution in the second task. However, both of these features are captured with accuracy when α is higher, as illustrated by the results obtained when $\alpha = 1$.

As expected, choosing one value of α or another in AADM significantly changes the results obtained. In particular, when $\alpha = 10^{-4}$ the predictive distribution focuses more on minimizing the squared error and less on the log-likelihood of the data. By contrast, when $\alpha = 1.0$, the predictive distribution plays a closer attention to the log-likelihood of the data, and can hence obtain a more accurate predictive distribution. As shown in Table 4.1, although the squared error obtained when $\alpha = 10^{-4}$ and $\alpha = 1.0$ is very similar, the test log-likelihood obtained when $\alpha = 1.0$ is much better, which indicates that this value of α produces more accurate predictive distributions. Moreover, the CRPS values also improve when α is 1.0 rather than 10^{-4}. Note that the squared error only measures the expected squared deviation from target value. On the other hand, the test log-likelihood and the CRPS, measure the overall quality of the predictive distribution, taking into account, for example, features such as multiple-modes, heavy-tails or skeweness.

Finally, other values of α give similar results (not shown here). In particular, for $\alpha < 0.5$ similar results to those of $\alpha = 10^{-4}$ are obtained. By contrast, when $\alpha > 0.5$ similar results to those of $\alpha = 1.0$ are obtained (only if the training procedure is carried out carefully to avoid bad local optima).

4.5.2 Experiments on UCI Datasets

To analyze in more detail the results of the proposed method, AADM, we have considered eight UCI datasets (Dua and Graff 2017) that are widely spread for

Table 4.2: Characteristics of the UCI datasets used in the experiments.

Dataset	Instances	Attributes	Epochs
Boston	506	13	2000
Concrete	1,030	8	2000
Energy Efficiency	768	8	2000
Kin8nm	8,192	8	400
Naval	11,934	16	400
Combined Cycle Power Plant	9,568	4	250
Wine	1,599	11	2000
Yatch	308	6	2000

regression (Hernández-Lobato and Adams 2015). The characteristics of these datasets are displayed in Table 4.2. Each dataset has a different size, and in order to train the different methods until convergence we have employed a different number of epochs in each case. The number of epochs selected is presented finally in Table 4.2. Note that, even though there are differences in the epochs employed for training, all of the datasets share the same model structure, which is the general one described at the beginning of this section. In all these experiments we employ the first 10% of the total training epochs for *warming-up* before the KL term is completely turned on as in (Sønderby et al. 2016). Moreover, the batch size is set to be 10 data points, and sampling-wise, we perform 10 samples in the training procedure and 100 for testing. We split the datasets in a 90%-10% for training/testing. The results reported are averages over 20 different random splits of the datasets into training and testing.

We compare the results of AADM with VI using a factorizing Gaussian as the posterior approximation and with regular AVB (which should be the same as our algorithm when $\alpha \to 0$). For all methods we employ the same two-layered system with 50 units per layer. To make fair comparisons we also perform the same warm-up period for both AVB and VI as we use in our method. Therefore only after the first 10% of the total number of epochs, the KL term is completely activated in the objective function.

The average performance of each method on each dataset, in terms of the test log-likelihood, is displayed Figure 4.3. In this case, the higher, the better. The test log-likelihood measures the overall quality of the predictive distribution, taking into account, for example, features such as multiple-modes, heavy-tails or skeweness. We observe that values of α that are different from 0 usually outperform both regular AVB and VI in terms of this metric (the higher the values the better). From these figures, it seems that higher values of α often lead to better predictive distributions it terms of the test log-likelihood, probably as a consequence of being able to better recover the real posterior distribution. The values obtained are similar and often better than those of other state of the art methods (Hernández-Lobato and Adams 2015). Each of the values shown represent the mean performance of a certain method across the 20 different splits of each dataset, which are averaged afterwards here. Importantly, we observe that standard VI is almost always outperformed by the two techniques that allow for implicit models in the posterior approximation $q(\mathbf{w})$. Namely, AVB and AADM. This points out the benefits of using an implicit model for the approximate distribution $q(\mathbf{w})$. Moreover, AVG and AADM give almost the same results when $\alpha \approx 0$, which confirms the correctness of our implementation.

The average results obtained for each method on each dataset, in terms of the root mean squared error (RMSE) are displayed in Figure 4.4. Note that the root mean squared error only measures the expected deviation from the target value and it may ignore if the model captures accurately the distribution of the target value. We can see that the proposed approach, AADM, also obtains better results than VI. In this case, nonetheless, increasing α values do not actually improve much over the basic results of AVB, and in general we can see that lower values for α are actually better for obtaining a good performance in terms of this metric (here, the lower values the better the performance). This seems to indicate that one should choose a value for α that is different, depending on the metric they are most interested in. These results are consistent in the sense that, as pointed out previously, values of α close to zero actually lead to the objective that is optimized in AVB and VI, which pays more attention to the training RMSE. By contrast, values closer to 1.0 result in an objective function that is more closely related to the log-likelihood of the training data.

We also report the performance of each method in terms of CRPS metric (lower values are better) in Figure 4.5. The results obtained are similar to those obtained in terms of the test log-likelihood. This is the expected behavior since the CRPS metric also evaluates quality of the predictive distribution for the test data and it should be correlated with the test log-likelihood. In particular, values of α different from 0 are expected to give more accurate predictive distributions. This is confirmed by the results. It seems that the CRPS metric improves in general when α increases. However, there are few exceptions, such as those of the *Yatch* and *Naval* datasets.

4.5.2.1 Average Rank Results on the UCI Datasets

To get an overall idea about the performance of AADM, for each value of α, on the previous experiments we have proceeded as follows: We have ranked the performance AADM for each α value (*i.e.*, rank 1 means that value of α gives the best result, rank 2 means that it gives the second best results, etc.). Then, we have computed the average rank over all the train / test splits of the datasets, and have calculated the standard deviation in each case. Figure 4.6 shows the results obtained for the RMSE, test log-likelihood and CRPS.

The results obtained are displayed in Figure 4.6. This figure confirms that the intermediate values of alpha usually present a better performance than the extremes (*i.e.*, $\alpha \approx 0$ or $\alpha = 1$), for either the RMSE, the test log-likelihood metrics and the CRPS). Furthermore, both for the test log-likelihood and the CRPS, higher values of α provide better results, which means that these values of α provide more

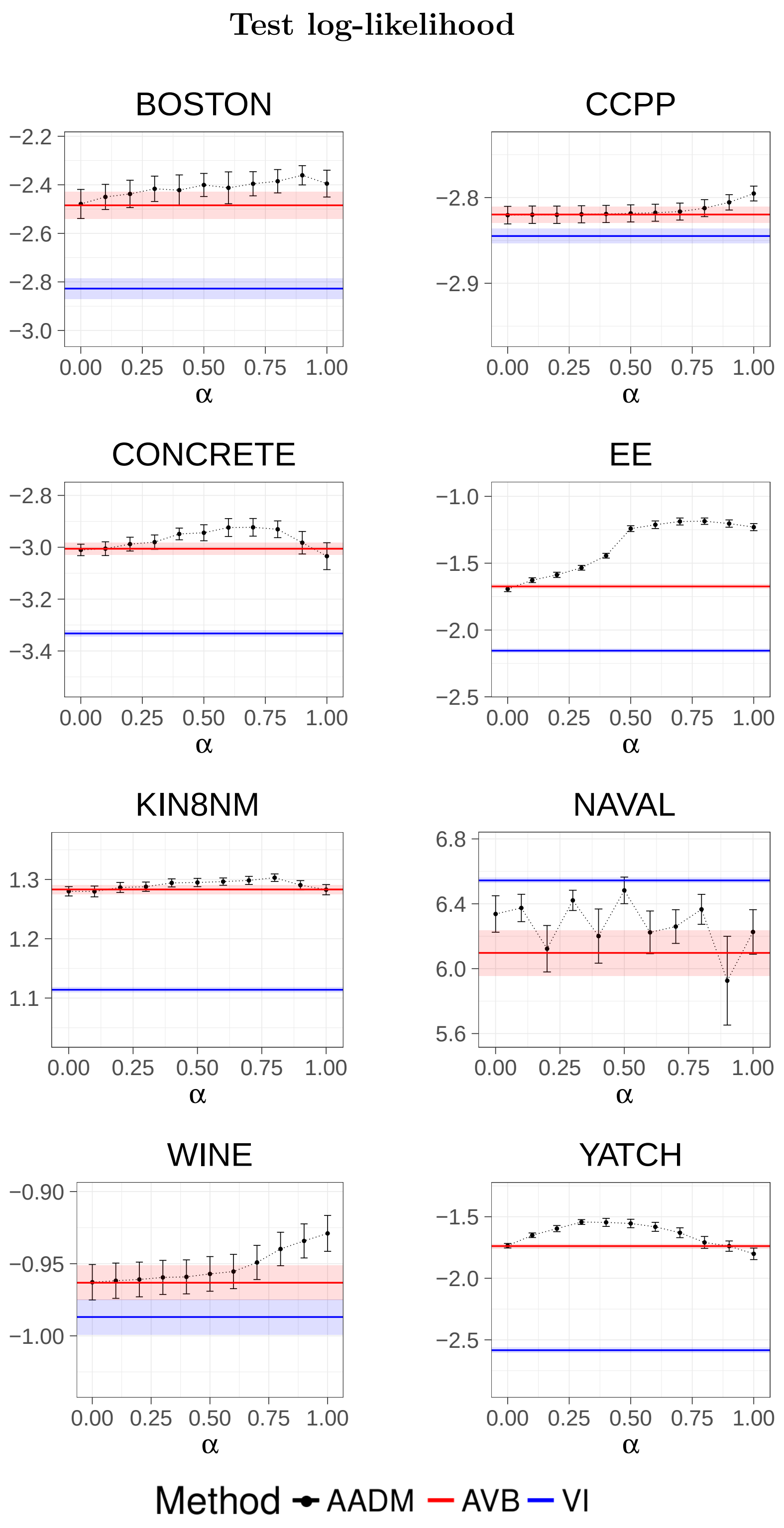


Figure 4.3: Average results in terms of the test log-likelihood for the different UCI datasets and methods compared (higher is better). Black represents the performance for our method, AADM, for different values of α. Red is the performance of AVB. VI is presented in blue. Best seen in color.

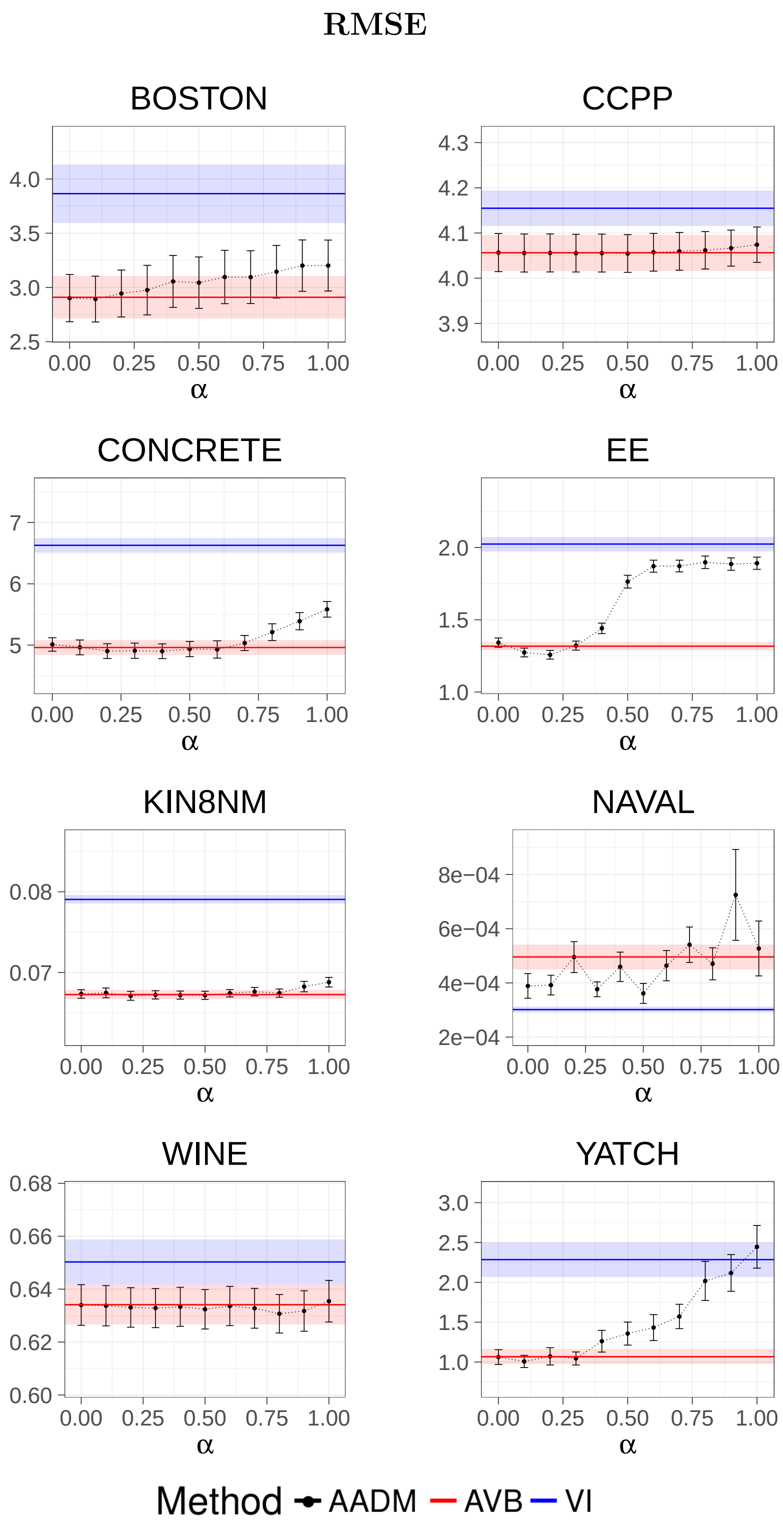


Figure 4.4: Average results in terms of the root mean squared error for the different UCI datasets and methods compared (lower is better). Black represents AADM for different values of α, red is AVB, and VI is presented in blue. Best seen in color.

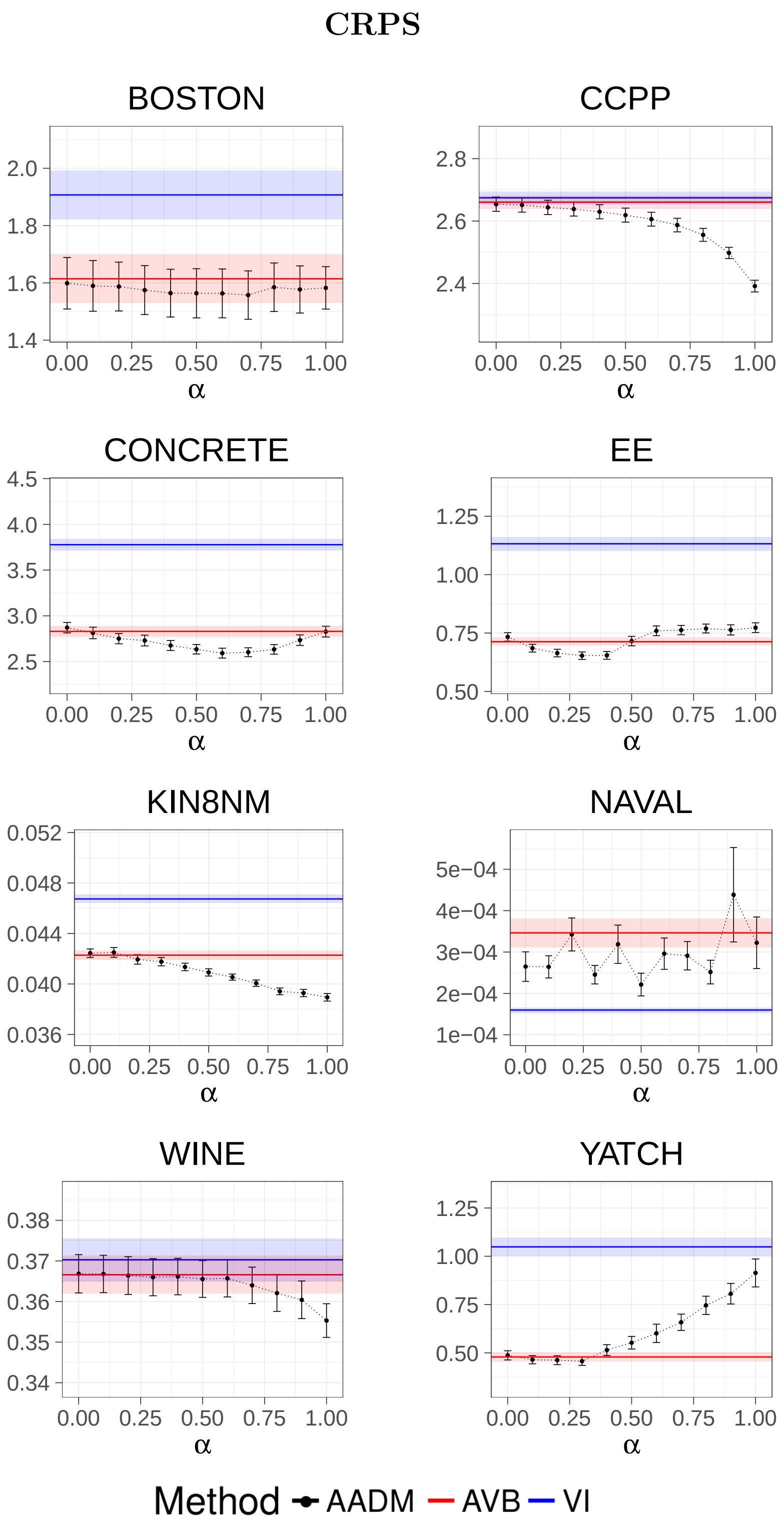


Figure 4.5: Average results in terms of the continuous ranked probability score (CRPS) for the different UCI datasets and methods compared (lower is better). Again, black here is AADM, red represents AVB and blue, VI. Best seen in color.

accurate predictive distributions. This is expected to be related to a better posterior approximation. In spite of this, lower values of α tend to perform better in terms of the RMSE (although the best results are still obtained when $\alpha > 0$). Again, this can be explained by paying attention to the form of the objective function that is maximized in both extremes, *i.e.*, for $\alpha \to 0$ and for $\alpha = 1$. Recall that the the VI objective is recovered when $\alpha \to 0$. This objective gives higher importance to the squared error since $\log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$ is precisely the squared error. By contrast, a similar objective function to the one used by expectation propagation is obtained when $\alpha = 1$. This objective includes terms that involve the log-likelihood of the training data. That is, $\log \mathbb{E}_q[p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})]$. The main conclusion from this analysis is that the optimal value for α depends on the metric we are considering, and that intermediate values of α, different from 0 or 1 can lead to better results.

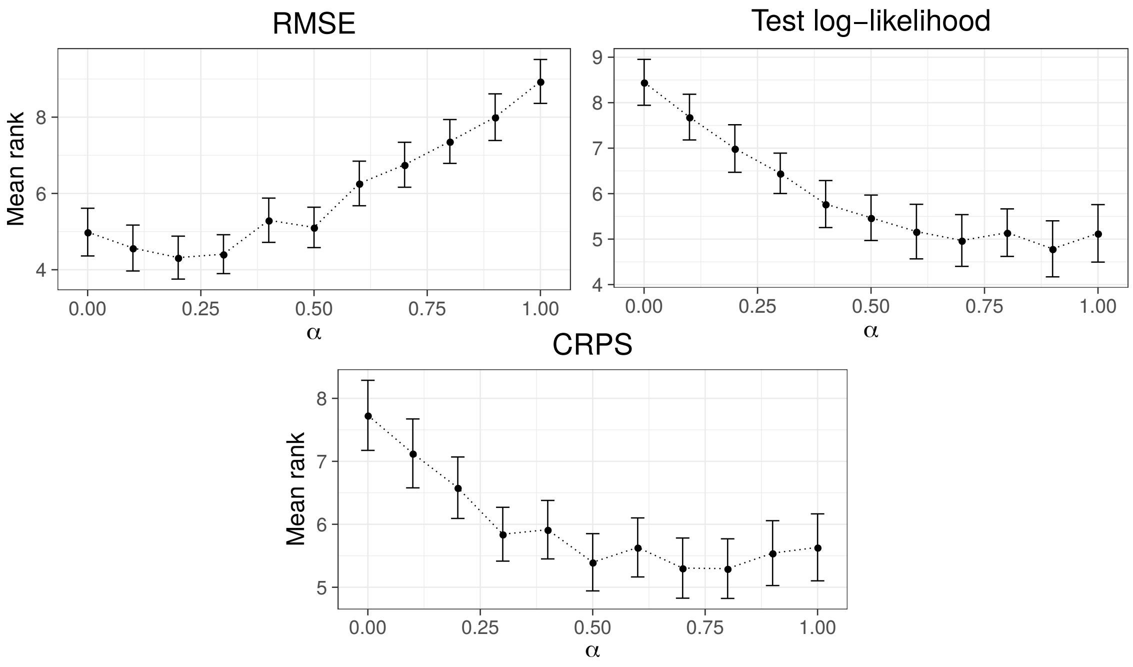


Figure 4.6: Average rank (the lower the better) for AADM and each value of α in terms of the RMSE (first row, left), test log-likelihood (first row, right) and CRPS (second row) across all the UCI datasets and splits.

A question that may arise at this point is how to choose the α value for a given task. In a broad sense, the optimal choice would strongly depend on the performance metric we are interested in. More precisely, as we have seen in the results of Figure 4.6, lower values of α tend to produce better predictive distributions in terms of the squared error, while higher values of α lead to better the predictive distributions in terms of both the test log-likelihood and the CRPS. Moreover, at the very least, $\alpha > 0$ improves the general performance of pre-existing methods, as can be concluded from Figures 4.3, 4.4 and 4.5. Our recommendation is to set α close to 0 if one is interested in low squared error and to choose α close to 1 if one is interested in capturing more general features of predictive distribution. In practice, however, one should carry out a model validation procedure (*e.g.* using cross validation) to test each value of α.

4.5.3 Binary and Multi-class Classification

We have also evaluated AADM in several binary classification tasks, and on two multi-class problems. Namely, the MNIST and CIFAR-10 datasets. The results of these experiments are found in the Appendix A. In those experiments, however, the differences among all the methods are very small. In spite of this, AADM has shown to be competitive providing slightly better results than those of VI and similar results to those of AVB.

4.5.4 Experiments on Big Datasets

To evaluate the performance of the proposed method on large datasets, we have carried out additional experiments considering two datasets: *Airlines Delay*, and *Year Prediction MSD*. Airlines Delay contains information about all commercial flights in the USA from January 2008 to April 2008 (Hensman et al. 2013b). The task of interest is to predict the delay in minutes of a flight based on 8 attributes: age of the aircraft, distance that needs to be covered, air-time, departure time, arrival time, day of the week, day of the month and month. This is hence a very noisy dataset. After removing instances with missing values, $2,127,068$ instances remain. From these, $10,000$ are used for testing and the rest are used for training. *Year Prediction MSD* is publicly accessible on the UCI repository (Dua and Graff 2017). This dataset has $515,345$ data instances and 90 attributes. Again, we use $10,000$ instances for testing and the rest of the data are used for training. In these experiments the mini-batch size has been set to 100 and we have not used the warm-up annealing scheme that deactivates the KL term in the objective of each method during the initial training iterations. For each method, we measured the performance in the test set, in terms of the RMSE, the test log-likelihood and the CRPS as a function of the training time.

The results obtained for each method on the Airlines dataset are displayed in Figure 4.7. In this figure dashed lines represent other methods, the black being AVB and the blue VI. Solid lines represent our method, AADM, for different values of alpha. The figure shows that AADM obtains better results than AVB and VI in terms of the test log-likelihood and CRPS when α approaches 1. When α is closer to 0, AADM, gives similar results to those of AVB and VI in the long term. The performance of our method w.r.t. the computational time is comparable to that of AVB. In terms of RMSE, however, large values of α seem to exhibit a more unstable behavior and in general give worse results. This is probably a consequence of this dataset being very noisy.

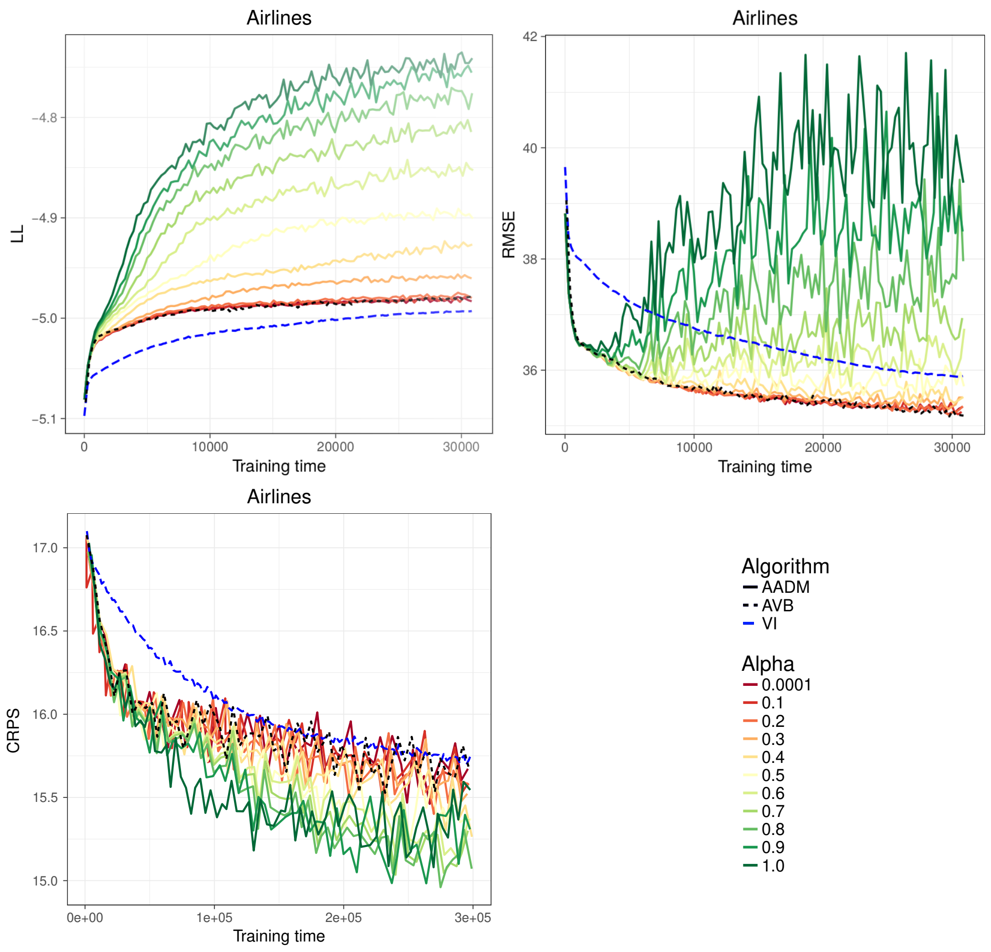


Figure 4.7: Performance as a function of the computational time in the Airlines dataset for each method. We report both in test log-likelihood (top-left), RMSE (top-right) and CRPS (bottom). The dashed blue line corresponds to the method VI, the dashed black line to AVB, and other solid lines represent our method, AADM, for different values of alpha. Best seen in color.

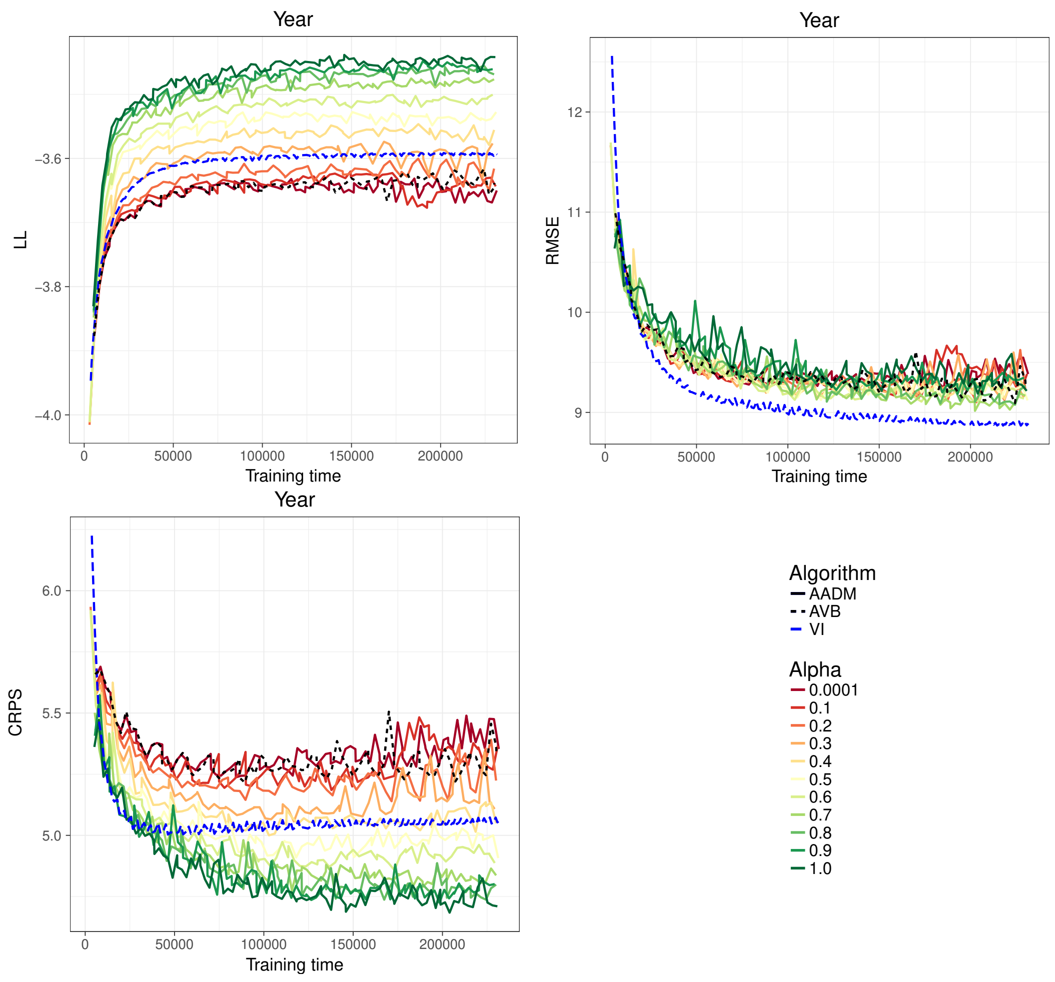


Figure 4.8: Performance as a function of the computational time in the Year dataset for each method. We report both in test log-likelihood (top-left), RMSE (top-right) and CRPS (bottom). The dashed blue line corresponds to the method VI, the dashed black line to AVB, and other solid lines represent our method, AADM, for different values of alpha. Best seen in color.

The results obtained for each method on the Year dataset are displayed in Figure 4.8. Again, in this figure dashed lines represent other methods, the black being AVB and the blue VI. Solid lines represent our method, AADM, for different values of alpha. As in the previous dataset, AADM obtains better results than AVB and VI in terms of both the test log-likelihood and the CRPS when α approaches 1. When α is closer to 0, AADM, gives similar results to those of AVB and VI. Specially, in the case of the CRPS we see that the VI performs better than both AVB and AADM, for α between 0 and 0.5. However, when α increases, AADM outperforms all of the previous methods. In terms of RMSE, lower values of α seems to give also the best results. However, in this case higher values of α do not seem to give significantly worse results in terms of this metric.

4.6 Conclusions

Here we have described a new general method for approximate Bayesian inference, Adversarial α-divergence Minimization (AADM), that is capable of tuning an approximate posterior distribution by approximately minimizing the α-divergence between this distribution and the posterior. AADM also allows to account for implicit models in the approximate posterior distribution. Implicit models, although very flexible, are difficult to work with since th p.d.f. of the implicit distribution is not known. We overcome the issues that this causes by following the approach described in (Mescheder et al. 2017). More precisely, we employ a discriminator model that estimates the log-ratio between the p.d.f. of the implicit model and a much simpler distribution (*i.e.*, a Gaussian distribution).

The proposed method has been evaluated on several experiments and compared to other methods for approximate inference such as Variational Inference (VI) with a factorizing Gaussian as the approximate distribution, and Adversarial Variational Bayes (AVB) (Mescheder et al. 2017). The experiments carried out, involving approximate inference with Bayesian neural networks, indicate that implicit models almost always provide better results than a factorizing Gaussian in terms of the metrics employed. Moreover, in regression tasks, the minimization of α-divergences seems to provide overall better results than the plain minimization of the KL divergence, as done by VI and AVB. In particular, values of α that are close, but not exactly equal to 1 seem to provide better predictive distributions in terms of the test log-likelihood and the CRPS metric. By contrast, in terms of the root mean squared error (RMSE) one should choose values of α that are close to, but not exactly equal to zero.

The approximate minimization of the α-divergence has been shown empirically to provide better results than the minimization of the KL divergence that is used in VI and AVB. More precisely, the proposed method, AADM, allows to capture patterns in the predictive distribution such as heteroscedastic noise or multiple modes. By contrast, these patterns are ignored when the typical KL divergence is minimized. This a consequence of using higher values for the α parameter that lead to a more inclusive behavior of the divergence. Specifically, higher values of α are expected to avoid that the approximate posterior distribution does not have high probability density in those regions of the parameter space in which the exact posterior has high probability density. This also follows recent results about improvements in performance caused by changing the regularization term in the inference objective function (Wenzel et al. 2020). Indeed, this increase in performance could be related, at least in appearance, to a connection between the method proposed and the *cold posterior effect* showcased by (Wenzel et al. 2020). This provides an interesting extension of this method which will be explored in future work.

Therefore, we conclude that one can obtain better results in terms of the quality of the predictive distribution (such as the RMSE, the test log-likelihood, or the CRPS) by employing the proposed method, AADM, and by choosing a value of α that may depend on the specific performance metric we are interested in. A better predictive distribution can be obtained, in terms of the RMSE, the test log-likelihood or the CRPS, by using intermediate values of α. In general, however, there is no simple way of choosing an adequate value of α for each task. Our recommendation is that if one is interested in a small prediction error, one should use small values for α. By contrast, if one is interested in more accurate predictive distributions in terms of the test log-likelihood or the CRPS, larger values for α are preferred. Ideally, one should carry out a cross validation procedure to choose the optimal value for α to choose the best value suited for each task, although these experiments may be useful as a first approach to the possible best range of values.

Chapter 5

Approximate Inference with Sparse Implicit Processes

Implicit Processes (IPs) represent a powerful tool that can be used to generalize the concepts initially introduced by GPs here. Based on previous works on this topic, we construct a new approach to conduct inference in the space of functions using IPs to model both the prior and the posterior components in the model. This is achieved by introducing a similar approximation to the one employed in sparse GPs coupled with an implicit model for the latent variables associated to the inducing points, which will keep the model efficient and scalable. This new approach represents the first method fully formulated in terms of IPs which is capable of simultaneously training both its prior and models, unlike previous approaches discussed in Chapter 3. We refer to this new approach as *Sparse Implicit Process*. Several experiments are conducted to showcase its properties in comparison to previous models. In these experiments, SIP shows to be capable of fitting both the prior and the posterior models in a sensible manner according to the presented data. Finally, through detailed regression experiments, SIP shows to be scalable and efficient while also achieving state-of-the-art performance.

5.1 Motivation

So far we have seen that approximate inference represents a widely spread research topic since it provides important features which are not common in the original ML formulation. However, most of the basic techniques employed have to deal with the intrinsic difficulties of the Bayesian formulation, such as the choice of a meaningful prior distribution or the intractability of some of the expressions needed (Bishop 2006). Moreover, for some complex real-world problems, bigger models are needed to reflect the finer details about the data. The increasing complexity of these models can result problematic as well, since the space of parameters increases in size rapidly and presents many strong correlations and local minima that ultimately complicate the process of approximate inference. All of these issues make most models covered thus far harder to fit. To do so it is required more data and solving troubles balancing different contributions throughout the training process, which can be troublesome and, in some cases unmanageable.

At first glance, the issues concerning weight space cannot be easily circumvented unless the formulation of the optimization problem is changed itself. To this end, we have introduced in Section 3.6 some recent works inside the framework of *implicit stochastic processes* (IPs) as a way to express prior distributions on the function space. Although they require a slightly more complex formulation, conducting optimization in the function space could potentially simplify the issues that arise with the original weight-space optimization. As we have seen in Section 3.6 recent methods such as *Variational Implicit Processes* (VIPs) (Ma et al. 2019) or *functional Bayesian Neural Networks* (FBNNs) (Sun et al. 2019) have only been able to show partial success in this matter: they are only able to update the prior according to the data by sacrificing the flexibility of their predictive distributions (VIP), or either can obtain more complex and flexible approximations to the posterior distribution at the expense of not training its the prior model (FBNN). Ideally, we would require a general-purposed IP-based approach to be able to do both things:

1. Update the models' prior so it comprehends important features of the dataset.

2. Provide flexible and generalized approximations for the posterior distributions.

Our contribution is a new method for approximate inference using IPs called *Sparse Implicit Process* (SIP), which fulfills both of these previous objectives. This makes SIP the first method to accomplish both goals at the same time, therefore being the first general-purposed IP-based approximate inference algorithm. We have evaluated SIP in the context of Bayesian NNs and neural samplers (Ma et al. 2019), as two illustrative examples to showcase the flexibility of our framework. In fact, IPs include many other types of models that could be used as a prior, *e.g. normalizing flows* (Rezende and Mohamed 2015), warped GPs (Snelson et al. 2004) or others. Here we perform most experiments with two different prior models, a NS and a BNN, to test both performances, as done in (Ma et al. 2019). Finally, to obtain a scalable method that can address very large datasets we consider an *inducing-point* approach in SIP in which the number of latent variables on which to perform inference is reduced from N to $M \ll N$, with N the training set size (Snelson and Ghahramani 2005). We test our claims evaluating SIP in extensive regression experiments, both in synthetic and public datasets (Dua and Graff 2017). When compared against other methods from the literature, we observe that SIP often leads to better generalization properties and that it can capture complex patterns in the predictive distribution. In very large datasets, it also remains scalable, reaching good performance levels with less computational time than other methods.

5.2 Background

As we have pointed out in Chapter 3, the usage of function-space for optimization in approximate inference may help with some of the intrinsic issues of the parameter-space formulation. Recent research (Sun et al. 2019) suggests that, when using parameter space, the optimization problem difficulty increases with the number of parameters not only due to the sheer size of the space itself, but also due to symmetries and strong correlations that may appear. By contrast, conducting this optimization in function space has proven to be challenging, but also worth the effort in order to tackle these issues. To formulate our problem, we will start by highlighting the main concepts of parameter-space approximate inference. After that, we will explain how to carry out this approximate inference in functional space using IPs.

5.2.1 Parameter-space Approximate Inference

Let us write again the data as $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, the prior distribution over the model parameters $\mathbf{w}$ as $p(\mathbf{w})$, and the likelihood function $p(\mathbf{y}|\mathbf{w}, \mathbf{X})$. As we have done earlier, the goal here is to find a distribution $q_\phi(\mathbf{w})$ to approximate the exact posterior $p(\mathbf{w}|\mathbf{y}, \mathbf{X})$. In variational inference (Jordan et al. 1999), q is found by maximizing the ELBO (3.3.4) we detailed in Chapter 3:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathbf{y}|\mathbf{w}, \mathbf{X})] - \mathrm{KL}(q_\phi(\mathbf{w})|p(\mathbf{w})) \tag{5.2.1}$$

where $\mathrm{KL}(\cdot, \cdot)$ is the Kullback-Leibler divergence between distributions. VI is based on the fact that maximizing (5.2.1) is equivalent to minimizing $\mathrm{KL}(q_\phi(\mathbf{w})|p(\mathbf{w}|\mathbf{y}, \mathbf{X}))$. Bayesian NN (BNN) models such as the one we introduced in Section 3.2.1 use a parametric distribution q that assumes independence among the components of $\mathbf{w}$ (Blundell et al. 2015a; Graves 2011). By contrast, in (Mescheder et al. 2017), or as we saw for AADM in Chapter 4, q can be made implicit, *i.e.*

$$q_\phi(\mathbf{w}) = \int q_\phi(\mathbf{w}|\boldsymbol{\epsilon})p(\boldsymbol{\epsilon})d\boldsymbol{\epsilon}, \tag{5.2.2}$$

with $\boldsymbol{\epsilon}$ as some random noise. Therefore, if $\boldsymbol{\epsilon}$ is high-dimensional and $q_\phi(\mathbf{w}|\boldsymbol{\epsilon})$ is complicated enough, the result is a very flexible approximate distribution $q_\phi(\mathbf{w})$. Note that, as we saw earlier, that although the first term in (5.2.1) can be estimated via Monte Carlo sampling, the KL contribution is intractable since q lacks a closed-form density. To solve this, we resort again to re-writing the KL term as:

$$\begin{aligned}\mathrm{KL}(q_\phi(\mathbf{w})|p(\mathbf{w})) &= \mathbb{E}_{q_\phi(\mathbf{w})}\left[\log q_\phi(\mathbf{w}) - \log p(\mathbf{w})\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{w})}\left[T(\mathbf{w})\right] ,\end{aligned} \tag{5.2.3}$$

where $T(\mathbf{w})$ is the log-ratio between $q_\phi(\mathbf{w})$ and the prior for $\mathbf{w}$, as we described in Section 3.3.3 for AVB or Section 4.2.1 for AADM. As we saw there, if the discriminator is flexible enough, its optimal value, $T_{\omega^\star}$, is exactly the log ratio between $q_\phi(\mathbf{w})$ and

$p(\mathbf{w})$, *i.e.*, $T_{\omega^\star}(\mathbf{w}) = \log q_\phi(\mathbf{w}) - \log p(\mathbf{w})$ (Mescheder et al. 2017). Using this, (5.2.1) becomes

$$\mathcal{L}(\phi) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi}[\log p(y_i|\mathbf{w},\mathbf{x}_i)] - \mathbb{E}_{q_\phi}[T_{\omega^\star}(\mathbf{w})]\,, \tag{5.2.4}$$

where the KL term is replaced by the discriminator, which is trained simultaneously to the rest of the system. This enables the use of an implicit model for for $q_\phi(\mathbf{w})$, which given that the inner transformations in the system are expressive enough, it could reproduce arbitrarily flexible distributions. Finally, besides this, instead of minimizing the regular KL-divergence between q and the posterior, we could also employ what we have presented earlier for AADM in Chapter 4 to minimize α-divergences, which includes the KL-divergence as a particular case, as we pointed out in Section 3.5.

5.2.2 Function-Space Approximate Inference

Finally, let us briefly review the main ideas behind implicit processes (IPs) which we described earlier in Section 3.6. We defined IPs (Ma et al. 2019) as a collection of random variables $f(\cdot)$ such that any finite set of evaluations $(f(\mathbf{x}_1),\cdots,f(\mathbf{x}_N))^\text{T}$ has joint distribution determined by a generative process as the one in (3.6.1). We use the notation $f(\cdot) \sim \mathcal{IP}(g_\theta(\cdot,\cdot),p_\mathbf{z})$ to indicate that f is sampled from the corresponding IP with parameters θ and $\mathbf{x}_i \in \mathcal{X}$. This definition of IPs results in a framework that is general enough to include many different models. For example, Bayesian NNs can be described using IPs if the randomness is given by the prior $p(\mathbf{w})$ over the NN weights $\mathbf{w}$. We would then sample a function parameterized by $\mathbf{w} \sim p(\mathbf{w})$, which specifies the output of the NN as $f(\mathbf{x}) = g_\theta(\mathbf{x},\mathbf{w})$ for every $\mathbf{x} \in \mathcal{X}$. θ are here the parameters of $p(\mathbf{w})$. If $p(\mathbf{w})$ is a factorizing Gaussian, θ will be the corresponding means and variances. Other important models that can be described as IPs include, *e.g.*, neural samplers (NS) or warped GPs (Ma et al. 2019; Snelson et al. 2004). Previous works using IPs for inference are the variational implicit processes (VIP) and the functional variational Bayesian NN (fBNN) (Ma et al. 2019; Sun et al. 2019). We have described both methods earlier in Section 3.6. We will summarize here the main ideas behind both approaches to provide a more complete perspective of our contributions in the following section.

First, VIPs approximate the marginal likelihood of the prior IP by the marginal likelihood of a GP. For this, an empirical covariance function is estimated by sampling from the prior IP. Namely, $f_s^\theta(\cdot) \sim \mathcal{IP}(g_\theta(\cdot,\cdot),p_\mathbf{z})$. As we saw in Section 3.6.1, the prior mean and covariances of the GP are:

$$\begin{aligned} m^\star(\mathbf{x}) &= \frac{1}{S}\sum_{s=1}^{S} f_s^\theta(\mathbf{x})\,, \\ \mathcal{K}^\star(\mathbf{x}_1,\mathbf{x}_2) &= \frac{1}{S}\sum_{s=1}^{S} \Delta_s^\theta(\mathbf{x}_1)\Delta_s^\theta(\mathbf{x}_2)\,, \end{aligned} \tag{5.2.5}$$

where $\Delta_s(\mathbf{x}) = f_s^\theta(\mathbf{x}) - m^\star(\mathbf{x})$. They then approximate $p(\mathbf{y}|\mathbf{X}, \theta)$ by $q_{\text{GP}}(\mathbf{y}|\mathbf{X}, \theta)$, *i.e.*, the marginal likelihood of a GP with the estimated means and covariances, and maximize the latter, expecting that it will increase $p(\mathbf{y}|\mathbf{X}, \theta)$ as well. To guarantee scalability and avoid the cubic cost of the GP they further approximate the GP using a linear model with the same mean and covariances. The linear model is efficiently tuned by optimizing the α-energy function, performing gradient descent w.r.t. θ via the method described in (Hernández-Lobato et al. 2016). A limitation of VIP, however, is that the final predictive distribution is Gaussian (that of a GP) which may lack flexibility.

By contrast, fBNNs rely on VI and, instead of using a GP, they use another IP to approximate the posterior of the prior IP (Sun et al. 2019). In the ELBO in (5.2.1), the first term can be estimated by Monte Carlo sampling when using an IP as the approximate posterior. However, the KL term becomes the KL-divergence between stochastic process, which is intractable. To address this, we saw in Section 3.6.2 that fBNN evaluates the KL-divergence between distributions at a finite set of points $\tilde{\mathbf{X}}$ drawn at random from the input space, leaving the ELBO as:

$$\mathcal{L}(q) = \mathbb{E}_q[\log p(\mathbf{y}|\mathbf{f}^{\mathbf{X}})] - \mathbb{E}_{\tilde{\mathbf{X}}}[\text{KL}(q(\mathbf{f}^{\tilde{\mathbf{X}}})|p(\mathbf{f}^{\tilde{\mathbf{X}}}))]\,, \tag{5.2.6}$$

where $\mathbf{f}^{\mathbf{X}}$ and $\mathbf{f}^{\tilde{\mathbf{X}}}$ are the IP values at $\mathbf{X}$ and $\tilde{\mathbf{X}}$, respectively. This objective function is then maximized w.r.t the parameters of the posterior approximate IP q. Critically, $\tilde{\mathbf{X}}$ must cover training and testing regions of the input space. Therefore, fBNN may suffer in large dimensional datasets. Moreover, fBNN is unable to fit the prior IP model by itself, as a consequence of estimating the gradients of the KL-divergence term using a spectral estimator (Sun et al. 2019).

5.3 Sparse Implicit Processes for Approximate Inference

We introduce here *Sparse Implicit Processes* (SIP), a new method for approximate inference when using IPs. In SIP we consider another IP to approximate the posterior of the prior IP, as in fBNN. However, unlike in fBNN we perform inference on finite sets of variables, as it is usually done with GPs. This avoids the problem of computing the KL-divergence between stochastic processes. SIP also allows to easily adjust the parameters of the prior IP. However, we will need to address two issues:

1. Avoid the number of latent variables increasing with the number of training points N.

2. Deal with the intractability of the computations.

The first of these two problems can be managed by considering an approximation based on inducing points, as in the sparse GPs models we introduced in Sections 2.3.3 and 3.3.1.2 (Snelson and Ghahramani 2005; Titsias 2009). Instead of making inference about $\mathbf{f} = (f(\mathbf{x}_1,), \ldots, f(\mathbf{x}_N))^{\text{T}}$ we perform inference about the process values at $M \ll N$ inducing points. We denote the set of inducing points $\overline{\mathbf{X}}$, and denote the IP values at these input locations $\mathbf{u} = (f(\overline{\mathbf{x}}_1), \ldots, f(\overline{\mathbf{x}}_M))^{\text{T}}$. Next, we focus on approximating $p(\mathbf{f}, \mathbf{u}|\mathcal{D})$, which only depends on finite sets of variables. For this, we consider the approximate posterior distribution:

$$q(\mathbf{f}, \mathbf{u}) = p_\theta(\mathbf{f}|\mathbf{u}) q_\phi(\mathbf{u})\,, \tag{5.3.1}$$

where $q_\phi(\mathbf{u})$ is an implicit distribution with parameters ϕ, and θ are the prior IP parameters. Critically, $p_\theta(\mathbf{f}|\mathbf{u})$ is fixed and $q_\phi(\mathbf{u})$ is tunable, as in the variational sparse GP approximation (Titsias 2009). Using this, the obtained VI ELBO is:

$$\begin{aligned}\mathcal{L}(\phi, \theta) &= \mathbb{E}_{q_{\phi,\theta}}\left[\log \frac{p(\mathbf{y}|\mathbf{f})\cancel{p_\theta(\mathbf{f}|\mathbf{u})}p_\theta(\mathbf{u})}{\cancel{p_\theta(\mathbf{f}|\mathbf{u})}q_\phi(\mathbf{u})}\right] \\ &= \mathbb{E}_{q_\phi}[\log p(\mathbf{y}|\mathbf{f})] - \text{KL}(q_\phi(\mathbf{u})|p_\theta(\mathbf{u}))\,. \end{aligned} \tag{5.3.2}$$

The first term can be estimated by Monte Carlo sampling, as in standard approaches for VI. However, the second term lacks any closed-form solution, since it is the KL-divergence between two implicit distributions. To estimate it we rely on the method described in Section 5.2.1 and 3.3.3, where a classifier is used to estimate the log-ratio (Mescheder et al. 2017; Rodríguez Santana and Hernández-Lobato 2020). Namely,

$$\text{KL}(q_\phi(\mathbf{u})|p_\theta(\mathbf{u})) = \mathbb{E}_q\left[T_{\omega^\star}(\mathbf{u})\right] \tag{5.3.3}$$

where $T_{\omega^\star}(\mathbf{u})$ is approximated by a NN that discriminates between samples from $q_\phi(\mathbf{u})$ and $p_\theta(\mathbf{u})$. Note, however, that $T_{\omega^\star}(\mathbf{u})$ depends on ϕ and θ. Nevertheless, as argued in (Mescheder et al. 2017), $\mathbb{E}_q(\nabla_\phi[T_{\omega^\star}(\mathbf{u})]) = 0$ if $T_{\omega^\star}(\mathbf{u})$ is optimal.

Regarding $\nabla_\theta T_{\omega^\star}(\mathbf{u})$, note that it is expected to be small when compared to $\nabla_\theta \mathbb{E}_{q_\phi}[\log p(\mathbf{y}|\mathbf{f})]$. This is an important matter, since although small, these gradients play a critical role to adjust the prior distribution to the observed data. To study the contributions made to the total gradient of the objective function by the different prior parameters, we study their effects separately. This way, we analyze the gradients w.r.t. the BNN parameters θ, and the location of the inducing points, $\overline{\mathbf{X}}$. In the case of θ, since we use a 2-layered BNN with 50 units in each layer as prior, there are a lot of parameters to choose from. In order to select among them the most relevant ones, we have chosen the 500 parameters that showed the biggest contribution to the gradient in the data term. Afterwards we perform the previous finite differences procedure and compare the gradients obtained in both terms for the same parameters. On the other hand, for the gradients regarding $\overline{\mathbf{X}}$, we estimate the KL gradient for every inducing point, since in this case we are only employing 50 inducing points. Therefore, we will be able to compare the gradients of both terms for every inducing point employed.

In Figure 5.1 we show the comparisons between the gradients of the data term in the f-ELBO and the two possible gradient contributions from the prior parameters,

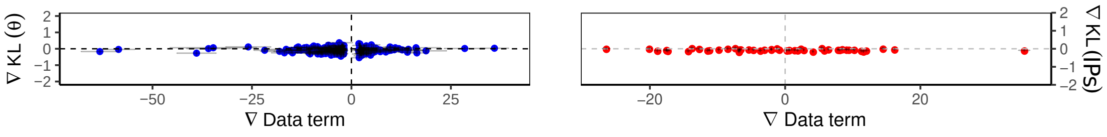


Figure 5.1: Comparison between the gradients of the data term (x-axis) and the two sets of prior parameters in the model, *i.e.* θ (left, in blue) and the inducing points' locations $\bar{\mathbf{X}}$ (right, in red). The scale of the x-axis is in both cases much bigger than the y-axis for visualization purposes. Error bars are included in both plots, although may not be visible when compared against the size of the points. Best seen in color.

i.e. θ (left image, in blue), and the inducing points' locations (right image, in red). We have included error bars, although specially in the second plot most of them are smaller than the size of the points themselves. In both figures, the x-axis represents the gradient in the data term, and the y-value for each point is the estimated gradient value for the KL term in the objective function, either for θ or for $\bar{\mathbf{X}}$. As can be seen, in both cases the x-axis has a wider range than the y-axis by a factor 10. As we mentioned, when comparing the gradient of the data term against the gradient of the KL term w.r.t. θ we selected the 500 parameters in θ that had the largest contributions to the data term, which explains the gap in $x = 0$. We see here that in every case $|y| < 1$, which means that, for the first plot, $\nabla_\theta \text{KL} \ll \nabla_\theta(\text{Data term})$. Moreover, in the second plot, we see the same behavior, meaning that $\nabla_{\bar{\text{X}}} \text{KL} \ll \nabla_{\bar{\text{X}}}(\text{Data term})$. Therefore, we obtain that $\nabla_\theta T_{\omega^\star}(\mathbf{u}) \ll \nabla_\theta \mathbb{E}_{q_\phi}[\log p(\mathbf{y}|\mathbf{f})]$, as expected.

Although $\nabla_\theta T_{\omega^\star}(\mathbf{u})$ gradients terms are small, they are of major importance to fit the prior distribution to the data. When a classifier is used to approximate the KL-divergence, these gradients will be ignored, which results in not properly adjusting the prior parameters θ. To partially correct for this we simply approximate the KL-divergence by the symmetrized KL-divergence:

$$\text{KL}(q_\phi|p_\theta) \approx \frac{1}{2}(\text{KL}(q_\phi|p_\theta) + \text{KL}(p_\theta|q_\phi)) \,, \tag{5.3.4}$$

where we have omitted the dependence on $\mathbf{u}$ of q_ϕ and p_θ. The KL-divergence in VI is just a regularizer enforcing the approximate distribution q to look similar to the prior. Modifying this regularizer in VI is a common approach that often gives better results. See, *e.g.*, (Wenzel et al. 2020). Importantly, the reversed KL-divergence, $\text{KL}(p_\theta|q_\phi)$, involves also the log-ratio between the prior and the posterior approximation q. Therefore, it can also be estimated using the same classifier $T_{\omega^\star}(\mathbf{u})$. Critically, however, it involves an expectation with respect to $p_\theta(\mathbf{u})$. This introduces some easy to compute dependencies with respect to the parameters of the prior θ. As shown in our experiments we have found those dependencies to be enough to provide some prior adaptation to the observed data. This choice is also supported by good empirical results obtained and because the prior adaptation is not observed when only the first KL-divergence term is considered (see the Appendix B.1 for an example). When changing the KL-divergence term the objective is no longer a lower bound on the log-marginal likelihood. However, this is also the case for VIP, which uses a GP approximation to the log-marginal likelihood (Ma et al. 2019).

An important remark is that the data-dependent term in (5.3.2) only considers the squared error (in the case of a Gaussian likelihood). We follow what we have done earlier in AADM in Chapter 4 and resort to the energy function employed in BB-α and VIP (Hernández-Lobato et al. 2016; Ma et al. 2019; Rodríguez Santana and Hernández-Lobato 2020). This replaces the data-dependent term in (5.3.2) resulting in the approximate optimization of α-divergences. With this re-formulation, the objective depends on α and is:

$$\begin{aligned}\mathcal{L}^\star_\alpha(\phi,\theta) = \frac{1}{\alpha}\sum_{i=1}^{N} \log \mathbb{E}_{q_{\phi,\theta}}[p(y_i|f_i)^\alpha] \\ - \frac{1}{2}\left[\mathrm{KL}(q_\phi|p_\theta) + \mathrm{KL}(p_\theta|q_\phi)\right] \,, \end{aligned} \tag{5.3.5}$$

where the first term can be estimated via Monte Carlo, using a small number of samples as in (Hernández-Lobato et al. 2016). Moreover, α can be chosen to target the data log-likelihood, *i.e.*, when $\alpha = 1$. When $\alpha \to 0$ the original data-dependent term in (5.3.2) is obtained. The bias introduced by the $\log(\cdot)$ function in (5.3.5) becomes negligible by using a small number of samples (Hernández-Lobato et al. 2016).

The other critical point of SIP is how to compute $p_\theta(\mathbf{f}|\mathbf{u})$ in (5.3.1). We approximate this conditional distribution by the conditional of a GP with the same covariance and mean function as the prior IP, as it is done in VIP (Ma et al. 2019). More precisely, given samples of $\mathbf{f}$ and $\mathbf{u}$, we employ (5.2.5) to estimate the means and covariances needed via Monte Carlo. We then resort to the GP predictive equations described in (Rasmussen and Williams 2006). Namely, $p_\theta(\mathbf{f}|\mathbf{u})$ is approximated as Gaussian with mean and covariance

$$\begin{aligned}\mathbb{E}[\mathbf{f}] &= m(\mathbf{x}) + \mathbf{K}_{\mathbf{f},\mathbf{u}}(\mathbf{K}_{\mathbf{u},\mathbf{u}} + \mathbf{I}\sigma^2)^{-1}(\mathbf{u} - m(\overline{\mathbf{X}}))\,, \\ \mathrm{Cov}(\mathbf{f}) &= \mathbf{K}_{\mathbf{f},\mathbf{f}} - \mathbf{K}_{\mathbf{f},\mathbf{u}}(\mathbf{K}_{\mathbf{u},\mathbf{u}} + \mathbf{I}\sigma^2)^{-1}\mathbf{K}_{\mathbf{u},\mathbf{f}}, \end{aligned} \tag{5.3.6}$$

with σ^2 a small noise variance (set to 10^{-5}). Moreover, $m(\cdot)$ and each entry in $\mathbf{K}_{\mathbf{f},\mathbf{u}}$ and $\mathbf{K}_{\mathbf{u},\mathbf{u}}$ is estimated empirically using (5.2.5), as in VIP (Ma et al. 2019).

Finally, predictions at a new point $\mathbf{x}^\star$ are estimated via Monte Carlo sampling. Let $\mathbf{u}_s \sim q_\phi(\mathbf{u})$. Then,

$$p(f(\mathbf{x}_\star)|\mathbf{y},\mathbf{X}) \approx \frac{1}{S}\sum_{s=1}^{S} p_\theta(f(\mathbf{x}_\star)|\mathbf{u}_s)\,. \tag{5.3.7}$$

Thus, the predictive distribution is a mixture of Gaussians, with each Gaussian determined by one sample extracted from the approximate posterior IP at $\overline{\mathbf{X}}$. This means that SIP can produce complicated predictive distributions that need not be Gaussian, unlike VIP.

To illustrate the SIP method, in Figure 5.2 we include its assumed graphical model. Here Θ represents the prior and posterior IPs parameters, θ and ϕ in the

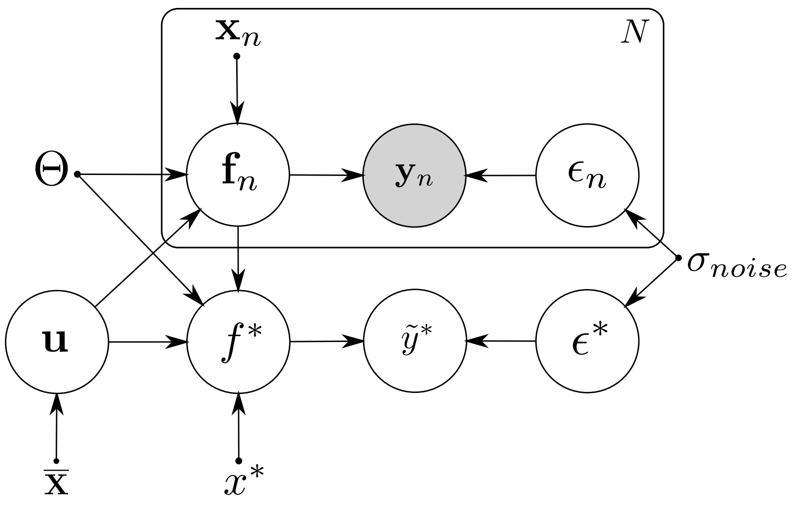


Figure 5.2: Graph model for the SIP method. Point-like vertices denote deterministic variables and circles represent random variables, which can either be observed (*gray*) or unobserved (*white*).

main text. The inducing points' locations are explicitly depicted as $\overline{\mathbf{X}}$, being their respective sampled functions values locations $\mathbf{u}$. Using all of the previous, the function values on the input data are sampled, which are $\mathbf{f}_n$ for the N training data points ($\mathbf{x}_n$, $n = 1, \ldots, N$), and f^* for the test data (x^*). Finally, after sampling the appropriate noise contribution from $\mathcal{N}(0, \sigma_{noise})$ (*i.e.* ϵ_n for training and ϵ^* for testing) we estimate the output from the system, which can either correspond to an observed ($\mathbf{y}_n$, in training) or unobserved variable ($\tilde{y}^*$, in testing).

5.4 Previous Work

As we have seen in Chapter 3, approximate inference is conducted in the parameter space for the most part. Bayesian Neural Networks (BNNs) are a notable example of this: they enable predictive distributions accounting for prediction uncertainty in the context of NN. A widely explored technique for approximate inference in BNNs is Bayes by backpropagation (BBB) (Blundell et al. 2015a; Graves 2011; Jordan et al. 1999). BBB performs variational inference (VI) in the space of weights using a parametric approximate distribution q (MacKay 1992; Graves 2011). Novel approaches are leading to new interpretations and generalizations based on VI, from which the resulting methods can have appealing theoretical properties (Knoblauch et al. 2019). However, constraining the approximate solution to a certain parametric family which in most cases assumes independence may be too restrictive in practice, as we have mentioned when reviewing the VI formalism. More precisely, in complex NNs this approach presents pathological behaviors that may lead to poor generalization properties: in (Sun et al. 2019) it is shown that bigger NN models tend to *forget* important features of the data that simpler architectures were capable of capturing.

This is due to an imbalance in the objective function that leads to a lower weight to the data-related loss, and obtaining a generalized correction for it is not trivial in most cases. Asides from the other alternatives reviewed for VI, another very successful approach for BNNs is Probabilistic Back-propagation (PBP) (Hernández-Lobato and Adams 2015). This method propagates probabilities through the NN to later perform back-propagation of the gradients. PBP has proven to be efficient and scalable, although it has limited expressiveness in the posterior approximation, which must be Gaussian. Therefore, as BBB, PBP introduces a similar approximation bias that may incur in bad-performing final predictions (Graves 2011; Blundell et al. 2015a).

Recent works have analyzed the properties of simple variational approximation methods. In (Foong et al. 2020) they are shown to underestimate the prediction uncertainty. Wide BNNs are also subject of study in (Coker et al. 2021) under the mean-field assumption. Pathological behaviors arise for deeper BNN models in the approximate posterior, which strongly differs from the exact one (see Section 3.6.2). Therefore, simple VI approximations based on, *e.g.*, mean-field should be avoided. More flexible approximations can be obtained using normalizing flows (NF) (Rezende and Mohamed 2015). NFs perform a series of non-linear invertible transformations on the variables of a tractable parametric distribution, obtaining a more complex distribution with closed-form density. For this, the transformations must be invertible, which may limit the flexibility of the approximate solution.

There has been a growing interest in increasing the flexibility of the approximate distribution (Liu and Wang 2016; Salimans et al. 2015; Tran et al. 2017). Implicit models as the ones introduced earlier have shown to be useful for this (Li and Liu 2016; Mescheder et al. 2017; Rodríguez Santana and Hernández-Lobato 2020). There, the approximate distribution lacks a closed-form density, but one can easily generate samples from it. Since q lacks a density expression, it becomes difficult to evaluate the KL-divergence term of the VI objective. AVB deals with these difficulties, as described in detail in Section 3.3.3. As a brief reminder, AVB proposes an implicit approximation model, and is capable of solving the intractabilities by introducing the auxiliary discriminator problem to estimate the log-ratio between the posterior approximation q and p (Mescheder et al. 2017). In Chapter 4 we have proposed an extension to AVB in the form of AADM, which combines the BB-α objective (Hernández-Lobato et al. 2016) with an implicit model for q and a discriminator to locally minimize α-divergences. In AADM $\alpha \in (0, 1]$ is an adjustable parameter which allows to interpolate between targeting the direct and the reversed KL-divergence. Furthermore, AADM can model complex predictive distributions, unlike AVB.

Other works include also the concept of inducing points and sparse models in the context of BNN (Immer et al. 2021; Ritter et al. 2021). However, in contrast to SIP, the location of the inducing points are fixed, and also, unlike SIP, approximate inference is carried out in the parameter space. SIP is also more general since it is not restricted to work with BNNs.

The methods described so far suffer from the problems of working in the space of parameters, which is high-dimensional and includes strong dependencies. Recent works have shown better results by performing approximate inference in the space of functions (Ma et al. 2019; Sun et al. 2019). Characterizing function-space inference, as well as constructing effective frameworks for it focuses many research efforts nowadays (Burt et al. 2020). In some of the most successful approaches, the underlying model is constructed using an implicit stochastic process (IPs), which we introduced in Chapter 3. Two successful methods for approximate inference in the context of IPs are VIP (Ma et al. 2019) and fBNN (Sun et al. 2019), both of which we described earlier in Section 3.6. VIP is limited to having a Gaussian predictive distribution, which may lack flexibility. By contrast, in fBNN is difficult to infer the parameters of the prior IP. Moreover, fBNN also relies on uniformly covering the input space to guarantee that the posterior IP looks similar to the prior IP in regions with no data. This is challenging in high-dimensional spaces. Thus, neither of these approaches successfully meet both of the criteria we established in Section 5.1. SIP does not have the limitations of VIP and fBNN and can produce flexible predictive distributions and adjust the prior parameters to the observed data.

Approximate inference in the functional space is not a new concept. Methods based on GPs have been used extensively (Rasmussen and Williams 2006), and we have seen extensively their properties and limitations in Chapter 2. In GPs the calculations can be done analytically. Nevertheless, they are restricted to Gaussian predictions. Furthermore, GPs have a big cost. Inducing points approximations, similar to those of SIP, can be combined with stochastic VI for GP scalability on very large datasets (Titsias 2009; Hensman et al. 2013a). These modifications in GPs, however, do not enable more flexible predictive distributions, which are fixed to be Gaussian.

Finally, as we saw in Chapter 3, the golden standard for Bayesian inference is provided by Markov Chain Monte Carlo (MCMC) sampling methods, since in the infinite time limit they converge to the true posterior of the model. In particular, the Hamilton Monte Carlo (HMC) or Hybrid Monte Carlo (Neal 2011) is one of the most successful approaches, since it avoids the inefficient random-walk behaviour of other MCMC methods by using the gradient information of the target distribution. Nonetheless, these methods have a few important drawbacks that prevent them from being used extensively. Among these, HMC is computationally expensive and does not scale well to big datasets, requiring also to be run for long time to obtain a fitting approximation for the posterior distribution. Moreover, it also has some sensitive paramenters that must be carefully tailored for each problem to obtain a good performance. If the model or the data being used are complex enough, using HMC and similar approaches becomes practically unfeasible.

5.5 Experiments

To test our proposal we have employed different NN models for the IPs being used. These NNs are illustrative examples to serve as IPs, although our framework can be employed with many other setups. We set two different systems for the prior: a

NS and a BNN, both of them with 2 layers of 50 units[1]. In the NS, input noise is a standard Gaussian with 10 dimensions. Moreover, in all of the experiments, The posterior implicit model in SIP uses another NS with 100 noise dimensions. In every experiment we have run each method until it converges using batch training with a batch size of 10, using 100 samples to estimate (5.3.5) and its gradients, while in test we resort to 500 samples to approximate (5.3.7). For SIP, the selected number of inducing points is specified in each experiment. Finally, we use $\alpha = 1$ in the synthetic experiments to show the capabilities of the method. For the rest of the experiments, we employed $\alpha = 0.5$ since it has been shown to be a compromise point for different types of performance (Hernández-Lobato et al. 2016; Rodríguez Santana and Hernández-Lobato 2020). Thus, we must remind that the results could be further improved by changing this parameter accordingly. In all methods the noise variance is selected by maximizing the corresponding estimate of the log-marginal likelihood.

5.5.1 Synthetic Experiments

We will first compare the quality of the predictive distribution given by SIP and other methods of the literature that also make use of IPs (VIP and FBNNs). However, we train each system using the same prior model so we can make a fair assessment of the performance of each one of them. Attending to the indications in (Ma et al. 2019), we have employed the previously mentioned BNN as our common prior. Therefore, this means that SIP, VIP and FBNNs will employ the same 2-layered BNN with 50 units per layer, using 15 inducing points in our case. We initialize them adversarially concentrated in one input location and train the model. In this case we do not learn the parameters of q_ϕ so that the model focuses on the inducing points locations and the prior for prediction, preventing $q_\phi(\mathbf{u})$ to compensate for locations with not enough inducing points nearby. We have also done the experiments for VIP with the regularization term, and also for FBNN with a GP prior, employing also an heteroscedastic dataset as well. The results of both experiments are included in Sections B.3.1 and B.3.2 of the appendix.

We also compare against Hamilton Monte Carlo applied to the same datasets and models. The prior for HMC is the BNN prior obtained from SIP$_{\text{BNN}}$ once trained. For the different synthetic datasets we set $L = 25$ integration steps and $\epsilon = 5 \cdot 10^{-5}$ leapfrog size. In this work we have found that, for all the problems in consideration, $1e4$ steps was enough to reach the equilibrium distribution.

The synthetic dataset is generated by sampling 1000 for x from a uniform distribution $\mathcal{U}(-4, 4)$. Then for each x sampled, we randomly generate one of two possible values for y generated by

$$y_1 = 10\cos(x - 0.5) + \epsilon, \qquad y_2 = 10\sin(x - 0.5) + \epsilon$$

with $\epsilon \sim \mathcal{N}(0, 1)$. We select either y_1 or y_2 randomly with equal probability, producing bimodally-distributed data. We randomly split the datasets 80% -train 20%-test

[1]code availabe at https://github.com/simonrsantana

and sample both the prior and predictive distribution after training 2000 epochs for every method to ensure convergence.

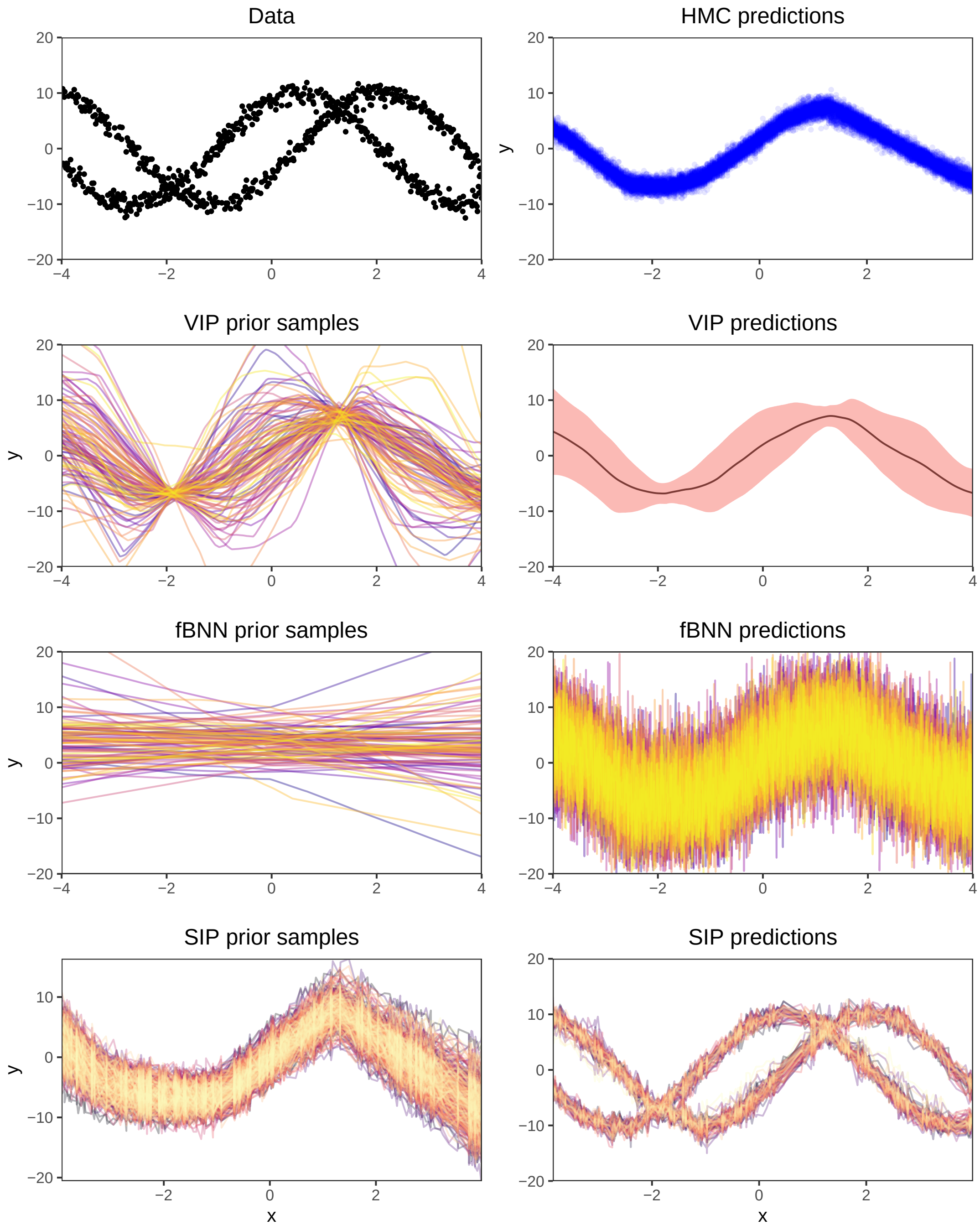


Figure 5.3: Samples from the prior and the predictive distribution of each method. First row contains the original data (first column, in black) and HMC predictions (second column, in blue). For the rest of the methods, the first column shows samples from the learned prior distribution. The second column shows the samples from the predictive distribution. Best seen in color.

Figure 5.3 show samples from the learned prior distribution and the predictive distribution for y of each method. The original training data is shown in the top left corner. We observe that in the case of VIP and SIP, the prior model captures the mean value of the training data, while in fBNN there seems to be no learned pattern at all. However, VIP's predictive distribution, which is Gaussian, is unable to represent the bimodality of the data. fBNN's predictive distribution, although more flexible than the one of VIP by construction, cannot capture the bimodality either. The reason behind this is that the data-dependent term of fBNNs focuses on minimizing the squared error, and hence it simply outputs the average prediction between the two modes, as illustrated in (Rodríguez Santana and Hernández-Lobato 2020). In summary, SIP is the only method that learns a sensible prior distribution and whose predictions capture the bimodality of the data. The appendix includes similar results for other synthetic problems.

An unexpected result in Figure 5.3 is that HMC cannot capture the bimodal predictive distribution. We believe this is simply because the assumed model (a Bayesian NN) is wrong. In particular, if one randomly generates functions from the NN prior to then contaminate them with additive Gaussian noise, the bimodal predictive distribution is never observed. A uni-modal predictive distribution is obtained instead. This is the one captured by HMC in Figure 5.3. By contrast, SIP is more flexible and thanks to the approximate inference mechanism is able to bypass this wrong model specification and produce a more accurate predictive distribution. The same behavior is observed for heteroscedastic data (see Appendix B.3.2).

5.5.1.1 Locations of the Inducing Points

The usage of inducing points is crucial to the implementation of SIP, and therefore it is important that it is capable of locating these pseudo-inputs to its advantage. To test this we have employed the dataset suggested in (Titsias 2009), using 15 inducing points as well. We initialize them adversarially concentrated in one input location and train the model. In this case we do not learn the parameters of q_ϕ so that the model focuses on the inducing points locations and the prior for prediction, preventing $q_\phi(\mathbf{u})$ to compensate for locations with not enough inducing points nearby.

In Figure 5.4 we represent the predictive distribution given by SIP, as well as the changes in positions of the inducing points across epochs (the number of epochs is scaled by $1e3$). We see that the inducing points begin in very concentrated positions close to $x = 2.7$, signaled by crosses in top of the first figure or the bottom locations of the second figure. As training continues, the inducing points locate themselves covering the whole range of the training data. After epoch 2000 they move very little except for those points more centered in the dataset, which distribute a bit more homogeneously as training advances. This shows that the method is able to successfully locate the inducing points to the most convenient positions to improve its performance (see the supplementary material for further experiments). Moreover, the predictive distribution seems to fit nicely the data even though we are preventing the system from training $q_\phi(\mathbf{u})$ here, which causes a slight underfitting behavior. For the remaining experiments, we will initiate the inducing points covering the whole range of the training data.

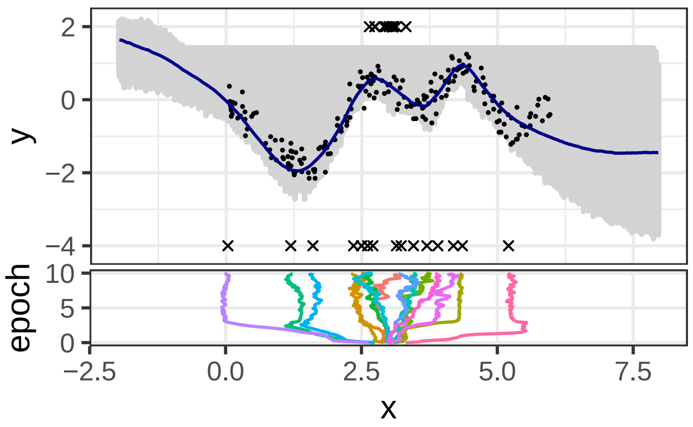


Figure 5.4: PPredictive distribution (top) and evolution of the location of the inducing points (bottom) for the dataset in (Snelson and Ghahramani 2005). The crosses at the top and bottom represent the starting and finishing positions of the inducing points, respectively. Training epochs are scaled by 10^3. Best seen in color.

5.5.2 Regression with UCI Data

We compare each method and SIP with each prior (*i.e.*, a BNN and a NS) on multivariate regression problems from the public UCI dataset repository (Dua and Graff 2017). We refer to these methods as SIP_{BNN} and SIP_{NS}, respectively. We select 8 datasets: Boston Housing, Concrete, Energy Efficiency, Kin8nm, Naval, Combined Cycle Power Plant, Wine and Yatch. We split the data 20 times into train and test with 80% and 20% of the instances, respectively. The performance metrics employed are the RMSE, the test log-likelihood (LL) and the Continuous Ranked Probability Score (CRPS). CRPS is a proper scoring rule that can be used as an alternative metric of the accuracy of the predictive distribution (Gneiting and Raftery 2007). In the case of fBNN we report results for a BNN prior since it performs better than a GP prior. In SIP we use 100 inducing points. In SIP_{NS} we implemented dropout with a rate of 0.05. SIP, fBNN and AADM use a *warm-up* period on which the KL in (5.3.5) is multiplied by a factor β that linearly increases from 0 to 1 for the first 20% of the total number of epochs. Finally, each method is trained until convergence on each dataset. A regularization term similar to the one implemented in VIP, as well as adaptive contrast can be used here. However, in our experiments we did not employ either technique. More details about this and the computational setup are given in the appendix.

To compare all methods we have ranked each one from first to last for each data split, and then averaged their rankings in each metric. The final results are presented in Table 5.1, where each value represents the mean ranking across all datasets (lower is better). For RMSE, SIP_{NS} is the overall winner, while SIP_{BNN} does

Table 5.1: Ranking analysis between methods across every multivariate regression problem (lower is better). The winning method in each metric is written in bold.

Method	BBB	AVB	AADM	FBNN_{BNN}	VIP	$\textbf{SIP}_{\text{NS}}$	$\textbf{SIP}_{\text{BNN}}$
RMSE	6.15±0.04	4.14±0.08	3.97±0.09	3.82±0.10	3.29±0.09	**3.10±0.09**	3.53±0.08
LL	5.39±0.05	3.91±0.06	2.82±0.06	5.61±0.07	3.70±0.12	3.81±0.07	**2.76±0.08**
CRPS	5.84±0.05	4.47±0.08	4.03±0.08	3.97±0.07	2.57±0.08	4.67±0.08	**2.45±0.08**

the same for LL and CRPS. As could be expected, VIP performs greatly in RMSE since its predictive distribution is encouraged to fit the mean of the data distribution. However, the NS prior for SIP achieves better results, while SIP_{BNN} occupies the third position, which makes SIP_{BNN} a competitive method in this metric as well. In LL, SIP_{BNN} wins by large margin, followed by AADM. This is also to be expected, since AADM is the only other method here able to reproduce similar behavior to what is shown in Section 5.5.1, which is the same behavior we observed in Chapter 4. However, AADM requires extra components such as *adaptive contrast*, which SIP does not need here (Mescheder et al. 2017; Rodríguez Santana and Hernández-Lobato 2020). Finally, in the CRPS, SIP_{BNN} leads again, followed by VIP. This here speaks to their respective performance is in RMSE and LL, since CRPS can be seen as a mixture between both of the previous metrics. However, the extra performance of SIP gives it the final edge to achieve first place. These results cement SIP as a sound alternative for approximate inference using IPs in terms of performance, while also providing extra features that neither VIP, FBNN or other IP-based methods are able to achieve.

5.5.3 Scalability and Convergence Comparisons

We also compared the convergence time in big datasets for all methods. We use the same datasets as in AADM: (**i**) *Year Prediction MSD*, with $515,345$ data instances with 90 attributes each, publicly available in the UCI repository (Dua and Graff 2017); (**ii**) *Airlines Delay* (Rodríguez Santana and Hernández-Lobato 2020), consisting on $2,127,068$ instances with 8 attributes. A random split of 10.000 points is used as test to measure each metric over time. For SIP employ 100 inducing points, with dropout rate 0.05 for SIP_{NS}. For VIP and FBNN_{GP}, a pre-training procedure has to be carried out, and the time for this is also accounted for in the analysis. α is fixed to 0.5, and as shown by (Rodríguez Santana and Hernández-Lobato 2020) this means that we could further improve the results for each metric in the methods with α-divergence optimization by selecting a different value.

In Figure 5.5 we have the training time in seconds in the x-axis for every plot in log-scale, and each corresponding metric in the y-axis, RMSE, LL, and CRPS respectively. The first column represents the performance results in the Airplanes dataset, and the second those of YearPrediction MSD. For every figure we have removed the first 3000s of training in order to make the final results more clearly visible. It can be appreciated in every figure that the fastest converging method is SIP_{NS}, and in the case of LL and CRPS it converges to values very close to the

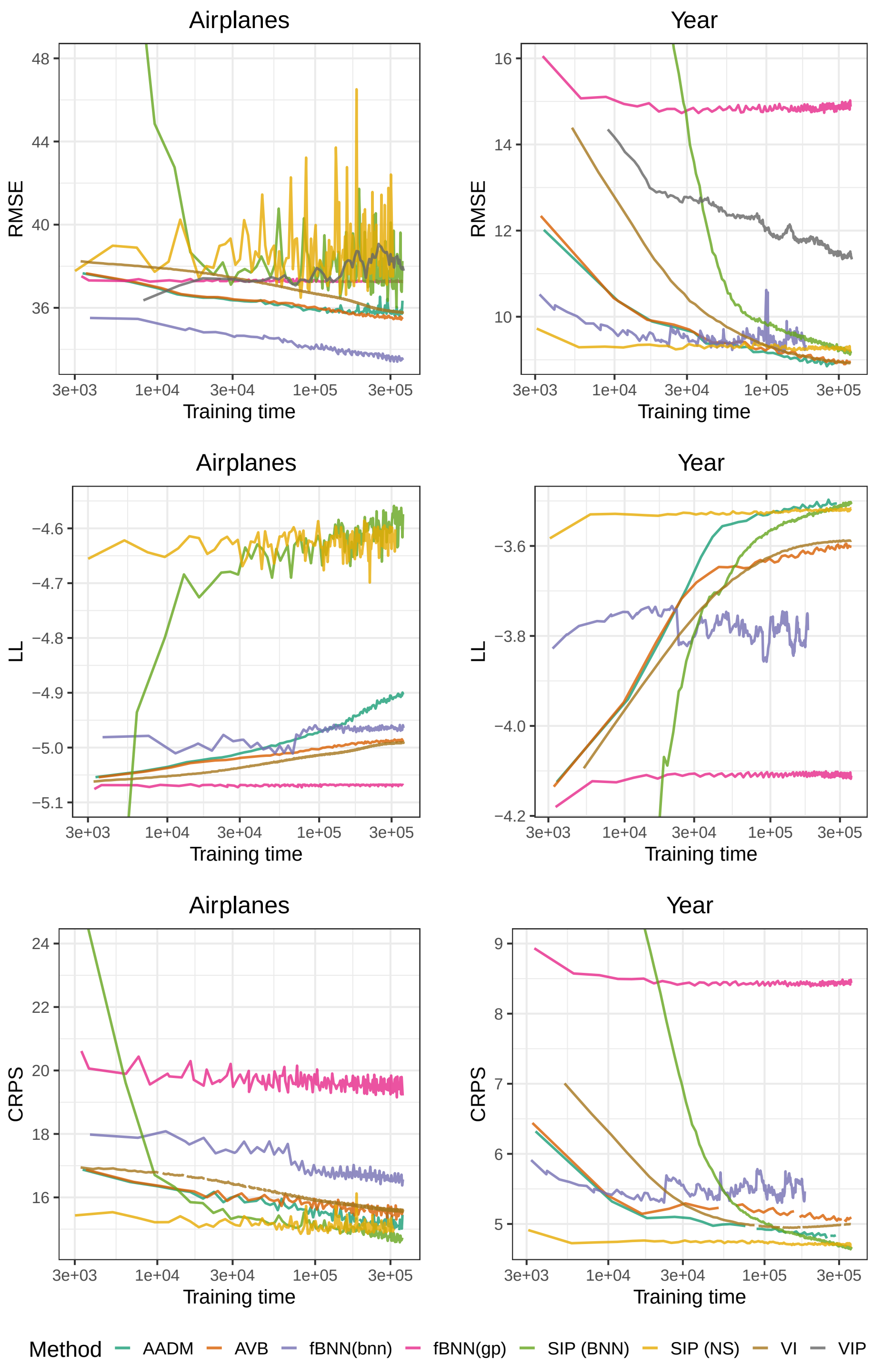


Figure 5.5: Performance as a function of training time (in seconds) in a log scale for each dataset, method and metric. Airplanes (first column) and Year (second column). The legend of each method is shown at the bottom. Best seen in color.

maximum performance in those metrics almost immediately. For the LL, SIP is the clear winner with in Airlines, although in Year AADM finishes close as well. The same happens in CRPS, where SIP wins with both priors to the rest of the methods. The faster convergence of SIP_{NS} when compared to SIP_{BNN} can be explained by the more efficient sampling taking place in the prior in this model. The rest of methods perform very similarly in general here, with FBNN_{GP} performing in general worse than the rest, far behind of FBNN_{BNN}. We associate this worse behavior to the inability of the method to properly train its prior model, leaving all of the learning to be conducted through the posterior model. Finally, in this case, VIP's performance is bad in general. Here, VIP under-estimates the noise variance, conducing to over-confident predictions that result in low CRPS and LL (much worse than the ones of the other methods and not shown in the figures for visualization purposes). Nevertheless, VIP's RMSE is good in general. SIP and AADM tend to perform better in terms of LL and CRPS than RMSE, which is targeted for medium values of α (Rodríguez Santana and Hernández-Lobato 2020) as we saw earlier (smaller α values target the RMSE more effectively).

In terms of performance vs. training time, SIP_{NS} is shown to be the fastest method among the ones being compared. However, given enough training time, SIP_{BNN} can outperform the other methods in terms of LL and CRPS, leaving AADM as the only other approach with a performance similar to SIP's. In terms of RMSE, SIP performs similarly to the other methods in Year, although the noise present in Airplanes makes it less suitable in this case. These experiments confirm again that SIP is competitive with other state-of-the-art methods, in some cases improving the performance by clear margins.

5.6 Conclusions

We have proposed SIP for approximate inference, the first candidate for a completely general-purposed approach using IPs. SIP can be used in several models (*e.g.*, Bayesian NNs and neural samplers). Moreover, it meets both of the criteria we established in Section 5.1: it can adjust the prior parameters to the data, and also use a flexible IP to approximate the posterior distribution. Current methods cannot perform these two tasks simultaneously. Importantly, SIP can generate flexible predictive distributions that capture complicated patterns in the data such as bimodality or heteroscedasticity (for an example of the latter, see the appendix).

An important extra feature we have obtained is that SIP seems much more robust to model selection than many other methods. Indeed, if the selected model for the data is wrongly chosen, the predictive distribution obtained from a method such as Hamilton Monte Carlo can be very far from the real data. In our experiment with bimodal data, this is caused by the fact that the prior distribution (shared with SIP) is unimodal, which is also the case for the posterior of the model. This conduces to unimodal predictions for HMC (and the same issue is observed when using heteroscedastic data, as shown in the Appendix). By contrast, our approximate inference method is able to circumvent this issue and produce predictive distributions much closer to that of the original data, even if the selected model may not be the most suited for the task at hand.

We have evaluated SIP on several tasks for regression. It gives similar and often better results, in terms of several performance metrics, than state-of-the-art methods for approximate inference in the context of Bayesian NNs. SIP is scalable and can be used in datasets with millions of instances. This is achieved by a sparse approximation based on inducing points similar to the one often used in GPs. Our experiments also show that SIP can learn a sensible location for the inducing points. A limitation of SIP is, however, that it requires the evaluation of complicated conditional distributions. Nevertheless, they can be approximated using a GPs with the same covariances as the prior IP. The covariances can be estimated via Monte Carlo methods.

Overall, SIP is a promising approach to perform inference using the function space. It has shown competitive performance results, scalability properties, flexibility in its formulation and robustness to the model selection. Several points remain to be explored here, such as the behavior of the inducing-point-based approach in higher dimensionality data, or whether its formulation can be simplified by the usage of alternative methods. SIP could have an important societal impact, specially when accurate predictive distributions are critical. For example, when the decisions made can have an influence on people's life, such as in autonomous vehicles or automatic diagnosis tools.

Chapter 6

Conclusions and Future Work

To finalize, we will briefly outline the main conclusions extracted from this thesis. We will review the core ideas related to approximate inference in ML, and then present a short summary of the main contributions presented to the field. We will close this final chapter outlining new interesting research paths we expect to explore in future work.

6.1 Conclusions

The supervised machine learning paradigm faces difficult questions in its implementation. Between all of the complex issues present, one of the first to appear in most cases is the selection of a learning model to describe the data. A simple model can successfully avoid issues when faced with outliers and noisy data, while may fail at capturing complex patterns intrinsic to the data observed. By contrast, more flexible models can provide better final predictions and capture the patterns that were missed earlier, although they may also be prone to fit spurious noise present in the data, therefore generalizing poorly to new data.

Several approaches have been proposed inside supervised ML that try to balance both settings, and we have introduced *neural networks* and *Gaussian processes* as examples. In general, NN models are considered flexible and versatile, and moreover their architecture can be tailored to each specific task to extract as much information from the data as possible. Nonetheless, their flexibility also makes them subject to overfitting the data. NNs also present intrinsic properties such as the automated feature detection which, despite of their drawbacks, have lead them to yield excellent results across many different applications. On the other hand, models based on Gaussian processes can be seen as simpler and more interpretable approximate inference techniques than NNs. GPs are also robust to overfitting, while also provide a simple manner to encode prior knowledge about the data. The Gaussianity restrictions they impose lead to closed-form expressions for the training procedure, although they have unscalable computational cost in big datasets and are only capable of producing normally-distributed predictions. Even if the scalability issues are addressed, the restrictive behavior of their predictions is hard to get rid of.

Additionally, another complicated subject inside supervised machine learning has to deal with making predictions in sensitive problems, specially if decisions must be made based on them. In such problems, it can be crucial to provide more complete information, including the certainty of the automated methods about their forecasts. Ideally, the estimated uncertainties may encompass the doubt on different parts of the forecasting procedure, ranging from the intrinsic uncertainty present in the measured data to the more systematic ambiguities introduced by the formulation of the model. This adds an extra layer of complexity to the choice of model itself, and one that will also have strong effects on the final results. Bayesian statistics provides here a natural framework from which we can try to address all of these issues, while also granting extra features which can result of major importance in a wide arrange of practical cases. However, although the Bayesian formulation may be promising, its practical implementation poses serious challenges related to the intractability of the equations involved in most ML approaches.

The Bayesian language enables the possibility of encoding *a priori* information about the system and update our comprehension of it according to observed data in the inference process. This scheme has proven to be very efficient to extract more information in cases where few data points are available, while also providing a simple way for controlling learning systems in cases where the data is noisy or we may need a more robust manner to interpret it. In particular, when referring to supervised ML, the Bayesian perspective allows us to encode information in the formulation of our model, and update its parameters according to data thereafter. Nevertheless, conducting inference in this context usually requires performing intractable calculations related to obtaining the posterior probability of the model's parameters according to the presented data. To avoid this issue, different techniques have been proposed to obtain good approximations for the posterior, and with that another trade-off relation appears; indeed, if the approximation models are simple enough, closed-form expressions can be obtained, although these will most likely incur on oversimplifying the approximating distribution, compromising the final results of the inference method. On the other hand, if the models flexible enough, they may provide approximations that closely resemble the target posterior distribution. However, this could come at the expense of a higher computational cost, which may become prohibitive for certain practical issues. This dichotomy is exemplified by two of the large families of approximate inference techniques, both *sampling-based* or *optimization-based* approaches.

Among the presented approaches to approximate inference, optimization-based methods that rely on VI or EP show promising results, yet in most cases their flexibility is restricted by the formulation itself. As we mentioned, these approaches propose simple distributions to approximate the more complicated exact posterior distribution that we need to perform inference. In these cases, the calculations are made tractable thanks to the simple forms of the approximations, although the final results suffer from this oversimplified treatment. In fact, in most real scenarios, the final posterior distribution is rarely as well-behaved as these approaches assume, and therefore crucial information may be lost.

There are a number of methods proposed to improve the versatility and expressiveness of these optimization-based approximations. One of the main paths that lead to improvements in this matter is the usage of *implicit models*, where the predictive distribution's p.d.f. remains unknown but samples from it can be obtained easily. This is well exemplified by the AVB approach, although its formulations prevents it from reproducing complex patterns that may be relevant in real-world problems. This is clearly shown when data presents more *complex* features, *s.a.* bimodality, heteroscedasticiy and heavy tails. As it is the case for many other optimization-based approximation approaches, AVB relies on the minimization of a divergence measure between the proposed distribution and the exact posterior. The KL divergences used in these cases can be seen as two particular examples of the larger family of *α-divergences* (also called *Renyi divergences*), whose properties remain largely unexploited in the context of ML performance.

Based on previous advances in the usage of implicit models, and in combination with the concept of α-divergences, we have proposed a new method with novel properties, while also showing important properties of α-divergences inside the context of the performance of ML algorithms. Our new method, named *Adversarial α-divergence minimization*, has the ability of capturing complex patterns such as multimodality and heteroscedasticity, removing the constraints present in previous formulations. To this end, the usage of α-divergences is crucial to be capable of capturing patterns in the data that are ignored otherwise (as showcased by the difference in performance between AVB and AADM). The original setup for the VI-like objective function can be easily recovered in the lower regimes of α, while higher values for it provide an objective closer to that of EP. Moreover, we show that there is a convex behavior depending on α for the proposed metrics. We are able to deduce that lower values help improve the squared error, while higher values make the method follow more closely the original distribution of the data, reducing the log-likelihood results. In general, we show that better results can be obtained with small changes in the method by introducing this new set of divergences and selecting α between 0 and 1. The precise value of α that may provide the best results is greatly dependent on the task, and thus cross-validation techniques could be used to select the optimal value. However, our contribution here may be of use as a first approach to the best possible range of values selected initially.

Another important addition to the approximate inference perspective is conducting optimization in the function space rather than in weight space. Works in the literature have shown that training large systems the space of parameters can lead to undesirable behaviors that are intrinsically related to the formulation of the optimization problem. Since the original optimization space is complex, with many local minima and strong correlations between parameters, a proposed solution here is optimizing functions rather than individual parameters. For this task, *implicit processes* has been an interesting tool that can encompass a wide variety of models, ranging from the Bayesian NNs and Neural Samplers we described in Chapters 3, 4 and 5 to Normalizing Flows (Rezende and Mohamed 2015), warped GPs (Snelson et al. 2004), and others (Ma et al. 2019). However, current approaches to this problem fail to formulate a complete method that successfully make use of all their properties. In the most relevant cases, the resulting method is either unable to train its prior distribution model, or its predictions are approximated by a Gaussian process, therefore losing much of the expressiveness granted by the usage of IPs.

We have proposed the first general-purposed algorithm based on IPs that is capable of training both its prior an posterior model, while at the same time providing flexible predicting distributions. This means that our method, which we named *Sparse Implicit Processes* (SIP), is capable of updating all of its parts simultaneously in a sensible manner according to the data, while also providing expressive predictive distributions that may include complex features such as bimodal behavior, heteroscedasticity or heavy tails. Previous methods based on function-space optimization were unable to do both of these things at once, which is why we think our contribution here is significant. To achieve this, we formulated an IP approximation for the prior, as well as an IP-based posterior. The posterior approximation results from the combination of ideas present in sparse GPs and implicit models, retaining the initial versatility and expressiveness of the IP approach, while also staying scalable in cases with large amounts of data thanks to the usage of inducing points. Through experiments with synthetic data we have shown that this new method is capable of efficiently locating the inducing points across the whole range of the training data, and that is also able to reproduce complex features such as the ones we mentioned above. In addition, in cases where data has higher dimensionality, SIP is expected to perform well thanks to its scalability properties and by using the appropriate number of inducing points. In some cases, we show that even if the model selected is not completely suitable for the data, the final predictive distribution can be much closer to the original data distribution thanks to the nature of the approximations made. This makes SIP more robust to the choice of predictive model, which is not the same case for other approaches such as HMC, much more sensitive to choosing the wrong model. Moreover, we have conducted extensive experiments in public datasets, showing that SIP overall improves the previous achievements by IP-based methods (both in terms of performance and scalability), while also introducing a series of new properties never before seen in this context.

6.2 Future Work

Here we will briefly outline some possible future research lines that we find specially relevant inside the topics of this thesis:

- **New methods to estimate the KL divergence:** One of the core ideas behind both of our contributions is related to the usage of an adversarial problem to estimate the KL divergence. Estimating the KL through the adversarial system setup is truly convenient since we mostly deal with complex implicit distributions as out approximate posterior q. Although useful, this can also be tricky, since it introduces the extra parameters of the auxiliary problem, and moreover may be subject to issues in the optimization procedure, as we saw in Chapter 4. It would be truly helpful here having different methods to estimate either the objective function overall or the KL divergence when intractable approximate distributions are employed, which is currently a very active line of research. This includes works from the past few years regarding likelihood-free VI (Tran et al. 2017) and unbiased implicit VI (Titsias and Ruiz 2019). These approaches could help simplifying the formulation of the methods presented here, while also potentially enhancing their performance.

- **Test the AADM approach in other contexts:** We have conducted extensive experiments with AADM inside the context of NNs. From these, we have seen that it provides great results at practically the same cost as previous methods, while also resulting in much more expressive predictive distributions. However, since the formulation of AADM allows it to be implemented in other models, it remains to be seen how would it fend in other contexts, such as GPs and *deep GPs*. Although we have not delved into deep GPs, they can be described as deep belief networks based on GP mappings (Damianou and Lawrence 2013). AADM could prove to be useful for both of these approaches, especially in cases where they are combined with implicit models, such as the work in (Yu et al. 2019). These are only a couple examples among many others, and it would be useful and interesting to examine how would AADM impact their performance. Moreover, the behavior we have obtained through AADM can be related to some recent works that point out that changes in the regularization term in the objective function can positively impact the performance of inference methods (Wenzel et al. 2020). This may lead to a connection, at least in appearance, between AADM and the *cold posterior effect* of (Wenzel et al. 2020). This provides another new path to explore further implications of the α-divergence minimization proposal we have formulated with AADM.

- **Alternative models for inference with IPs**: Implicit processes are powerful tools for approximate inference, since a wide variety of models can be described using the same language (Ma et al. 2019). However, the formulations derived from them can be rather complex when using implicit models such as in the case of SIP. A possible way to take advantage of the flexibility of this formalism is employing alternative methods to use as IPs in the inference procedure. One of the most interesting candidates here are *Normalizing Flows* (NFs), firstly introduced by Rezende and Mohamed 2015. Here, a series of invertible transformations are applied to a source of Gaussian noise, obtaining tractable arbitrarily complex distributions as a result. Due to these properties, NFs have become the subject of intensive research (Kobyzev et al. 2020). Using NFs could enable us to obtain tractable distributions in the parts of the models where an IP is used, maybe avoiding the usage of inducing points altogether and simplifying the formulation of these functional methods. Additionally, it could be interesting trying to combine these IP-based approaches to function-space optimization with other novel proposals such as *functional neural processes* (Louizos et al. 2019).

On the other hand, the inducing-point-based approach employed in SIP can have its limitations when dealing with high-dimensional data. In these cases, it is possible that many inducing points may be needed to properly fit the model, which may entail a higher computational cost than what we have seen. It would be important to study the performance of SIP in those frameworks, trying to propose new solutions for these cases.

- **Deep IPs**: As we mentioned earlier, deep GPs can be seen as a combination between the concepts of NNs and GPs, where the deep GP is represented by a deep network where every node corresponds to a GP mapping (Damianou and Lawrence 2013). Deep GPs constitute an interesting line of work, and there has been important efforts recently into extending their formulation (Bui et al. 2016a; Salimbeni and Deisenroth 2017). An interesting contribution here would be substituting GPs in the deep network by IPs, potentially obtaining a more general and expressive model that could be useful in practical scenarios as well. This will come at the expense of a more complex mathematical formulation, although it could provide important results by removing the Gaussianity assumption present in GPs.

- **Variable selection methods:** GP-based variable selection methods rely on using an automated relevance measure, obtained using the inverse of the length scale of GP models for each input variable as proxy. Newer methods propose using alternative measures of relevance, for example employing the KL divergence between different predictive distributions obtained by producing small changes in the training points (Paananen et al. 2019). These approaches could be extended in a number of ways, ranging from the choice and sensitivity of the divergence measure to the GP model selected for the variable selection in the first place. Employing IPs could help obtaining better results here, while also introducing α-divergences may contribute as well.

- **Application of flexible approximate inference methods**: The additional versatility and expressiveness achieved in the approximate inference methods here could potentially be of use for many different real-world problems. In particular, the usage of IPs could provide more accurate predictive distributions, which may help in the decision making process. As an example, GPs and regular DNN models are the normal approaches when it comes to renewable energy production forecasts (Díaz-Vico et al. 2017; Salcedo-Sanz et al. 2014; Chen et al. 2013). This can be further refined by employing IPs. Other interesting example is the usage of similar models in the neuroscience context (Challis et al. 2015; Li et al. 2019; Morabito et al. 2017). This represents another complicated problem with sensitive consequences where the extra information provided by approximate inference techniques can help advance further current research.

Appendices

Appendix A

Appendix for Chapter 4

A.1 Annealing Factor

The proposed method may suffer from convergence to bad local optima. More precisely, paying too much attention to the KL term early may result into failing to explain the observed data. Therefore, it is convenient to bias the training of the method in such a way that, at least at the beginning, it does not consider relevant the KL term. If that is the case, it will not try to make the approximate distribution look like the prior during the first steps of the optimization process, hopefully avoiding bad local optima.

To solve this, we incorporated the technique described in (Sønderby et al. 2016). Using this as an example, we define a sufficiently large *warm-up* period for which we will train our model *turning on progressively* the KL term in the objective function. We do this by changing slightly the original formulation of of the AADM objective to introduce an extra *annealing parameter* β. That is,

$$\mathcal{L}_\alpha(\phi) \simeq \frac{1}{\alpha}\sum_{i=1}^{N} \log \mathbb{E}_{q_\phi(\mathbf{w})}[p(y_i|\mathbf{x}_i, \mathbf{w})^\alpha] - \beta\mathrm{KL}(q_\phi(\mathbf{w})|p(\mathbf{w}))\,, \tag{A.1.1}$$

where β starts being equal to 0 and grows linearly to 1 during a certain number of epochs. If not stated otherwise, in each experiment the number of warm-up epochs is selected to be the initial 10% of the total epochs assigned for training the algorithm in a given dataset (mostly those extracted from the UCI repository). We have observed that this significantly improves the results. In the case of the synthetic problems, however, the warm up period is set to 500 epochs from a total number of 3000 epochs. In the experiments with big data we have not included the annealing factor since it has not been observed to be beneficial to the final results.

A.2 Classification Experiments

To analyze AADM in a more complete manner we have also carried out experiments on binary and multi-class classification problems. For the former we employed six

UCI datasets that are very common in the literature. In the case of multi-class classification, we have used the MNIST and CIFAR-10 datasets. We will start by analyzing the results on the binary case first, and finish with the more complex multi-class problems.

In the experiments carried out in this section we have not used the CRPS metric since this metric is suited only for regression problems. Instead, we made use of the Brier score (Gneiting and Raftery 2007). This is a strictly proper scoring rule that is well defined and broadly used both for binary as well as multi-class classification tasks. See (Gneiting and Raftery 2007) for more information. The brier score is simply the average squared difference between the predicted probabilities and a one-hot encoding of the target class label. Of course, we also report here prediction error and test log-likelihood, as in the regression setting.

A.2.1 Binary Classification

We consider 6 different datasets that are often used for binary classification benchmarking (Bui et al. 2017; Dua and Graff 2017). We have performed the same analysis done in the case of regression problems. More precisely, we have defined 20 splits for each dataset, trained each method on each split and afterwards we have analyzed the average results across splits in terms of the test log-likelihood, the classification error, and the Brier score. In each dataset we have trained the methods for a total of 3000 epochs, ensuring convergence. We have also split them using 90%-10% of data for training and testing, respectively, as we did for the regression experiments. All of the datasets share the same model structure, which is the general one described in the main article. In all these experiments we employ the first 10% of the total training epochs for *warming-up* before the KL term is completely turned on as in (Sønderby et al. 2016) (see the description of the *annealing factor* above for more details on this). Moreover, the batch size is set to be 10 data points. The number of samples from the posterior distribution is set to 10, when training, and to 100, when testing. For more precise information about the datasets employed see Table A.1. The likelihood factors in this particular case of binary classification are simply sigmoid activation functions.

As in the regression case, we compare the results of AADM with VI using a factorizing Gaussian as the posterior approximation and with regular AVB (whose results should be similar to the ones obtained with AADM when $\alpha \to 0$). To make fair comparisons we also perform the same warm-up period that we use in our method for both AVB and VI.

The average results for each method on each dataset, in terms of the test log-likelihood, are displayed in Figure A.1 (the higher the result the better). When compared to the results of regression, we observe that in this case changing the value of α does not have a clear impact on the final results. For these binary classification datasets, the different approximating distributions that are obtained for each value of α seem to produce similar test log-likelihood results across all splits. If we take into account previous work such as the one presented in (Bui et al. 2017), we observe

Dataset	**N for train/test**	**Attributes**	**Positive/negative instances**
Australian	621/69	15	222/468
Breast	614/68	11	239/443
Crabs	180/20	7	100/100
Iono	315/35	35	126/224
Pima	690/77	9	500/267
Sonar	186/21	61	111/96

Table A.1: Characteristics of the UCI datasets used in the experiments.

that the results achieved by AADM here, in terms of the test log-likelihood, are either similar or better than the ones reported there. In all experiments, VI has the worse results, even in datasets such as *Australian* or *Breast*. The width of the error-bars relative to the mean result in the latter case appear to be substantially wider than for the rest of the datasets. In general it is not possible to extract definite conclusions on whether AVB or AADM perform better overall, although the results in *Sonar* and *Iono* may lead us to think so. Nevertheless, the performance of AADM for all the α values is similar to that of AVB, being almost the same when $\alpha \to 0$, as expected.

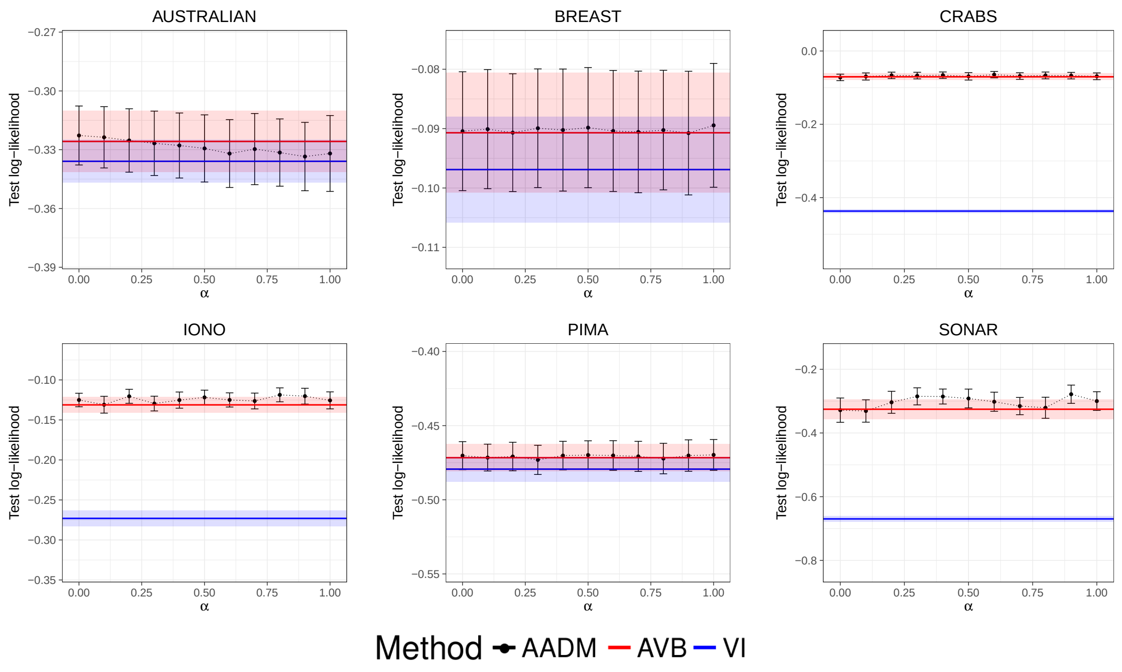


Figure A.1: Average results in terms of the test log-likelihood for the different UCI datasets for binary classification to compare the methods. Black represents the performance for our method, AADM, for different values of α. Red is the performance of AVB. VI is presented in blue. Higher values are better. Best seen in color.

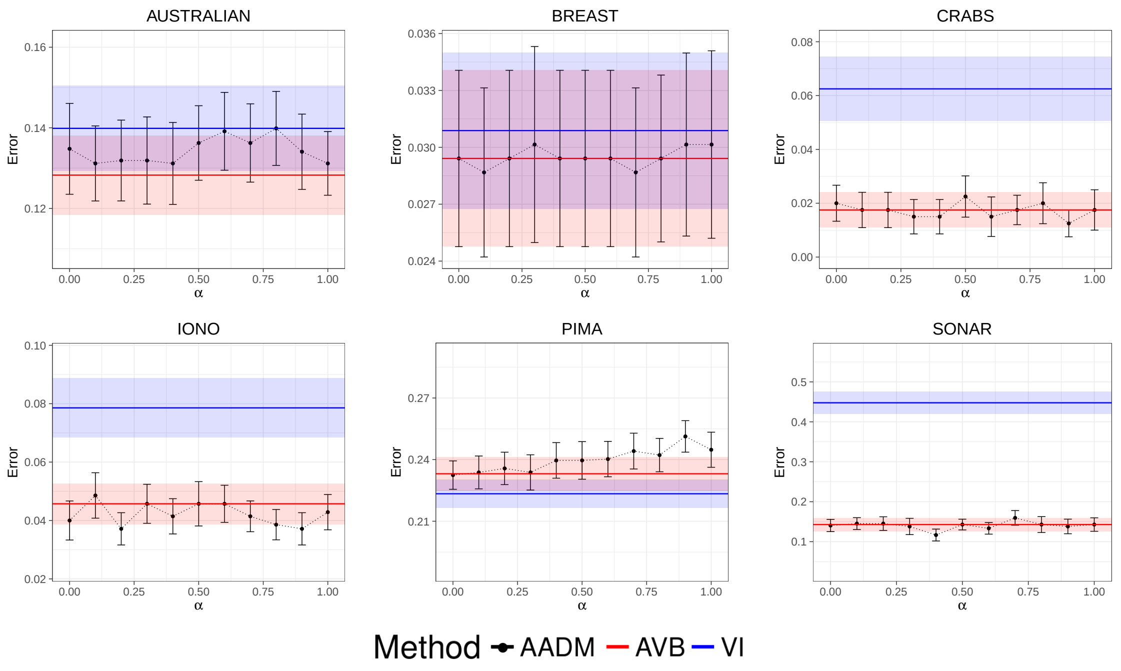


Figure A.2: Average results in terms of the classification error rate for the different UCI datasets for comparison of the methods. Black represents the performance for our method, AADM, for different values of α. Red is the performance of AVB. VI is presented in blue. Lower values are better. Best seen in color.

The results for the classification error are displayed in Figure A.2. In the case of this other performance metric we observe the same general trends as for the test log-likelihood (the lower the result the better). Overall, AADM and AVB perform better than VI with the exception of *Pima*, on which VI is able to yield better results although not by a big difference. VI performs worse across datasets, but yet again it is difficult to conclude whether AVB or AADM perform better than the other since their results are very similar (again, in cases such as *Australian* or *Breast*, the width of the error-bars relative to the mean error is similar to the other experiments, although the scale of the y-axis make them appear a lot wider). Summing up, there is not one single clear better value for the α parameter. As expected, AADM and AVB have very similar results when $\alpha \to 0$ in AADM.

Finally we show the results obtained in terms of the Brier score for all the datasets in Figure A.3. These plots follow the same legend as for the previous plots (black for AADM, red for AVB, blue for VI). As in the case of the test log-likelihood and the prediction error, in general, there are no substantially different results for different values of alpha, although in some cases it is clear that both AADM and AVB outperform VI (the lower the values, the better).

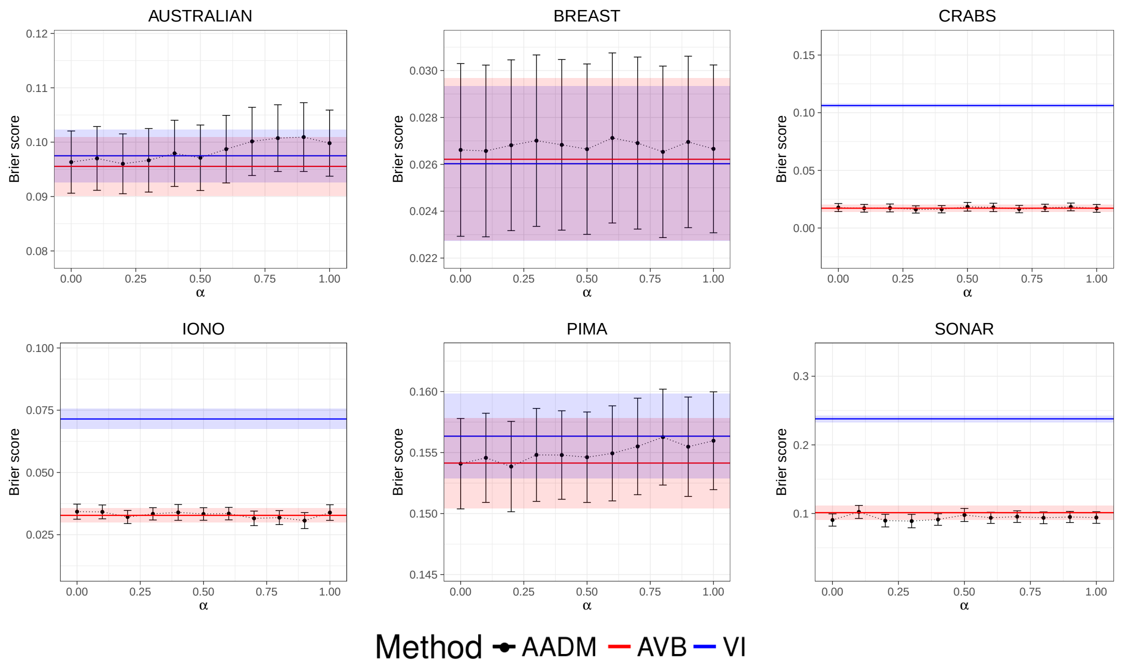


Figure A.3: Average results in terms of the Brier score rate for the different UCI datasets for comparison of the methods. Black represents the performance for our method, AADM, for different values of α. Red is the performance of AVB. VI is presented in blue. Lower values are better. Best seen in color.

A.2.2 Average Rank Results on Binary Classification Tasks

We have carried out the same average rank analysis that was done in the main manuscript for regression. More precisely, we rank the performance of AADM for each value of α from best to worse (*e.g.* rank 1 represents the best result) and compare average ranks for each value of α, for each metric. The results for all metrics are displayed in Figure A.4.

The average rank results obtained here are not as clear as the ones of regression. There is no trend suggesting that a particular value of α may be better than the others. In the case of the classification error there is only a slight improvement for α values lower than 0.5. The same could be said for the Brier score, and for the test log-likelihood. However, especially for this last metric, the error bars prevent drawing any solid conclusion. In spite of this, the performance achieved across all the α values tested here remains comparable to that of AVB. Therefore, in binary classification tasks AADM gives results that are similar to the ones of other methods from the literature.

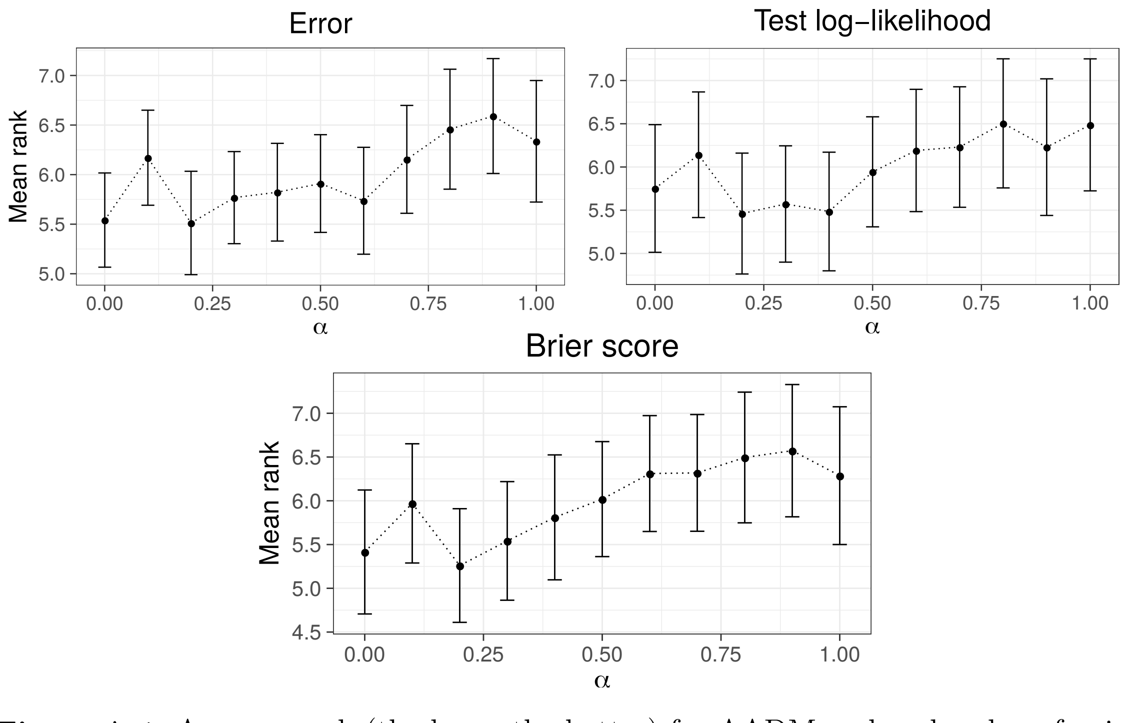


Figure A.4: Average rank (the lower the better) for AADM and each value of α in terms of the classification error rate (top-left), test log-likelihood (top-right) and Brier score (bottom) across all the binary classification datasets and splits.

A.2.3 Multi-class Classification

We have evaluated AADM on the MNIST (LeCun et al. 2010) and CIFAR-10 datasets (Krizhevsky, Hinton, et al. 2009). We have used the same architecture for the neural network as in the previous problems. Namely, a *fully-connected* (FC) network. The likelihood factors in this case correspond to a soft-max activation function. In MNIST we use $60,000$ data instances for training and $10,000$ for testing. In CIFAR-10 we use $50,000$ data instances for training and $10,000$ for testing.

In these experiments we have considered bigger networks, as described below. To speed-up the training process we have also considered an alternative implicit model for the posterior approximation, as described in (Louizos and Welling 2017a). In particular, we assume that the weights of each layer are obtained from:

$$w_{ij} \sim \mathcal{N}(z_j \mu_{ij}, \sigma_{ij}^2)\,, \qquad\qquad z_j \sim q(z_j)\,, \tag{A.2.1}$$

where w_{ij} (for $i = 1$, to the total number of hidden units, and for $j = 1$, to the total number of input dimensions to the layer) is the network weight, $z_j \mu_{ij}$ is the mean of a Gaussian distribution and σ_{ij}^2 its variance. Importantly, z_j is a random variable generated from an implicit distribution similar to the one we consider for binary and regression problems. The actual distribution of the weights is hence an infinity mixture of Gaussians, obtained by marginalizing z_j. Although potentially less flexible, the main advantage of using this implicit model for the posterior approximation is that it allows using the local reparametrization trick, which is key in speeding-up the training process in bigger networks (Kingma et al. 2015).

In both MNIST and CIFAR-10, the generator, discriminator and the main network have 2 hidden layers with 200 units each. We employed 1000 epochs for training. Training is done drawing 10 samples for the weights, while 750 samples are used for testing (the batch size for training is 200). In these experiments we have disabled adaptive contrast since we observed slightly better results when it was not used. The code used in these experiments is available at `https://github.com/simonrsantana/AADM`.

A.2.3.1 MNIST

The results obtained on the MNIST dataset are displayed in Figure A.5 for each performance metric considered. We observe that there are not big differences among the values obtained for AADM, AVB and VI. However, AADM seems to give slightly better results than AVB and VI in terms of the test log-likelihood and the Brier score. In both cases there is a preference for higher values of α. This behavior can be explained as a consequence of the changes that the objective function suffers when increasing α. As we mentioned in the main part of the article, higher values of α focus more on the test log-likelihood. Moreover, the Brier score is closely related to the test log-likelihood since it is concerned with the goodness-of-fit of the predictive distribution. Therefore, it is to be expected a correlation among the two metrics.

We would like to point out that the results obtained here on MNIST are similar to those in the literature for the type of networks used in these experiments (fully

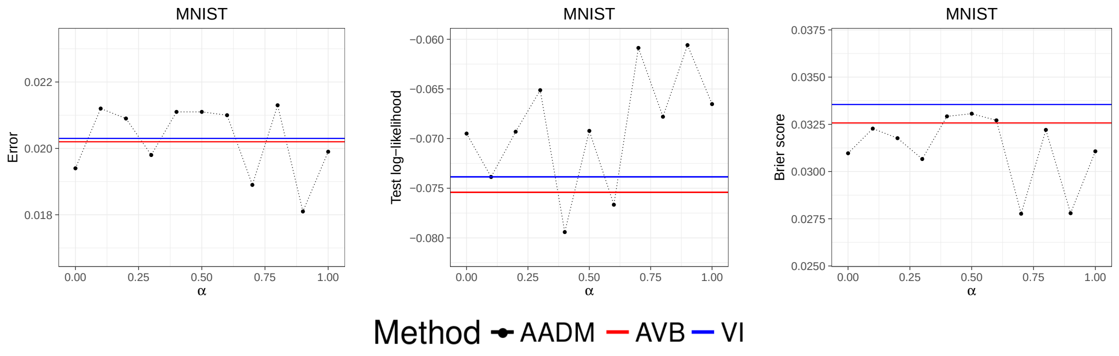


Figure A.5: Results in the MNIST dataset comparing the three different methods. Black represents the performance for our method, AADM, for different values of α. Red is the performance of AVB. VI is presented in blue. Here are represented the classification error rate (left - the lower the better), the test log-likelihood (center - the higher the better) and the Brier score (right - the lower the better). Best seen in color.

connected networks using a Gaussian prior). See (Blundell et al. 2015b) for further details. It is expected that better results could be obtained by fine tuning the network architecture, using specific priors, or other networks based on convolutional layers. The evaluation of AADM on those models is left for future work. The results obtained by AADM are also similar to or better than those obtained by other models such as multi-class Gaussian process classifiers (Villacampa-Calvo and Hernández-Lobato 2017).

A.2.3.2 CIFAR-10

The results on the CIFAR-10 dataset are displayed in Figure A.6. In this case, the experiment results are less clear than those of MNIST. AADM and AVB seem to outperform VI in most cases, but we cannot extract definitive conclusions about what value of alpha is better. We believe this is due to the fact that CIFAR-10 is a complicated dataset with a lot of noise, and the results reflect these difficulties.

As in MNIST, the results obtained in CIFAR-10 are similar to those reported in the literature for fully connected neural networks (Lin et al. 2015). Classification error rates of $\sim 50\%$ are common when these neural networks are used, particularly if they are not optimized specifically for the task at hand. Nevertheless, in (Lin et al. 2015) it is shown that using specific architectures and more units per hidden layer (as well as more hidden layers) one could improve the accuracy of the method. However, in our case, since we are only interested in comparing the results of each method for approximate inference, we decided to use the same architecture as in MNIST. Further improvement in the CIFAR-10 dataset is expected by making use of convolutional layers. Nevertheless, evaluating AADM in these networks is left as future work. The results obtained by AADM are also similar to or better than

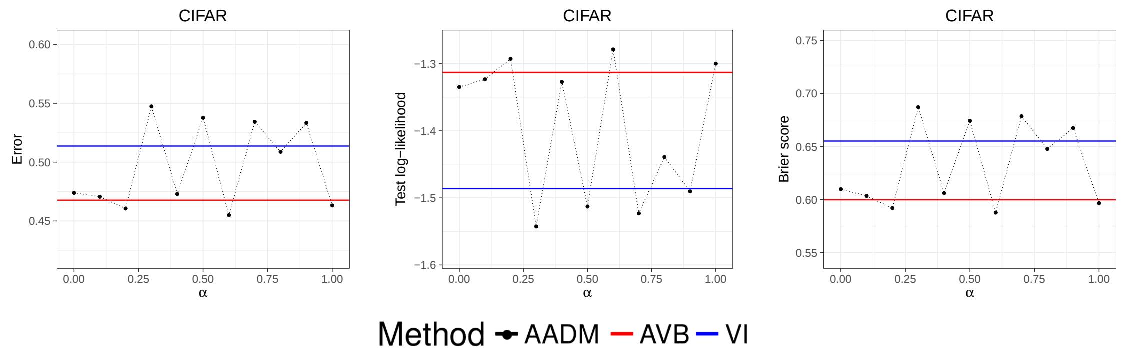


Figure A.6: Results in the CIFAR-10 dataset comparing the three different methods. Black represents the performance for our method, AADM, for different values of α. Red is the performance of AVB. VI is presented in blue. Here are represented the classification error rate (left - the lower the better), the test log-likelihood (center - the higher the better) and the Brier score (right - the lower the better). Best seen in color.

those obtained by other models such as multi-class Gaussian process classifiers using convolutional kernels (Wilk et al. 2017)

Appendix **B**

Appendix for Chapter 5

B.1 Results with the Single-KL Objective Function

SIP can be trained using this original formulation of the objective function with a single KL contribution. Although the resulting predictive distribution produces good results, the negligible contributions to the gradients made by the prior parameters makes the system unable to train its prior model. The results from this process, using the SIP$_{\mathrm{BNN}}$ model, are shown in Figure B.1. There, the samples from the prior model can be seen to not follow the behavior of the original data, while the predictive distribution is capable of reproducing the original bimodal behavior. This means that the prior is not being properly trained, although the posterior model is flexible enough to compensate for this fact and accurately follow the data behavior.

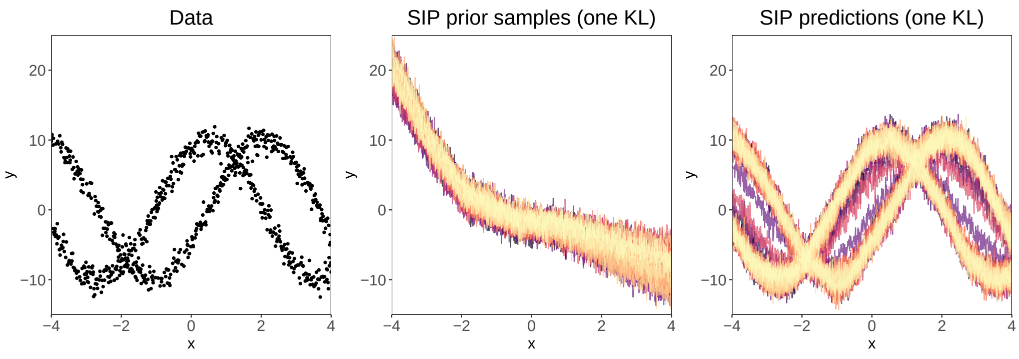


Figure B.1: Results of training SIP with the original f-ELBO formulation with only one KL contribution for the bimodal data presented in the main text (*left*). The predictive distribution (*right*) reproduces the bimodal behavior correctly. However, the functions sampled from the final prior model (*center*) do not follow the same behavior of the data.

From these results we conclude that the contribution to the gradient of the original objective function from the prior parameters is much less relevant than the one coming from the data term. Therefore, we need to introduce the new $\mathcal{L}^\star(q)$ objective, which allows us to train simultaneously both the prior and posterior parameters.

B.2 Adaptive Contrast for SIP

Adaptive contrast (AC) is introduced by (Mescheder et al. 2017) as an attempt to improve the accuracy of the estimation of the log ratio between distributions in the KL. This estimation will follow from the result of the auxiliary discriminator problem, whose optima is $T_{\omega^\star}$. However, since the distributions being compared are usually very different, the discriminator has no problem telling apart samples from each of them, which does not encourage for an optimal fit of its parameters. Therefore, to make this task harder, an extra Gaussian distribution is introduced in (Mescheder et al. 2017). In our case, since the KL distribution is estimated between two IPs, we need to reformulate the original AC to accommodate for this fact. This will imply extending the original AC to use two discriminators instead of one. Let us define two Gaussian distributions $\bar{q}$ and $\bar{p}$ with the same moments as the samples from both q and p (our IP approximating posterior and IP prior)

$$\bar{q} \sim \mathcal{N}(\mu(q), \sigma^2(q)), \qquad \bar{p} \sim \mathcal{N}(\mu(p), \sigma^2(p)) \tag{B.2.1}$$

with $\mu(\cdot)$ and $\sigma^2(\cdot)$ as the mean and variance across samples from q or p. Using this, we rewrite the two-KL part of the f-ELBO objective as the following:

$$\begin{aligned}
\mathrm{KL}(q|p) + \mathrm{KL}(p|q) &= \mathbb{E}_q\left[\log \frac{q\bar{q}\bar{p}}{p\bar{q}\bar{p}}\right] + \mathbb{E}_p\left[\log \frac{p\bar{p}\bar{q}}{q\bar{p}\bar{q}}\right] \\
&= \mathbb{E}_q\left[\log \frac{q}{\bar{q}}\right] + \mathbb{E}_q\left[\log \frac{\bar{p}}{p}\right] + \mathbb{E}_q\left[\log \frac{\bar{q}}{\bar{p}}\right] \\
&\quad + \mathbb{E}_p\left[\log \frac{p}{\bar{p}}\right] + \mathbb{E}_p\left[\log \frac{\bar{q}}{q}\right] + \mathbb{E}_p\left[\log \frac{\bar{p}}{\bar{q}}\right] \\
&= \mathbb{E}_q[T(q,\bar{q}) - T(p,\bar{p})] + \mathbb{E}_p[T(p,\bar{p}) - T(q,\bar{q})] \\
&\quad + \mathbb{E}_q\left[\log \frac{\bar{q}}{\bar{p}}\right] + \mathbb{E}_p\left[\log \frac{\bar{p}}{\bar{q}}\right].
\end{aligned} \tag{B.2.2}$$

Now we will have to employ *two* discriminators, one for samples from q and $\bar{q}$ ($T(q,\bar{q})$), and for samples from p and $\bar{p}$ ($T(p,\bar{p})$). The two last contributions to this expression are given by the log-ratio of two Gaussian distributions with given mean and variances, and are thus tractable and have a closed-form solution.

In practice, both discriminators needed here are estimated once with the expected value w.r.t. samples from q and from p. To help with this task, one could standardize the samples from q and p in beforehand, following the description in (Mescheder et al. 2017; Rodríguez Santana and Hernández-Lobato 2020)

$$\mathrm{KL}(q|p) + \mathrm{KL}(p|q) = \mathrm{KL}(q_0|p_q) + \mathrm{KL}(p_0|q_p), \tag{B.2.3}$$

where

$$q_0 = \frac{q - \mu(q)}{\sigma(q)}; \qquad p_0 = \frac{p - \mu(p)}{\sigma(p)}, \qquad q_p = \frac{q - \mu(p)}{\sigma(p)}; \qquad p_q = \frac{p - \mu(q)}{\sigma(q)}. \quad \text{(B.2.4)}$$

This simplifies the involved calculations for the discriminators involved, since now several terms employ standardized Gaussian distributions.

We opt to include the description of AC here for the sake of completeness, although in the experiments conducted in the main text, we found AC produced overfitting in the training of the prior. This would then require regularization techniques such as dropout or the *llh regularization* term used in VIP (Ma et al. 2019). We have tested these models as well, and thus far we have seen they also provide good performance. However, we think the added difficulties of training an extra discriminator and the need for a regularization term surpasses the benefits for this extra additions to the original SIP model.

B.3 Extra Synthetic Data Comparisons

B.3.1 Alternative Setups for Bimodal Data

The experiments conducted with synthetic data in the main text were all performed using the same prior model, a BNN with 2 layers with 50 units per layer. However, to produce those comparisons we have made changes on both methods that could affect their final results: the original code for VIP has a regularization term that was removed for the comparisons there, helping both the prior and the predictive distribution to represent heteroscedastic behavior (Ma et al. 2019). On the other hand, fBNN can employ a GP prior (or a sparse GP), which would have to be trained in beforehand (Sun et al. 2019). For the sake of completeness, we have run these tests as well with the same bimodal data being used in the main text (Depeweg et al. 2017; Rodríguez Santana and Hernández-Lobato 2020).

In Figure B.2 we can see the comparisons between methods using these alternative setups, namely VIP with the regularization term (denoted as VIP*) and fBNN with the GP prior (fBNN$_{\text{GP}}$). As in the figure in the main text, the first column represents both the original bimodal data (first row, black) and the predictive distribution obtained from applying HMC sharing the trained prior parameters of SIP (second row, blue). The other figures represent both the prior samples (first row) or the predictive distribution (second row) for VIP*, fBNN$_{\text{GP}}$ and SIP$_{\text{BNN}}$ respectively. In the case of fBNN$_{\text{GP}}$ we see that the prior is now being trained (although that needs to happen separately from the rest of the model), resulting in samples that fit better the behavior of the data here. The predictive distribution follows the mean of the data as it did in the case of fBNN$_{\text{BNN}}$, and although the noise is slightly smaller in this case, it is not able to reproduce the original bimodality. However, an important detail when comparing these figures against the ones present in the main text is that the y axis range has been increased here w.r.t. the one used in the original one, which may make these figures seem more concentrated towards the mean of the data.

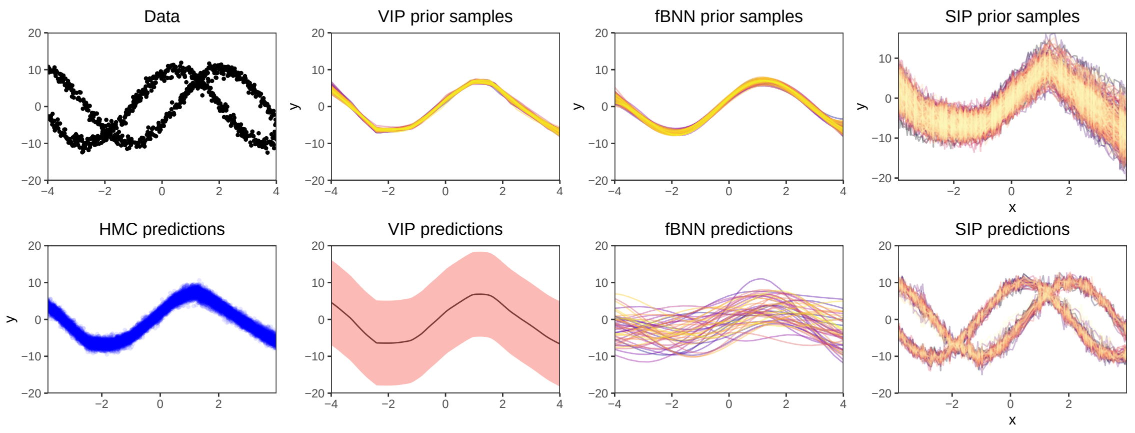


Figure B.2: Comparison between methods using bimodal data and alternative setups for VIP and fBNN: VIP includes the regularization term as in the other experiments, and fBNN uses the GP prior. The first column represents the original data (first row, in black) and the HMC predictive distribution (second row, in blue). The remaining figures represent the prior samples (first row) and the predictive distribution (second row) for each method, namely VIP* (with regularizer), fBNN$_{\text{GP}}$ and SIP$_{\text{BNN}}$. For the predictive distribution of VIP* the black line represents the predictive mean and the shaded area represents the $\pm 2\sigma$ region. For comparison w.r.t. the original image in the main text, the y-axis range has been increased here so that every plot could be easily seen. Best seen in color.

On the other hand, for VIP* we can clearly see that both figures are now not showing any of the heteroscedastic behavior they had in the main text. In this case, the prior, as well as the predictive distribution, fit closely the mean of the data, and the regions of $\pm 2\sigma$ in the latter have a fixed width, unlike in the original figure. Thus, removing the regularizer has helped the method to fit a more expressive prior and to obtain better predictive distribution samples, since introducing it hinders the expressiveness of the model overall. However, since this is the original setup the authors in (Ma et al. 2019) propose, this is the one we employed in both the UCI multivariate regression problems as well as in the convergence experiments. Finally, the good performance of the method regarding the RMSE metric can be explained by its focus on adjusting the predictions to the mean of the training data.

B.3.2 Heteroscedastic Data

To test the ability of SIP to reproduce other complex features present in the training data, we have employed as well an heteroscedastic dataset (Depeweg et al. 2017; Rodríguez Santana and Hernández-Lobato 2020). We constructed this dataset by uniformly sampling 1000 values for x between $[-4, 4]$ and then obtaining y for each sample as

$$y = 7\sin(x) + \epsilon\sin(x) + 10, \quad \epsilon \sim \mathcal{N}(0, \sigma = 2). \tag{B.3.1}$$

To perform these experiments we employ the same setup as the one in the main text for the bimodal problem.

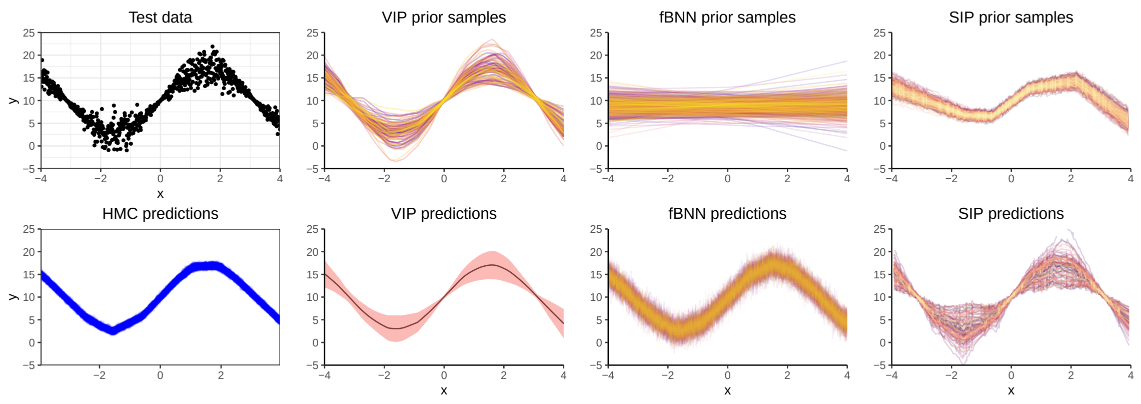


Figure B.3: Comparison between methods using heteroscedastic data. The first column represents the data (first row, in black) and the HMC predictive distribution (second row, in blue). The remaining figures represent the prior samples (first row) and the predictive distribution (second row) for each method, namely VIP, fBNN$_{\mathrm{BNN}}$ and SIP$_{\mathrm{BNN}}$. For the predictive distribution of VIP the line represents the predictive mean and the shaded area represents the $\pm 2\sigma$ region. Best seen in color.

In Figure B.3 we present the results of training the same models we used in the main text for the bimodal dataset (HMC, VIP, fBNN$_{\mathrm{BNN}}$ and SIP$_{\mathrm{BNN}}$) but with the new heteroscedastic synthetic data (represented in the first column, second row, in black). Here, as was the case earlier, HMC (first column, second row, in blue) is unable to reproduce the heteroscedastic behavior present in the data (first column, first row, in black). Even though it uses the trained prior from SIP, since the selected NN model is not originally capable of reproducing this behavior, HMC is unable to reproduce it either: the prior does not behave in an heteroscedastic manner, and neither does the likelihood, resulting in a non-heteroscedastic predictive distribution. For the rest of the methods we have the prior samples (first row) and the predictive distribution samples (second row) to compare against them. In the case of VIP (second column), opposite to what is done in Section B.3.1, the regularization term here is turned off again, thus obtaining heteroscedastic behavior both in the prior and predictive distributions. If the regularization term is turned on, however, the heteroscedasticity here is lost, as happened in Section B.3.1. For fBNN (third column) we see that the BNN prior is not being trained. Its predictive distribution follows the original data mean closely although without any heteroscedasticity. Finally, in SIP (last column) we can see that the prior behaves somewhat similarly to the original data distribution mean, as it did in the bimodal case. In this case, SIP's predictive distribution also closely follows that of the original data, with heteroscedastic behavior as well. In this case, if VIP uses the regularization term, SIP is the only method able to reproduce this behavior, resulting in a better final predictive distribution than the one provided by HMC. This implies what is mentioned in the main text: if the wrong model is chosen, HMC may be unable to reproduce important features from the data, ones that SIP captures in its predictive distribution. This makes SIP much less susceptible to implicit bias errors caused by choosing the incorrect model for a given problem.

B.3.3 Inducing Points Evolution

To complement the analysis of the evolution of the location of the inducing points in the main text, we conducted an extra experiment using an alternative dataset. This dataset is generated by sampling uniformly 1000 values for x between $[-4, 5]$, and then obtaining y from the following definition:

$$y = \begin{cases} 10 + \epsilon, & x < 0 \\ 10(1 + \sin(x)) + \epsilon, & x \geq 0, \end{cases} \tag{B.3.2}$$

with $\epsilon \sim \mathcal{N}(0, \sigma = 2)$. The piece-wise definition of y is constructed so that y is continuous but with a non-defined derivative in $x = 0$. We train SIP with this dataset until it converges, and register the location of the inducing points for each epoch of training (2000 here). We employ 50 inducing points here for visualization purposes.

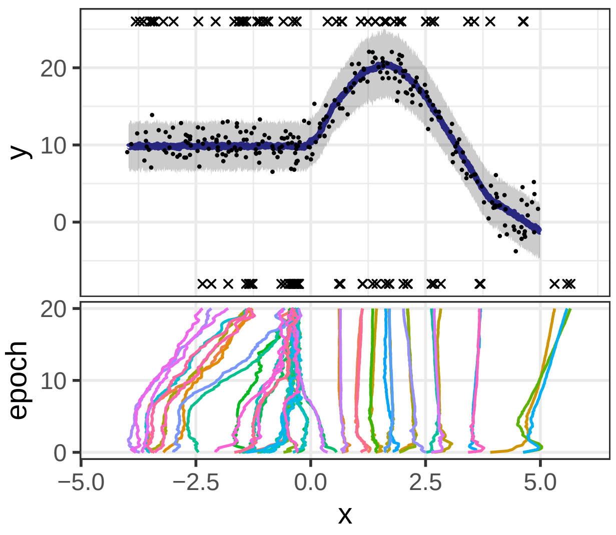


Figure B.4: Results of the SIP method applied to the piece-wise defined synthetic dataset: (*top plot*) samples from the predictive distribution of the method (mean as the blue line, training points as black dots, with crosses representing initial and final positions for the inducing points (top and bottom crosses, respectively); (*bottom plot*) locations of the inducing points for every training epoch, which are scaled by 100. Best seen in color.

In Figure B.4 we see the results of training SIP with this alternative dataset, using the same representation we employed in the experiment included in the main text: the top figure contains the original data (black dots), the predictive distribution samples (with the mean as a blue line), and the initial and final location of the inducing points represented as crosses both in the top and bottom of the figure respectively. The bottom plot represents the location of each inducing point for each epoch in the y-axis (scaled by a factor of 100, ranging from 0 to 2000). Moreover, in this case, we have set the starting positions of the 50 inducing points in random locations throughout the whole range of the training set. In the figure, we see that the predictive distribution follows closely that of the original data. The inducing points that started in the region where the data is constant ($x < 0$) seem to have a tendency to concentrate around the region of $x = 0$, moving to the right to the matching point between the two pieces of the function of y. Due to the simple nature of the data before this point, the model focuses the inducing points originally positioned in this region and places them closer to $x = 0$, helping it when modeling the change in behavior between the two parts of y. On the other side, for the $x > 0$ region, the inducing points are distributed more homogeneously to model the sine behavior. Furthermore, the right-most points seem to stray away from the region of the dataset, which is caused by employing far more inducing points than needed in such a simple dataset. This has been done for illustration purposes only, which also causes the overlapping of points across the dataset when there are more than what are needed in the same region. From all of this, we conclude that SIP seems able to distinguish between simpler and more complex regions of the data, and employs its resources effectively to model both at the same time.

B.4 Computational Details for the Convergence Experiments

The convergence experiments have been conducted employing the same computational resources for each method: Each of them employed on its own 2 CPU Intel(R) Xeon(R) Gold 5218 CPU at 2.30GHz (16 cores, 22 Mb L3 cache), [32 cores in total], with 192 GB RAM at 2,4 GHz. In the experiments, only the training time is reported. The same setup is used for both experiments (*Airplanes delay* and *Year Prediction MSD* datasets).

B.5 Regression Results for UCI Datasets

We include here the mean results for the UCI datasets for each method and metric. Each method is trained on 20 different random train/test splits of the dataset, and the values reported here represent the mean performance of each method in each dataset for the given metric in each case. For each dataset, the winning method is highlighted in bold while the one in second place is marked in red (in LL, the higher the better; in RMSE and CRPS, the lower the better). This is done as well for the rankings.

Table B.1: LL results in UCI datasets

Dataset	BBB	AVB	AADM	fBNN$_{\text{BNN}}$	VIP	SIP$_{\text{BNN}}$	SIP$_{\text{NS}}$
boston	-2.816±0.183	-2.409±0.153	**−2.348±0.151**	-3.193±0.687	-2.559±0.409	-2.51±0.321	-2.595±0.521
ccpp	-2.844±0.0449	-2.819±0.0499	-2.816±0.0495	-2.789±0.0805	-4.018±0.965	**−2.761±0.0507**	-2.775±0.0494
concrete	-3.331±0.0567	-2.976±0.0928	**−2.896±0.102**	-3.514±0.491	-3.208±1.04	-3.364±0.424	-2.934±0.165
ee	-2.155±0.0733	-1.678±0.0825	-1.239±0.101	-1.284±0.0485	**−0.995±0.146**	-1.002±0.292	-1.106±0.0888
wine	-0.979±0.0481	-0.955±0.0479	**−0.949±0.0476**	-19.77±4.32	-0.992±0.0691	-0.972±0.0683	-1.149±0.168
yatch	-2.588±0.0968	-1.742±0.085	-1.559±0.157	-2.114±0.0299	**−0.041±0.511**	-0.207±0.427	-0.559±0.172
kin8nm	1.114±0.0204	1.283±0.0354	**1.295±0.0312**	0.999±0.0687	0.979±0.0513	1.178±0.0264	1.223±0.023
naval	6.544±0.0724	6.097±0.628	6.483±0.366	6.157±0.235	5.944±1.8	**7.28±0.0598**	5.651±0.0445
Avg. ranking	5.39±0.05	3.91±0.06	2.82±0.06	5.61±0.07	3.70±0.12	**2.76±0.08**	3.81±0.07

Table B.2: RMSE results in UCI datasets

Dataset	BBB	AVB	AADM	fBNN$_{\text{BNN}}$	VIP	SIP$_{\text{BNN}}$	SIP$_{\text{NS}}$
boston	3.52 ± 0.804	2.53 ± 0.461	2.55 ± 0.481	5.66 ± 10.2	2.62 ± 0.697	2.88 ± 0.616	**2.29 ± 0.452**
ccpp	4.15 ± 0.201	4.05 ± 0.206	4.04 ± 0.207	**3.77 ± 0.205**	3.88 ± 0.224	3.92 ± 0.214	3.87 ± 0.198
concrete	6.64 ± 0.562	4.95 ± 0.569	4.92 ± 0.584	4.78 ± 0.694	4.53 ± 0.548	4.9 ± 0.825	**4.45 ± 0.56**
ee	2.02 ± 0.231	1.32 ± 0.133	1.77 ± 0.208	**0.639 ± 0.116**	0.789 ± 0.13	1.47 ± 0.316	0.836 ± 0.106
wine	0.649 ± 0.0388	0.633 ± 0.0342	0.631 ± 0.0339	0.78 ± 0.0645	**0.622 ± 0.0437**	0.636 ± 0.0367	0.625 ± 0.0452
yatch	2.3 ± 0.98	1.1 ± 0.42	1.3 ± 0.64	0.82 ± 0.31	0.58 ± 0.22	**0.36 ± 0.11**	0.54 ± 0.27
kin8nm	0.079 ± 0.0022	0.067 ± 0.0024	**0.067 ± 0.0023**	0.074 ± 0.0022	0.079 ± 0.0025	0.075 ± 0.0021	0.07 ± 0.0021
naval	(3.02 ± 4.73)e^{-5}	(4.96 ± 0.21) e^{-5}	(3.61 ± 0.17)e^{-5}	(3.78 ± 2.17)e^{-5}	(3.18 ± 1.06)e^{-5}	(**1.73 ± 0.15**) e^{-5}	(6.37 ± 0.98) e^{-5}
Avg. ranking	6.15±0.04	4.14±0.08	3.97±0.09	3.82±0.10	3.29±0.09	3.53±0.08	**3.10±0.09**

Table B.3: CRPS results in UCI datasets

Dataset	BBB	AVB	AADM	fBNN$_{\text{BNN}}$	VIP	SIP$_{\text{BNN}}$	SIP$_{\text{NS}}$
boston	1.859±0.363	1.475±0.281	1.456±0.298	1.861±1.137	**1.348±0.241**	1.573±0.278	1.45±0.259
ccpp	2.641±0.103	2.615±0.102	2.608±0.104	**2.013±0.08**	2.229±0.081	2.114±0.082	2.2±0.072
concrete	3.755±0.294	2.811±0.262	2.615±0.214	2.35±0.301	**2.273±0.196**	2.485±0.238	2.683±0.24
ee	1.121±0.139	0.703±0.07	0.694±0.082	0.41±0.03	**0.387±0.061**	0.532±0.078	0.591±0.078
wine	0.362±0.021	0.356±0.019	0.363±0.019	0.479±0.049	**0.341±0.02**	0.353±0.017	0.366±0.028
yatch	1.026±0.185	0.475±0.105	0.519±0.128	0.838±0.056	0.212±0.071	**0.154±0.03**	0.384±0.094
kin8nm	0.047±0.001	0.042±0.002	**0.041±0.001**	0.042±0.002	0.044±0.001	0.042±0.001	0.047±0.002
naval	(1.6±0.28) e^{-4}	(3.46 ±1.56) e^{-4}	(2.21± 1.22)e^{-4}	(2.56± 0.95) e^{-5}	(1.81± 0.68) e^{-5}	(**9.4 ± 0.70**) e^{-5}	(6.8± 0.28) e^{-5}
Avg. ranking	5.84±0.05	4.47±0.08	4.03±0.08	3.97±0.07	2.57±0.08	**2.45±0.08**	4.67±0.08

Bibliography

Abbasi-Asl, R. and Yu, B. (2017). “Structural compression of convolutional neural networks”. In: *arXiv preprint arXiv:1705.07356*.

Abramowitz, M. and Stegun, I. A. (1964). *Handbook of mathematical functions with formulas, graphs, and mathematical tables.* Vol. 55. US Government printing office.

Amari, S. (2012). *Differential-geometrical methods in statistics.* Vol. 28. Springer Science & Business Media.

Barber, D. and Bishop, C. M. (1998). “Ensemble learning in Bayesian neural networks”. In: *Nato ASI Series F Computer and Systems Sciences* 168, pp. 215–238.

Bauer, M., Wilk, M. van der, and Rasmussen, C. E. (2016). “Understanding probabilistic sparse Gaussian process approximations”. In: *Advances in Neural Information Processing Systems*, pp. 1533–1541.

Bayraktar, H. and Turalioglu, F. S. (2005). “A Kriging-based approach for locating a sampling site—in the assessment of air quality”. In: *Stochastic Environmental Research and Risk Assessment* 19.4, pp. 301–305.

Beal, M. J. (2003). “Variational algorithms for approximate Bayesian inference”. PhD thesis. University College London.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* New York, NY, USA: Springer Science+Business Media, LL¬C. ISBN: 0-387-31073-8, 978-0-387-31073-2.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015a). “Weight Uncertainty in Neural Network”. In: *International Conference on Machine Learning*, pp. 1613–1622.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015b). “Weight uncertainty in neural networks”. In: *International Conference on Machine Learning*, 1613–1622.

Bui, T. D., Yan, J., and Turner, R. E. (2017). “A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation”. In: *The Journal of Machine Learning Research* 18.1, pp. 3649–3720.

Bui, T., Hernandez-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016a). “Deep Gaussian Processes for Regression using Approximate Expectation Propagation”. In: *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1472–1481.

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016b). “Deep Gaussian processes for regression using approximate expectation propagation”. In: *International Conference on Machine Learning*, pp. 1472–1481.

Burt, D. R., Ober, S. W., Garriga-Alonso, A., and Wilk, M. van der (2020). “Understanding variational inference in function-space”. In: *3rd Symposium on Advances in Approximate Bayesian Inference.*

Challis, E., Hurley, P., Serra, L., Bozzali, M., Oliver, S., and Cercignani, M. (2015). “Gaussian process classification of Alzheimer’s disease and mild cognitive impairment from resting-state fMRI”. In: *NeuroImage* 112, pp. 232–243.

Chapelle, O., Scholkopf, B., and Zien, A. (2009). “Semi-supervised learning”. In: *IEEE Transactions on Neural Networks.*

Chen, N., Qian, Z., Nabney, I. T, and Meng, X. (2013). “Wind power forecasts using Gaussian processes and numerical weather prediction”. In: *IEEE Transactions on Power Systems* 29.2, pp. 656–665.

Chiles, J. P., Delfiner, P., et al. (1999). “Modeling spatial uncertainty”. In: *Geostatistics, Wiley series in probability and statistics.*

Cho, Y. (2012). *Kernel methods for deep learning.* University of California, San Diego.

Chu, W., Ghahramani, Z., Falciani, F., and Wild, D. L. (2005). “Biomarker discovery in microarray gene expression data with Gaussian processes”. In: *Bioinformatics* 21.16, pp. 3385–3393.

Coker, B., Pan, W., and Doshi-Velez, F. (2021). “Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data”. In: *arXiv preprint arXiv:2106.07052.*

Cortes, C. and Vapnik, V. (1995). “Support-vector networks”. In: *Machine learning* 20.3, pp. 273–297.

Cressie, N. (1990). “The origins of kriging”. In: *Mathematical geology* 22.3, pp. 239–252.

Cui, J. and Krems, R. V. (2016). “Efficient non-parametric fitting of potential energy surfaces for polyatomic molecules with Gaussian processes”. In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 49.22, p. 224001.

Damianou, A. and Lawrence, N. D. (2013). “Deep Gaussian processes”. In: *Artificial intelligence and statistics*, pp. 207–215.

Daniely, A., Frostig, R., and Singer, Y. (2016). “Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity”. In: *Advances In Neural Information Processing Systems* 29, pp. 2253–2261.

Datta, A., Banerjee, S., Finley, A. O., and Gelfand, A. E. (2016). “Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets”. In: *Journal of the American Statistical Association* 111.514, pp. 800–812.

Delhomme, J. P. (1978). “Kriging in the hydrosciences”. In: *Advances in water resources* 1.5, pp. 251–266.

Denker, J. S., Gardner, W., Graf, H. P., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., Baird, H. S., and Guyon, I. (1989). “Neural network recognizer for hand-written zip code digits”. In: *Advances in Neural Information Processing Systems*, pp. 323–331.

Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., and Hopfield, J. (1987). “Large automatic learning, rule extraction, and generalization”. In: *Complex systems* 1.5, pp. 877–922.

Depeweg, S., Hernández-Lobato, J. M., Doshi-Velez, F., and Udluft, S. (2017). “Learning and policy search in stochastic dynamical systems with Bayesian neural networks”. In: *International Conference on Learning Representations*.

Der Kiureghian, A. and Ditlevsen, O. (2009). “Aleatory or epistemic? Does it matter?” In: *Structural safety* 31.2, pp. 105–112.

Díaz-Vico, D., Torres-Barrán, A., Omari, A., and Dorronsoro, J. R. (2017). “Deep neural networks for wind and solar energy prediction”. In: *Neural Processing Letters* 46.3, pp. 829–844.

Dua, D. and Graff, C. (2017). *UCI Machine Learning Repository*. URL: `http://archive.ics.uci.edu/ml`.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). “Hybrid Monte Carlo”. In: *Physics letters B* 195, pp. 216–222.

Elhoseiny, M., Huang, S., and Elgammal, A. (2015). “Weather classification with deep convolutional neural networks”. In: *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 3349–3353.

Fang, D., Zhang, X., Yu, Q., Jin, T. C., and Tian, L. (2018). “A novel method for carbon dioxide emission forecasting based on improved Gaussian processes regression”. In: *Journal of cleaner production* 173, pp. 143–150.

Foong, A. Y. K., Burt, D. R., Li, Y., and Turner, R. E. (2020). “On the expressiveness of approximate inference in Bayesian neural networks”. In: *34th Conference on Neural Information Processing Systems.*

Fox, R. F. (1978). “Gaussian stochastic processes in physics”. In: *Physics Reports* 48.3, pp. 179–283.

Frey, B. J, Patrascu, R., Jaakkola, T., and Moran, J. (2001). “Sequentially fitting "inclusive" trees for inference in noisy-OR networks”. In: *Advances in Neural Information Processing Systems*, pp. 493–499.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). *The elements of statistical learning.* Vol. 1. 10. Springer series in statistics New York.

Fujiyoshi, H., Hirakawa, T., and Yamashita, T. (2019). “Deep learning-based image recognition for autonomous driving”. In: *IATSS research* 43.4, pp. 244–252.

Fukushima, K. and Miyake, S. (1982). “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets.* Springer, pp. 267–285.

Gal, Y. (2016). “Uncertainty in deep learning”. PhD thesis. PhD thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2015). “Dropout as a Bayesian approximation: Insights and applications”. In: *Deep Learning Workshop, ICML.* Vol. 1, p. 2.

Gal, Y. and Ghahramani, Z. (2016). “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning”. In: *International Conference on Machine Learning*, pp. 1050–1059.

Gao, L., Guo, Z., Zhang, H., Xu, X., and Shen, H. T. (2017). “Video captioning with attention-based LSTM and semantic consistency”. In: *IEEE Transactions on Multimedia* 19.9, pp. 2045–2055.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian data analysis.* CRC press.

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., and Modrák, M. (2020). “Bayesian workflow”. In: *arXiv preprint arXiv:2011.01808.*

Glasserman, P. (2013). *Monte Carlo methods in financial engineering.* Vol. 53. Springer Science & Business Media.

Gneiting, T. and Raftery, A. E (2007). “Strictly proper scoring rules, prediction, and estimation”. In: *Journal of the American statistical Association* 102.477, pp. 359–378.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning.* Vol. 1. MIT press Cambridge.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Graves, A. (2011). “Practical variational inference for neural networks”. In: *Advances in Neural Information Processing Systems*, pp. 2348–2356.

Grimit, E. P., Gneiting, T., Berrocal, V. J, and Johnson, N. A. (2006). “The continuous ranked probability score for circular variables and its application to mesoscale forecast ensemble verification”. In: *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography* 132.621C, pp. 2925–2942.

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Scholkopf, B. (1998). “Support vector machines”. In: *IEEE Intelligent Systems and their applications* 13.4, pp. 18–28.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013a). “Gaussian Processes for Big Data”. In: *Uncertainty in Artificial Intelligence*, 282–290.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013b). “Gaussian processes for big data”. In: *Uncertainty in Artificial Intellegence*, pp. 282–290.

Hernández-Lobato, J. M. and Adams, R. P. (2015). “Probabilistic backpropagation for scalable learning of Bayesian neural networks”. In: *International Conference on Machine Learning*, pp. 1861–1869.

Hernández-Lobato, J. M., Li, Y., Rowland, M., Hernández-Lobato, D., Bui, T. D., and Turner, R. E. (2016). “Black-box α-divergence minimization”. In: *International Conference on Machine Learning*, pp. 1511–1520.

Heskes, T. and Zoeter, O. (2002). “Expectation propagation for approximate inference in dynamic Bayesian networks”. In: *Uncertainty in Artificial Intelligence*, pp. 216–223.

Heskes, T. and Zoeter, O. (2012). “Expectation Propogation for approximate inference in dynamic Bayesian networks”. In: *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*.

Hinton, G. E. (2007). “Learning multiple layers of representation”. In: *Trends in cognitive sciences* 11.10, pp. 428–434.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7, pp. 1527–1554.

Hinton, G. E., Sejnowski, T. J., et al. (1999). *Unsupervised learning: foundations of neural computation*. MIT press.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580.*

Hinton, G. E. and Van Camp, D. (1993). “Keeping the neural networks simple by minimizing the description length of the weights”. In: *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13.

Ho, J. and Ermon, S. (2016). “Generative adversarial imitation learning”. In: *Advances in Neural Information Processing Systems* 29, pp. 4565–4573.

Hochreiter, S. and Schmidhuber, J. (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

Hoerl, A. E. and Kennard, R. W. (1970). “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1, pp. 55–67.

Hoffman, M. D. (2017). “Learning deep latent Gaussian models with Markov chain Monte Carlo”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1510–1519.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). “Stochastic variational inference.” In: *Journal of Machine Learning Research* 14.5.

Hornik, K., Stinchcombe, M., and White, H. (1989). “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5, pp. 359–366.

Hosaka, T. (2019). “Bankruptcy prediction using imaged financial ratios and convolutional neural networks”. In: *Expert systems with applications* 117, pp. 287–299.

Hu, G., Yang, Y., Yi, D., Kittler, J., Christmas, W., Li, S. Z., and Hospedales, T. (2015). “When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition”. In: *Proceedings of the IEEE international conference on computer vision workshops*, pp. 142–150.

Huszár, F. (2017). *Variational inference using implicit distributions.* ArXiv preprint arXiv:1702.08235.

Immer, A., Korzepa, M., and Bauer, M. (2021). “Improving predictions of Bayesian neural nets via local linearization”. In: *International Conference on Artificial Intelligence and Statistics*, pp. 703–711.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134.

Jaakkola, T. (2001). “Tutorial on variational approximation methods”. In: *Advanced mean field methods: theory and practice*, p. 129.

Jolliffe, I. (2005). “Principal component analysis”. In: *Encyclopedia of statistics in behavioral science.*

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). “An introduction to variational methods for graphical models”. In: *Machine learning* 37, pp. 183–233.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). “Exploring the limits of language modeling”. In: *arXiv preprint arXiv:1602.02410.*

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). “A convolutional neural network for modelling sentences”. In: *arXiv preprint arXiv:1404.2188.*

Karras, T., Laine, S., and Aila, T. (2019). “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410.

Kendall, A. and Gal, Y. (2017). “What uncertainties do we need in Bayesian deep learning for computer vision?” In: *31st Conference on Neural Information Processing Systems.*

Kingma, D. P. and Ba, J. (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980.*

Kingma, D. P. and Ba, J. (2015). “ADAM: A method for stochastic optimization”. In: *International Conference on Learning Representations.*

Kingma, D. P., Salimans, T., and Welling, M. (2015). “Variational dropout and the local reparameterization trick”. In: *Advances in Neural Information Processing Systems*, pp. 2575–2583.

Kleiber, W., Katz, R. W, and Rajagopalan, B. (2012). “Daily spatiotemporal precipitation simulation using latent and transformed Gaussian processes”. In: *Water Resources Research* 48.1.

Knoblauch, J., Jewson, J., and Damoulas, T. (2019). “Generalized variational inference: Three arguments for deriving new posteriors”. In: *arXiv preprint arXiv:1904.02063.*

Kobyzev, I., Prince, S., and Brubaker, M. (2020). “Normalizing flows: An introduction and review of current methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Krizhevsky, A., Hinton, G., et al. (2009). “Learning multiple layers of features from tiny images”. In.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems* 25, pp. 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). “Deep learning”. In: *Nature* 521, pp. 436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4, pp. 541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Y., Cortes, C., and Burges, C. J. C. (2010). “MNIST handwritten digit database”. In: URL: http://yann.lecun.com/exdb/mnist/.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). “Deep neural networks as Gaussian processes”. In: *International Conference on Learning Representations.*

Li, H., Habes, M., Wolk, D. A., Fan, Y., Initiative, A. D. N., et al. (2019). “A deep learning model for early prediction of Alzheimer’s disease dementia based on hippocampal magnetic resonance imaging data”. In: *Alzheimer’s & Dementia* 15.8, pp. 1059–1070.

Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D. D., and Chen, M. (2014). “Medical image classification with convolutional neural network”. In: *2014 13th international conference on control automation robotics & vision (ICARCV)*, pp. 844–848.

Li, X. and Wu, X. (2015). “Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4520–4524.

Li, Y. and Gal, Y. (2017). “Dropout inference in Bayesian neural networks with alpha-divergences”. In: *International Conference on Machine Learning*, pp. 2052–2061.

Li, Y., Hernández-Lobato, J. M., and Turner, R. E. (2015). “Stochastic expectation propagation”. In: *Advances in Neural Information Processing Systems 28.*

Li, Y. and Liu, Q. (2016). “Wild variational approximations”. In: *NIPS workshop on advances in approximate Bayesian inference.*

Li, Y. and Turner, R. E. (2016). “Rényi divergence variational inference”. In: *Advances in Neural Information Processing Systems*, pp. 1073–1081.

Lin, Z., Memisevic, R., and Konda, K. (2015). “How far can we go without convolution: Improving fully-connected networks”. In: *arXiv preprint arXiv:1511.02580.*

Liu, M., Li, F., Yan, H., Wang, K., Ma, Y., Shen, L., Xu, M., Initiative, A. D. N., et al. (2020). “A multi-model deep convolutional neural network for automatic hippocampus segmentation and classification in Alzheimer's disease". In: *Neuroimage* 208, p. 116459.

Liu, Q. and Wang, D. (2016). "Stein variational gradient descent: A general purpose Bayesian inference algorithm". In: *Advances In Neural Information Processing Systems*, pp. 2378–2386.

Liu, Y., Racah, E., Correa, J., Khosrowshahi, A., Lavers, D., Kunkel, K., Wehner, M., Collins, W., et al. (2016). "Application of deep convolutional neural networks for detecting extreme weather in climate datasets". In: *arXiv preprint arXiv:1605.01156*.

Louizos, C., Shi, X., Schutte, K., and Welling, M. (2019). "The functional neural process". In: *33rd Conference on Neural Information Processing Systems*.

Louizos, C. and Welling, M. (2017a). "Multiplicative Normalizing Flows for Variational Bayesian Neural Networks". In: *International Conference on Machine Learning*, pp. 2218–2227.

Louizos, C. and Welling, M. (2017b). "Multiplicative normalizing flows for variational Bayesian neural networks". In: *International Conference on Machine Learning*, pp. 2218–2227.

Ma, C., Li, Y., and Hernández-Lobato, J. M. (2019). "Variational implicit processes". In: *International Conference on Machine Learning*, pp. 4222–4233.

MacKay, D. J. C. (1992). "A practical Bayesian framework for backpropagation networks". In: *Neural computation* 4.3, pp. 448–472.

MacKay, D. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Mescheder, L., Geiger, A., and Nowozin, S. (2018). "Which training methods for GANs do actually converge?" In: *International conference on machine learning*, pp. 3481–3490.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). "Adversarial Variational Bayes: Unifying variational autoencoders and generative adversarial networks". In: *International Conference on Machine Learning*, pp. 2391–2400.

Minka, T. P. (2001a). "A family of algorithms for approximate Bayesian inference". PhD thesis. Massachusetts Institute of Technology.

Minka, T. P. (2001b). "Expectation propagation for approximate Bayesian inference". In: *Uncertainty in Artificial Intelligence*, pp. 362–369.

Minka, T. P. (2004). *Power EP*. Tech. rep. Technical report, Microsoft Research, Cambridge.

Minka, T. (2005). *Divergence measures and message passing*. Tech. rep. Technical report, Microsoft Research.

Morabito, F. C., Campolo, M., Mammone, N., Versaci, M., Franceschetti, S., Tagliavini, F., Sofia, V., Fatuzzo, D., Gambardella, A., Labate, A., et al. (2017). “Deep learning representation from electroencephalography of early-stage Creutzfeldt-Jakob disease and features for differentiation from rapidly progressive dementia”. In: *International journal of neural systems* 27.02, p. 1650039.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective.* MIT press.

Naveiro, R., Redondo, A., Insua, D. R., and Ruggeri, F. (2019). “Adversarial classification: An adversarial risk analysis approach”. In: *International Journal of Approximate Reasoning* 113, pp. 133–148.

Neal, R. M. (1996). “Priors for infinite networks”. In: *Bayesian Learning for Neural Networks.* Springer, pp. 29–53.

Neal, R. M. (2011). “MCMC using Hamiltonian dynamics”. In: *Handbook of Markov chain Monte Carlo*, pp. 113–162.

Neal, R. M. (2012). *Bayesian learning for neural networks.* Vol. 118. Springer Science & Business Media.

Oh, K. S. and Jung, K. (2004). “GPU implementation of neural networks”. In: *Pattern Recognition* 37.6, pp. 1311–1314.

Ong, C. S., Mary, X., Canu, S., and Smola, A. J. (2004). “Learning with non-positive kernels”. In: *Proceedings of the twenty-first International Conference on Machine Learning*, p. 81.

Paananen, T., Piironen, J., Andersen, M. R., and Vehtari, A. (2019). “Variable selection for Gaussian processes via sensitivity analysis of the posterior predictive distribution”. In: *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1743–1752.

Paisley, J., Blei, D., and Jordan, M. (2012). “Variational Bayesian inference with stochastic search”. In: *Proceedings of the 29th International Conference on Machine Learning.*

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). “Deep face recognition”. In.

Pastorello, N., Forbes, D. A, Foster, C., Brodie, J. P., Usher, C., Romanowsky, A. J., Strader, J., and Arnold, J. A. (2014). “The SLUGGS survey: exploring the metallicity gradients of nearby early-type galaxies to large radii”. In: *Monthly Notices of the Royal Astronomical Society* 442.2, pp. 1003–1039.

Pearlmutter, B. A. (1990). *Dynamic recurrent neural networks.* Tech. rep.

Qing, X. and Niu, Y. (2018). “Hourly day-ahead solar irradiance prediction using weather forecasts by LSTM”. In: *Energy* 148, pp. 461–468.

Quinonero-Candela, J. and Rasmussen, C. E. (2005). “A unifying view of sparse approximate Gaussian process regression”. In: *Journal of Machine Learning Research* 6, pp. 1939–1959.

Rasmussen, C. E. (2003). “Gaussian processes in machine learning”. In: *Summer school on machine learning*, pp. 63–71.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press.

Rényi, A. (1961). “On measures of entropy and information”. In: *Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 547–561.

Rezende, D. and Mohamed, S. (2015). “Variational inference with normalizing flows”. In: *International conference on machine learning*, pp. 1530–1538.

Ritter, H., Kukla, M., Zhang, C., and Li, Y. (2021). “Sparse Uncertainty Representation in Deep Learning with Inducing Weights”. In: *arXiv preprint arXiv:2105.14594*.

Rodríguez Santana, S. and Hernández-Lobato, D. (2020). “Adversarial α-divergence minimization for Bayesian approximate inference”. In: *Neurocomputing.*

Rosenblatt, F. (1958). “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6, p. 386.

Rosenblatt, F. (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms.* Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY.

Ruck, D. W., Rogers, S. K, and Kabrisky, M. (1990). “Feature selection using a multilayer perceptron”. In: *Journal of Neural Network Computing* 2.2, pp. 40–48.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). “Learning representations by back-propagating errors”. In: *Nature* 323.6088, pp. 533–536.

Sak, H., Senior, A. W, and Beaufays, F. (2014). “Long short-term memory recurrent neural network architectures for large scale acoustic modeling”. In.

Salcedo-Sanz, S., Casanova-Mateo, C., Muñoz-Marí, J., and Camps-Valls, G. (2014). “Prediction of daily global solar irradiation using temporal Gaussian processes”. In: *IEEE Geoscience and Remote Sensing Letters* 11.11, pp. 1936–1940.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). “Improved techniques for training GANs”. In: *Advances in Neural Information Processing Systems* 29, pp. 2234–2242.

Salimans, T., Kingma, D., and Welling, M. (2015). “Markov chain Monte Carlo and variational inference: Bridging the gap”. In: *International Conference on Machine Learning*, pp. 1218–1226.

Salimbeni, H. and Deisenroth, M. (2017). “Doubly stochastic variational inference for deep Gaussian processes”. In: *31st Conference on Neural Information Processing Systems*.

Santosa, F. and Symes, W. W. (1986). “Linear inversion of band-limited reflection seismograms”. In: *SIAM Journal on Scientific and Statistical Computing* 7.4, pp. 1307–1330.

Sarraf, S. and Tofighi, G. (2016). “Classification of alzheimer’s disease using fmri data and deep learning convolutional neural networks”. In: *arXiv preprint arXiv:1603.08631*.

Scholkopf, B. (2001). “The kernel trick for distances”. In: *Advances in Neural Information Processing Systems*, pp. 301–307.

Senge, R., Bösner, S., Dembczyński, K., Haasenritter, J., Hirsch, O., Donner-Banzhoff, N., and Hüllermeier, E. (2014). “Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty”. In: *Information Sciences* 255, pp. 16–29.

Serna, I., Morales, A., Fierrez, J., Cebrian, M., Obradovich, N., and Rahwan, I. (2019). “Algorithmic discrimination: Formulation and exploration in deep learning-based face biometrics”. In: *arXiv preprint arXiv:1912.01842*.

Shah, A., Wilson, A., and Ghahramani, Z. (2014). “Student-t processes as alternatives to Gaussian processes”. In: *International Conference on Artificial Intelligence and Statistics*, pp. 877–885.

Sharma, R., Kamble, S. S., Gunasekaran, A., Kumar, V., and Kumar, A. (2020). “A systematic literature review on machine learning applications for sustainable agriculture supply chain performance”. In: *Computers & Operations Research* 119, p. 104926.

Snelson, E. and Ghahramani, Z. (2005). “Sparse Gaussian processes using pseudo-inputs”. In: *Advances in Neural Information Processing Systems* 18, pp. 1257–1264.

Snelson, E. and Ghahramani, Z. (2007). “Local and global sparse Gaussian process approximations”. In: *International Conference on Artificial Intelligence and Statistics*, pp. 524–531.

Snelson, E., Rasmussen, C. E., and Ghahramani, Z. (2004). “Warped Gaussian processes”. In: *Advances in Neural Information Processing Systems* 16, pp. 337–344.

Solomonoff, R. J. (1964). “A formal theory of inductive inference. Part II”. In: *Information and control* 7.2, pp. 224–254.

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). “Ladder variational autoencoders”. In: *Advances in Neural Information Processing Systems*, pp. 3738–3746.

Song, S., Huang, H., and Ruan, T. (2019). “Abstractive text summarization using LSTM-CNN based deep learning”. In: *Multimedia Tools and Applications* 78.1, pp. 857–875.

Soudry, D., Hubara, I., and Meir, R. (2014). “Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights”. In: *Advances in Neural Information Processing Systems*, pp. 963–971.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Steinwart, I. and Christmann, A. (2008). *Support vector machines*. Springer Science & Business Media.

Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). “Functional variational Bayesian neural networks”. In: *International Conference on Learning Representations*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). “Sequence to sequence learning with neural networks”. In: *Advances in Neural Information Processing Systems*, pp. 3104–3112.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tibshirani, R. (1996). “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1, pp. 267–288.

Tishby, N., Levin, E., and Solla, S. A. (1989). “Consistent inference of probabilities in layered networks: Predictions and generalization”. In: *International Joint Conference on Neural Networks*. Vol. 2, pp. 403–409.

Titsias, M. K. and Ruiz, F. (2019). “Unbiased implicit variational inference”. In: *Artificial Intelligence and Statistics*, pp. 167–176.

Titsias, M. (2009). “Variational learning of inducing variables in sparse Gaussian processes”. In: *Internacional Conference on Artificial Intelligence and Statistics*, pp. 567–574.

Tran, D., Ranganath, R., and Blei, D. (2017). “Hierarchical implicit models and likelihood-free variational inference”. In: *Advances in Neural Information Processing Systems*, pp. 5523–5533.

Van Erven, T. and Harremos, P. (2014). “Rényi divergence and Kullback-Leibler divergence”. In: *IEEE Transactions on Information Theory* 60.7, pp. 3797–3820.

Vehtari, A., Gelman, A., Sivula, T., Jylänki, P., Tran, D., Sahai, S., Blomstedt, P., Cunningham, J. P., Schiminovich, D., and Robert, C. P. (2020). “Expectation propagation as a way of life: A framework for Bayesian inference on partitioned data”. In: *Journal of Machine Learning Research* 21, pp. 17–1.

Villacampa-Calvo, C. and Hernández-Lobato, D. (2017). “Scalable Multi-Class Gaussian Process Classification using Expectation Propagation”. In: *International Conference on Machine Learning*, pp. 3550–3559.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). “Show and tell: A neural image caption generator”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164.

Wang, C., Zhang, W., and Villarini, G. (2021). “On the use of convolutional Gaussian processes to improve the seasonal forecasting of precipitation and temperature”. In: *Journal of Hydrology* 593, p. 125862.

Wenzel, F., Roth, K., Veeling, B., Swiatkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. (2020). “How Good is the Bayes Posterior in Deep Neural Networks Really?” In: *International Conference on Machine Learning*, pp. 10248–10259.

Werbos, P. (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University.

Wilk, M. Van der, Rasmussen, C. E., and Hensman, J. (2017). “Convolutional Gaussian processes”. In: *Advances in Neural Information Processing Systems*, pp. 2849–2858.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA.

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., and Wang, C. (2018). “Machine learning and deep learning methods for cybersecurity”. In: *IEEE access* 6, pp. 35365–35381.

Yadav, S. S. and Jadhav, S. M. (2019). “Deep convolutional neural network based medical image classification for disease diagnosis”. In: *Journal of Big Data* 6.1, pp. 1–18.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). “Convolutional neural networks: an overview and application in radiology”. In: *Insights into imaging* 9.4, pp. 611–629.

Yu, H., Chen, Y., Dai, Z., Low, K. H., and Jaillet, P. (2019). “Implicit posterior variational inference for deep Gaussian processes”. In: *33rd Conference on Neural Information Processing Systems*.

Zhang, L. W., Zhu, P., and Liew, K. M. (2014). “Thermal buckling of functionally graded plates using a local Kriging meshless method”. In: *Composite Structures* 108, pp. 472–492.

Zimmerman, D., De Marsily, G., Gotway, C. A., Marietta, M. G., Axness, C. L., Beauheim, R. L., Bras, R. L., Carrera, J., Dagan, G., Davies, P. B., et al. (1998). “A comparison of seven geostatistically based inverse approaches to estimate transmissivities for modeling advective transport by groundwater flow”. In: *Water Resources Research* 34.6, pp. 1373–1413.