Modeling Uncertainty for Reliable Probabilistic Modeling in Deep Learning and Beyond

November, 2021

Autor: Juan Maroñas Molano

Directores: Roberto Paredes Palacios
Daniel Ramos Castro

This dissertation is submitted for the degree of
Doctor of Philosophy

Department of Computer Science

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

I received a new Ballade from Chopin. It seems to be a work closest to his genius (although not the most ingenious) and I told him that I like it best of all his compositions. After quite a lengthy silence he replied with emphasis, 'I am happy to hear this since I too like it most and hold it dearest.

Robert Schumann

Summary

This thesis is framed at the intersection between modern Machine Learning techniques, such as Deep Neural Networks, and reliable probabilistic modeling. In many machine learning applications, we do not only care about the prediction made by a model (e.g. *this lung image presents cancer*) but also in how confident is the model in making this prediction (e.g. *this lung image presents cancer with* 67% *probability*). In such applications, the model assists the decision-maker (in this case a doctor) towards taking the final decision. As a consequence, one needs that the probabilities provided by a model reflects the true underlying set of outcomes, otherwise the model is useless in practice. When this happens, we say that a model is perfectly calibrated.

Bayes Decision Rule provides a principled framework for decision making under uncertainty and guarantees optimal performance (i.e. minimum error probabilities). For Bayes Decision Rule to work, one needs to use a calibrated model since that implies that the model has (better) recovered the data generating distribution. Calibration is not the only thing that matters, but also the refinement which is the ability of the classifier to recover how the data of the different classes are separated.

However, modern machine learning techniques, such as Deep Neural Networks, are uncalibrated, which compromises their deployment in high-risk applications. Many works have attempted to solve the miscalibration of modern Deep Neural Networks, and this is one of the main objectives of this thesis.

This thesis starts by reviewing the elements involved in Bayes Decision Rule, through the lens of Proper Scoring Rules. This let us introduce one of the key concepts, at least in my personal opinion, that one should take into consideration in order to train a machine learning model. This is the data uncertainty in the target distribution, i.e. how the different samples from the different classes overlap. This observation is used in the first contribution of this thesis to justify why any Data Augmentation techniques will not guarantee calibrated distributions, even though empirical evidence has been provided in the opposite direction. We show how a proposed loss function that takes into account data uncertainty solves the miscalibration introduced by Mixup training, a state-of-the-art Data Augmentation technique.

However, since Deep Neural Networks are expensive models to train, techniques that aim at implicitly calibrate these models are costly to be deployed in practice since one has to deal with model selection techniques. To this end, the second contribution proposes to recalibrate the output of a Deep Neural Network using a Bayesian Neural Network. With this, we show that one can use expressive models as long as uncertainty is incorporated, which contrasts with many of the recent contributions that hypothesize that the calibration space is inherently simple because simpler techniques work better than their complex counterparts. We also show that the main criticism of Bayesian techniques, when applied to modern Neural Networks, is overcome by combining the capabilities of Deep Neural Networks with the proposed decoupled Bayesian Neural Network.

One of the problems of Bayesian Neural Networks is the specification of a meaningful prior in the parameter space that induces a useful prior in the function space. A bad specified prior will wrongly bias the way we quantify uncertainty with the posterior, leading to suboptimal Bayesian predictions. In the final contribution of this thesis, we introduce a new prior that directly applies to the function space, named the Transformed Gaussian Process. This new prior over functions is constructed by warping samples from a Gaussian Process using an invertible transformation. These warping functions are parameterized by Bayesian Neural Networks, which allow us to model non-stationary processes accounting for parameter uncertainty, clearly improving the performance over the point estimate counterpart. We introduce a sparse variational inference algorithm that allows us to lighten the computational burden that we would inherit from standard Gaussian Processes, to target the intractable posterior, to train the model using Stochastic variational inference and to use any observation model; among other nice properties.

Resumen

Esta tesis se enmarca en la intersección entre las técnicas modernas de Machine Learning, como las Redes Neuronales Profundas, y el modelado probabilístico confiable. En muchas aplicaciones, no solo nos importa la predicción hecha por un modelo (por ejemplo *esta imagen de pulmón presenta cáncer*) sino también la confianza que tiene el modelo para hacer esta predicción (por ejemplo *esta imagen de pulmón presenta cáncer con* 67% *probabilidad*). En tales aplicaciones, el modelo ayuda al tomador de decisiones (en este caso un médico) a tomar la decisión final. Como consecuencia, es necesario que las probabilidades proporcionadas por un modelo reflejen las proporciones reales presentes en el conjunto al que se ha asignado dichas probabilidades; de lo contrario, el modelo es inútil en la práctica. Cuando esto sucede, decimos que un modelo está perfectamente calibrado.

La regla de decisión de Bayes proporciona un marco fundamentado para la toma de decisiones en condiciones de incertidumbre y garantiza un rendimiento óptimo (es decir, probabilidades mínimas de error). Para que funcione la regla de decisión de Bayes, es necesario utilizar un modelo calibrado, ya que eso implica que el modelo ha recuperado (approximado mejor) la distribución de generación de datos. La calibración no es lo único que importa, sino también el refinamiento que es la capacidad del clasificador para recuperar cómo se separan los datos de las diferentes clases.

Sin embargo, las técnicas modernas de aprendizaje automático, como las redes neuronales profundas, no están calibradas, lo que compromete su implementación en aplicaciones de alto riesgo. Numerosos trabajos han intentado

solucionar el error de calibración de las redes neuronales profundas modernas, y este es uno de los principales objetivos de esta tesis.

Esta tesis comienza revisando los elementos involucrados en la regla de decisión de Bayes, a través de las denominadas *Proper Scoring Rules*. Esto nos permite introducir uno de los conceptos clave, al menos en mi opinión personal, que se debe tener en cuenta para entrenar un modelo de aprendizaje automático. Esta es la incertidumbre de los datos en la distribución objetivo, es decir, cómo se superponen las diferentes muestras de las diferentes clases. Esta observación se utiliza en la primera contribución de esta tesis para justificar por qué cualquier técnica de aumento de datos no garantizará distribuciones calibradas, a pesar de que la evidencia empírica se ha proporcionado en la dirección opuesta. Mostramos cómo una función de pérdida propuesta que tiene en cuenta la incertidumbre de los datos resuelve el error de calibración introducido por el entrenamiento Mixup, una técnica de aumento de datos de última generación.

Sin embargo, dado que las redes neuronales profundas son modelos costosos de entrenar, las técnicas que tienen como objetivo calibrar implícitamente estos modelos son costosas de implementar en la práctica, ya que uno tiene que lidiar con técnicas de selección de modelos. Para ello, la segunda contribución propone recalibrar la salida de una Red Neural Profunda usando una Red Neural Bayesiana. Con esto, mostramos que se pueden usar modelos expresivos siempre que se incorpore la incertidumbre, lo que contrasta con muchas de las contribuciones recientes que plantean la hipótesis de que el espacio de calibración es inherentemente simple porque las técnicas más simples funcionan mejor que sus contrapartes complejas. También mostramos que la principal crítica de las técnicas bayesianas, cuando se aplican a las redes neuronales modernas, se supera combinando las capacidades de las redes neuronales profundas con la red neuronal bayesiana desacoplada propuesta.

Uno de los problemas de las redes neuronales bayesianas es la especificación de un prior significativo en el espacio de parámetros que induce un prior útil en el espacio funcional. Un priori mal especificado sesgará erróneamente la forma en que cuantificamos la incertidumbre con el posterior, lo que conducirá a predicciones bayesianas subóptimas. En la contribución final de esta tesis, presentamos un nuevo previo que se aplica directamente al espacio funcional, denominado *Transformed Gaussian Process*. Esta nuevo prior sobre el espacio de funciones se construye deformando muestras de un proceso gaussiano utilizando una transformación invertible. Estas funciones de deformación están parametrizadas por redes neuronales bayesianas, que nos permiten modelar procesos no estacionarios que tienen en cuenta la incertidumbre de los parámetros, mejorando claramente el rendimiento sobre La estimación puntual. Intro-

ducimos un algoritmo de inferencia variacional escasa que nos permite aligerar la carga computacional que heredaríamos de los procesos gaussianos estándar, aproximar al posterior intratable, entrenar el modelo usando inferencia variacional estocástica y usar cualquier modelo de observación; entre otras propiedades.

Resum

Aquesta tesi s'emmarca en la intersecció entre les tècniques modernes d'aprenentatge automàtic, com les xarxes neuronals profundes, i el modelatge probabilístic fiable. En moltes aplicacions, no només ens preocupa la predicció feta per un model (*aquesta imatge pulmonar presenta càncer*), sinó també la confiança del model per fer aquesta predicció (*aquesta imatge pulmonar) presenta càncer amb probabilitat de 67* %). En aquestes aplicacions, el model ajuda el que pren la decisió (en aquest cas un metge) a prendre la decisió final. Com a conseqüència, cal que les probabilitats que proporciona un model reflecteixin el veritable conjunt subjacent de resultats, en cas contrari, el model no serà inútil a la pràctica. Quan això passa, diem que un model està perfectament calibrat.

La regla de decisió de Bayes proporciona un marc de principis per a la presa de decisions sota incertesa i garanteix un rendiment òptim (és a dir, probabilitats mínimes d'error). Perquè la regla de decisió de Bayes funcioni, cal utilitzar un model calibrat, ja que això implica que el model ha recuperat (millor) la distribució que genera dades. La calibració no és l'únic que importa, sinó també el refinament que és la capacitat del classificador per recuperar la forma en què es separen les dades de les diferents classes.

Tot i això, les tècniques modernes d'aprenentatge automàtic, com les xarxes neuronals profundes, no estan calibrades, cosa que compromet el seu desplegament en aplicacions d'alt risc. Molts treballs han intentat resoldre l'error de calibració de les xarxes neuronals profundes modernes, i aquest és un dels principals objectius d'aquesta tesi.

Aquesta tesi comença revisant els elements implicats en la regla de decisió de Bayes, a través de la lent de les regles de puntuació adequades. Això ens permet introduir un dels conceptes clau, almenys en la meva opinió personal, que s'hauria de tenir en compte per formar un model d'aprenentatge automàtic. Aquesta és la incertesa de les dades en la distribució objectiu, és a dir, com se superposen les diferents mostres de les diferents classes. Aquesta observació s'utilitza en la primera contribució d'aquesta tesi per justificar per què qualsevol tècnica d'augment de dades no garantirà distribucions calibrades, tot i que s'han aportat proves empíriques en la direcció contrària. Mostrem com una proposta de funció de pèrdua que té en compte la incertesa de les dades resol el mal calibrat introduït per l'entrenament Mixup, una tècnica d'augment de dades d'última generació.

Tanmateix, atès que les xarxes neuronals profundes són models cars per entrenar, les tècniques que volen calibrar implícitament aquests models costen de desplegar-se a la pràctica, ja que s'ha de fer front a tècniques de selecció de models. Per a això, la segona contribució proposa recalibrar la sortida d'una xarxa neuronal profunda mitjançant una xarxa neuronal bayesiana. Amb això, demostrem que es poden utilitzar models expressius sempre que s'incorpori la incertesa, cosa que contrasta amb moltes de les recents contribucions que hipoteten que l'espai de calibratge és intrínsecament senzill perquè les tècniques més senzilles funcionen millor que les seves contraparts complexes. També demostrem que les principals crítiques a les tècniques bayesianes, quan s'apliquen a les xarxes neuronals modernes, es superen combinant les capacitats de les xarxes neuronals profundes amb la xarxa neuronal bayesiana desacoblada proposada.

Un dels problemes de les xarxes neuronals bayesianes és l'especificació d'un prior significatiu a l'espai de paràmetres que indueix un prior útil a l'espai de funcions. Un anterior especificat malament esbiaixarà erròniament la manera de quantificar la incertesa amb el posterior, la qual cosa conduirà a prediccions bayesianes subòptimes. En la contribució final d'aquesta tesi, introduïm un nou prior que s'aplica directament a l'espai de funcions, anomenat Procés Gaussià Transformat. Aquest nou anterior sobre funcions es construeix mitjançant la deformació de mostres d'un procés gaussià mitjançant una transformació inversible. Aquestes funcions de deformació estan parametritzades per les xarxes neuronals bayesianes, que ens permeten modelar processos no estacionaris que tinguin en compte la incertesa dels paràmetres, millorant clarament el rendiment respecte a la contrapart estimada per punts. Introduïm un algorisme d'inferència variacional escassa que ens permet alleugerir la càrrega computacional que heretaríem dels processos gaussians estàndard, orientar-nos

a la part posterior intractable, entrenar el model mitjançant la inferència variacional estocàstica i utilitzar qualsevol model d'observació; entre altres boniques propietats.

Acronyms

MLE: Maximum Likelihood Estimation.

ERM: Empirical Risk Minimization

VRM: Vicinal Risk Minimization

KLD: Kullback Leibler Divergence

DNN: Deep Neural Network

CNN: Convolutional Neural Network

BNN: Bayesian Neural Network

DA: Data Augmentation

TS: Temperature Scaling

NE: Network Ensembles

MAP: Maximum A Posterior

SGD: Stochastic Gradient Descent

RHS: Right Hand Side

ARC: Auto Regularized Confidence

ARC_V1: Version 1 of Auto Regularized Confidence

ARC_V2: Version 2 of Auto Regularized Confidence

CCE: Categorical Cross Entropy

MMCE: Maximum Mean Calibration Error

ELBO: Evidence Lower Bound

ELL: Expected Log Likelihood

VUE: Variance Under Estimation

VI: Variational Inference

MFVILR: Mean-field Variational Inference with Local Reparameterization

MFVI: Mean-field Variational Inference

GP: Gaussian Process

DSVI: Double Stochastic Variational Inference

DGP: Deep Gaussian Process

WGP: Warped Gaussian Process

V-WGP: Variational Warped Gaussian Process

SVGP: Sparse Variational Gaussian Process

TGP: Transformed Gaussian Process

BATGP: Input-dependent Bayesian Transformed Gaussian Process

PETGP: Input-dependent Point Estimate Transformed Gaussian Process

MCMC: Markov Chain Monte Carlo

HMC: Hamiltonian Monte Carlo

NUTS: No U-Turn Sampler

MC: Monte Carlo

BDR: Bayes Decision Rule

BRisk: Bayes Risk

PSR: Proper Scoring Rule

BS: Brier Score

LS: Logarithmic Score

ECE: Expected Calibration Error

MCE: Maximum Calibration Error

ACC: Accuracy

RMSE: Root Mean Squared Error

NLL: Negative Log-likelihood

SAL: Sinh-Arcsinh-Linear

MCDROP: Monte Carlo Dropout

VWCI: Variance-Weighted Confidence-Integrated Loss

SP: Softplus

Acknowledgments

First, I will like to acknowledge my family, since that is the most important thing one has in any step taken during his life.

On the other hand, I thank Roberto and Daniel for being my advisors and giving me the opportunity to pursue a PhD, and obviously to the PRHLT research center. They support me and provide advice in different ways, which have made me a better researcher, and even more important, a better person. I thank Jose Miguel Benedí as well for opening the doors of his office each time I needed.

Moreover, I am super proud of having such wonderful research internship advisors: Costantino Grana and Theo Damoulas. Both gave me the opportunity to feel part of his group during my collaborations and I am really happy with the results obtained during these internships.

During my PhD, I met wonderful people like Mario Parreño, Salva, Federico Bolleli, Federico Pollastri, Laura, Michele, Stefano, Jeremias, Patrick, James, Ayman, Virginia, Maud, Kangrui, Deniz, Ollie and many more both in the Warwick Machine Learning group, the AimageLab and the PRHLT.

Finally, I really thank all the people involved in this thesis review period either as a principal or substitute reviewer, since in any case, they allocated time for being able to review this manuscript. This includs Daniel Hernandez Lobato, Luciana Ferrer, Thang D. Bui, Arno Solin, Stefanos Eleftheriadis, Francois-Xavier Briol, Carlos Alaiz, Mauricio Alvarez, Paco Casacuberta, Jose Miguel Benedí and Michael Thomas Smith.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Describing the processes that drive the evolution of nature is one of the classical concerns of human being. The field of Machine Learning or Pattern Recognition (which in my opinion are different ways of talking about the same thing) addresses this objective from the perspective of assuming that the sources of variation to be modeled can be described by some unknown function $\mathcal{F}^*$ (that usually represents some degree of uncertainty), which is learned by feeding some data $\mathcal{D}$ into an algorithm $\mathcal{A}$.

The success of the representation learned by $\mathcal{F}^*$ mainly depends on the expressiveness and inductive bias from the set of possible $\mathcal{F}$, the quality of $\mathcal{D}$ in representing the task to learn, and the effectiveness and time required by $\mathcal{A}$ to select the best possible $\mathcal{F}^*$.

For this reason, it wasn't until the introduction of the AlexNet in 2012 (Krizhevsky et al. 2012), when practitioners realized that Machine Learning techniques could suppose a breakthrough in the way we describe nature. Since then, Machine Learning algorithms have shown an outstanding representation power in many areas of human understanding like for example Machine Translation (Vaswani et al. 2017), Speech Recognition (Graves et al. 2013) or Computer Vision (He et al. 2016a).

In my opinion, the contributions of this paper have also inspired ways of boosting other classical branches from this field such as Gaussian Processes (GP) (Rasmussen et al. 2005) or Markov Chain Monte Carlo (MCMC) algorithms (Brooks et al. 2011); by the use of for example: parallel computations in modern GPUs, automatic differentiation, or the use of stochastic gradients to scale computation to large datasets.

1.1 Motivation

Due to the versatility of these models, especially those parameterized by Deep Neural Networks (DNN), practitioners use them in applications where we do not only care about the decision to be taken but also about its reliability – for example when using a Convolutional Neural Network (CNN) to provide a confidence for an image presenting some form of disease (Pollastri et al. 2021b), see Figure 1.1.

In such situations, optimal performance (w.r.t. some loss function) is guaranteed if one takes an action based on Bayes Decision Rule (BDR), where a correct modeling of data uncertainty is mandatory. However, the expressiveness and inductive biases of DNN, which is what makes them an appealing tool to represent high dimensional distributions in a successful way, is not accompanied by a correct modeling of this data uncertainty. In other words, DNN are not well-calibrated. This fact limits their applicability in high risk decision applications, such as medical diagnosis (Caruana et al. 2015), self-driving cars (Bojarski et al. 2016), robotics (Kober et al. 2013), forensic science (Ramos et al. 2021) and many more.

In other words, one can achieve optimal performance if decisions are taken using the probability distribution that has generated the data. However, beyond simple cases[1], characterizing a complete distribution requires storing an infinity amount of samples. As a consequence, the goal of a practitioner is to set a model $\mathcal{F}^*$ as close as possible to this data generating distribution. To do so, several strategies can be used, such as: encoding desirable inductive biases in the set of possible $\mathcal{F}$, optimizing the expected value of a proper scoring rule (Degroot et al. 1983), minimizing a distance or divergence between the model and the data distribution, account for epistemic uncertainty by integrating out the model parameters (Wilson et al. 2020), encoding prior beliefs in the modeling process (Rasmussen et al. 2005), combining several probabilistic models (Dietterich 2000) etc.

[1] For example the trivial case in which data is generated from some known distribution.

1.1.1 Intuition behind the necessity of having calibrated classifiers.

An intuitive motivation to the necessity of having calibrated classifiers is exposed by considering a task where the reliability of the decision to be taken is important.

Figure 1.1 shows a renal biopsy captured on a fluorescence microscope presenting *mesangiale* and *parietale* patterns. The labeling procedure of this pattern is ambiguous among experts. Thus, in this task, our goal as machine learning practitioners is not to provide the final decision on whether an image presents the pattern or not, but to assist the medical expert giving a degree of belief (in form of a probability distribution) towards the image presenting the possible patterns. The medical expert uses our information and combines it with its expert knowledge towards taking a final action, for example, perform or not perform a surgery or perform or not perform additional costly diagnostic tests.

From this example, it is clear that our probability distribution will only be useful if it reflects the true state of nature, i.e. if it is *reliable*. For example, if we say that an image presents *mesangiale patterns* with 0.51 probabilitiy, then the medical expert might decide not to make additional tests since the

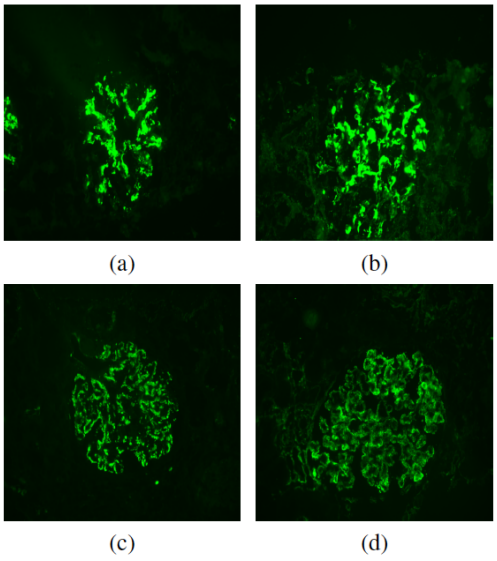


Figure 1.1: Figure obtained from (Pollastri et al. 2021b) showing images of a renal biopsy captured with a fluorescence microscope presenting *mesangial* (a),(b) and *parietale* (c),(d) patterns. Its labeling is ambiguous among experts.

probability of the pattern being present is just 0.51. But if we then take 1000 images presenting similar patterns and 995 actually present a *mesangiale pattern*, then the medical expert would have taken a decision with drastic consequences since the real proportion of samples presenting the pattern is 0.995 and thus, for this particular case, the medical expert would have decided to make additional checks. In this case, our confidence 0.51 is not reliable (is not well-calibrated) since it does not represent the real proportion of 0.995.

When a probability distribution reflects this proportion (this data uncertainty), we say that it is calibrated, but we will see later that calibration implies optimality in a much deeper sense, through the lens of Bayes Decision Rule.

1.2 Overview of Contributions

This thesis is focused on the intersection between DNN and model calibration, with a special emphasis on how accounting for different types of uncertainty can help to yield more calibrated predictions. In particular, the contributions of this thesis are three fold.

First, we make use of Bayes Decision Rule to justify why DNN trained with data augmentation techniques are not necessarily calibrated, in contrast with recent contributions. We show that a popular DA technique, named *mixup*, does not necessarily provide calibrated distributions and we show how to fix this by proposing a loss function that takes into account data uncertainty (Maroñas et al. 2021a). This opens a new perspective to design new loss functions to train reliable probabilistic models.

Second, we show how DNN can be calibrated by using an expressive post calibration stage if one takes into account parameter uncertainty. So rather than designing DNN that are explicitly calibrated, for example by using a deep ensemble, a much lighter model is used to map the uncalibrated logits from DNN into calibrated ones. For the purpose of this mapping we make use of a Bayesian Neural Network (BNN) with mean-field variational inference (Maroñas et al. 2020). We provide a wide analysis on the usage of BNN for this task and open new perspectives for future improvements based e.g. on robustness to prior misspecification.

Third, we propose a new non-parametric non-stationary prior over functions, named the Transformed Gaussian Processes (TGP) (Maroñas et al. 2021b). We show that, beyond other nice properties, TGP can provide much better calibrated outputs than standard Gaussian Processes (GP), and similar pre-

dictions to Deep Gaussian Processes (DGP) (Damianou et al. 2013), but at a much lower computational cost. Thus, the TGP is a candidate for further exploration to be used as a post-hoc calibration stage, something to be done in future work.

1.3 Thesis Structure

The remaining content of the thesis is organized as follows:

Chapter 2: this chapter serves as an introduction and motivation of the concepts and techniques that are used in the subsequent chapters.

Chapter 3: this chapter introduces a new paradigm in designing loss functions for probabilistic models that encourages the model to learn calibrated distributions using the uncertainty presented in the data and the learned representations.

Chapter 4: this chapter shows how can we can calibrate Deep Neural Networks by mapping its uncalibrated logits using a Bayesian Neural Network.

Chapter 5: this chapter introduces the Transformed Gaussian Process, a non-parametric non-stationary function over priors that can correctly model uncertainty at a fraction of the computational cost.

Chapter 6: this chapter presents conclusions and future lines of direction.

The complete list of references and contributions achieved during the three year period of this thesis (February 2018 - November 2020) are listed below:

- Transforming Gaussian Processes with Normalizing Flows, AISTATS 2021, (Maroñas et al. 2021b), *First author.*
- Calibration of Deep Probabilistic Models with Decoupled Bayesian Neural Networks, Neurocomputing 2020, (Maroñas et al. 2020), *First author.*
- On Calibration of Mixup Training For Deep Neural Networks, SSPR 2021, (Maroñas et al. 2021a), *First author.*
- Confidence Calibration for Deep Renal Biopsy Immunofluorescence Image Classification, ICPR 2020, (Pollastri et al. 2021b), *Second author.*

- Third position in 2019 ISIC cancer detection challenge, *First Author*.
- A Deep Analysis on High Resolution Dermoscopic Image Classification, Revista. (Pollastri et al. 2021a) *Third Author*. This paper extends the work and models used in the ISIC competition.
- Improving Calibration by Considering Uncertainty in Feature-Based LA-ICP-MS Forensic Glass Comparison, (Ramos et al. 2021), *First author*.
- CULAYERS, a GPU library based on CUDA that is used in the EDDL Library, a high performing computing library used within the Deep Health European Project, https://deephealth-project.eu/
- Generative models for deep learning with very scarce data, CIARP 2018, (Maroñas et al. 2019), *First author*.

Chapter 2

Background

This chapter introduces the use of Bayes Decision Rule through Proper Scoring Rules and information theory. We then show the caveats of approximating data generating distributions with a model and motivate the use of uncertainty quantification through Bayesian modeling.

2.1 Bayes Decision Rule

We start by introducing Bayes Decision Rule (BDR) in the context of class conditional probabilistic classifiers, although this exposition also holds for regression and unconditional modeling. We briefly show the connection at the end of this first subsection.

2.1.1 Classification

Throughout this exposition $\mathbf{X} \in \mathcal{X} \subset \mathbb{R}^d$ will refer to the input of a class conditional probability distribution $p(\mathbf{Y}|\mathbf{X})$ and $\mathbf{Y} \in \mathcal{Y} \subset \mathbb{N}$ to its categorical label with C possible states. We will refer to the probability assigned towards a particular class c as $p(\mathbf{Y} = c|\mathbf{X})$. With $\mathcal{D} = \{\mathbf{X}_n, \mathbf{Y}_n\}_{n=1}^N$ we denote a set of N observations assumed to be drawn i.i.d. from $p(\mathbf{Y}|\mathbf{X})p(\mathbf{X})$.

BDR relies on taking the action α^* that minimizes the conditional risk (BRisk):

$$\begin{aligned} \textsc{BRisk}(\alpha|\mathbf{X},\mathbf{Y}) &= \sum_{c=1}^{C} \mathcal{L}(\alpha,c)p(\mathbf{Y}=c|\mathbf{X}) \\ \alpha^* &= \underset{1\leq c\leq C}{\operatorname{argmin}}\, \textsc{BRisk}(\alpha=c|\mathbf{X},\mathbf{Y}) \end{aligned} \tag{2.1}$$

where $\mathcal{L}(\alpha, c)$ is a loss function which measures the discrepancy between the action taken α and the true state of nature c. An example of a loss function in a regression problem is the squared loss:

$$\mathcal{L}(\alpha,c) = (\alpha - c)^2 \tag{2.2}$$

In classification scenarios, however, it is more intuitive to define loss functions that directly penalize the action taken: *assign class* α when the value of the ground truth class is c. One consequence is that this loss function can encode background knowledge about the problem in which this probabilistic classifier is being used, and we illustrate it through an example.

Example 2.1.1. *Consider a binary classification problem in which a decision maker*[1] *receives an image* $\mathbf{X}$ *and has to decide if this image presents a lung cancer* $\mathbf{Y} = 1$ *or the patient is healthy* $\mathbf{Y} = 0$*. Given the image, the posterior probability* $p(\mathbf{Y}|\mathbf{X})$ *encodes the degree of uncertainty around the two possible states. Note that, in this context, the goal of the probabilistic classifier is not to decide which is the correct class, but to assist the oncologist by providing a confidence that can be used towards making the final decision. For example, if the probabilistic classifier assigns a confidence* 0.99 *towards* $\mathbf{Y} = 1$*, then the oncologist will be biased towards deciding action:* decide cancer *than if the probability is* 0.53*. In this scenario,* BDR *allows the oncologist to encode his expert knowledge in the loss function so that minimizing* BRisk *provides the optimal action. A possible loss function for this problem can be:*

$\mathcal{L}(\alpha = 0, \mathbf{Y} = 1)$*: Decide* no cancer *when there is actually a cancer.*

$\mathcal{L}(\alpha = 1, \mathbf{Y} = 0)$*: Decide* cancer *when there is actually* no cancer*.*

$\mathcal{L}(\alpha = 0, \mathbf{Y} = 0)$*: Decide* no cancer *when there is actually* no cancer*.*

$\mathcal{L}(\alpha = 1, \mathbf{Y} = 1)$*: Decide* cancer *when there is actually* cancer*.*

where reasonable values for these losses are given in the following table:

[1] An oncologist in this case.

α \ c	0	1
0	0.05	0.8
1	0.15	0.0

With these losses, being wrong when taking the action $\alpha = 0$ *(*decide no cancer*) is much more penalized (0.8) than the opposite (0.0). This is to be expected as it is undesirable to decide no cancer when there is a cancer. This is also encoded through the loss* $\mathcal{L}(\alpha = 0, \mathbf{Y} = 0)$ *having a value different from* 0.0*, because this penalizes deciding no cancer even when the patient image does not present cancer. With this setting, the decision threshold to take action* $\alpha = 0$ *is given by* $p(\mathbf{Y} = 0|\mathbf{X}) > 0.89$*, which is a much more restrictive probability than the usual threshold* $\alpha = 0.5$*, which is to be expected as deciding* no cancer *can have drastic consequences.*

It can be shown, that taking the action with lower BRisk guarantees optimal performance (Duda et al. 2000), i.e. it minimizes the probability of error for a given $\mathbf{X}$. Thus, this decision procedure minimizes the expected Risk or error probability, given by[2]:

$$\begin{aligned} p(\text{error}) = \int \text{BRisk}(\alpha|\mathbf{X}, \mathbf{Y}) p(\mathbf{X}) \mathrm{d}\mathbf{X} = \\ \int \sum_{c=1}^{C} \mathcal{L}(\alpha, c) p(\mathbf{Y} = c|\mathbf{X}) p(\mathbf{X}) \mathrm{d}\mathbf{X} \end{aligned} \tag{2.3}$$

Note that in the case in which we use a zero-one Loss $\mathcal{L}(\alpha, c) = 0,\ \alpha = c$ and $\mathcal{L}(\alpha, c) = 1,\ \alpha \neq c$ (the case in which each error is equally important), then the optimal error probability is the minimum error rate of the classifier:

$$p^*(\text{error}) = \int \left[1 - p(\mathbf{Y} = \alpha^*|\mathbf{X})\right] p(\mathbf{X}) \mathrm{d}\mathbf{X} \tag{2.4}$$

which is the common rule used when benchmarking models in the context of e.g. image classification (Deng et al. 2009). In this case the conditional risk is

[2]This probability of error is known as Bayes Risk. We note that in this work, our notation BRisk might led to confusion. However, we do so to differentiate it from the procedure of empirical risk minimization discussed in subsequent chapters.

minimized by using the Maximum a Posterior (MAP) rule, i.e. select $\alpha^* = c$ if $p(\mathbf{Y} = c|\mathbf{X}) > p(\mathbf{Y} \neq c|\mathbf{X})$.

We shall note that one might not see the connection between minimum error rate and optimal error probability directly. The error rate is commonly computed by the proportion of incorrect samples over the total amount of samples, while the error probability depends directly on the average of the probabilities assigned to a set of samples; so we might initially think that these two concepts are different. However, if we are given a set of 10 inputs $\mathbf{X}$ all of them belonging to class c with 0.9 probability, then if this probability distribution reflects the true state of nature, we would expect that 9 out of 10 samples belong to class c. Thus, the accuracy of this classifier will be $9/10$ (so the error rate is 0.1) which coincides with the optimal error probability, given by $1 - 0.9 = 0.1$.

2.1.2 Regression and Unconditional Modeling

To complete our introduction of BDR we shall briefly discuss the cases in which rather than a class conditional probability we directly model $p(\mathbf{X})$, and we do it jointly with the discussion of a regression model where the data $\mathbf{X}$ is modeled as a Gaussian distribution: $p(\mathbf{X}) = \mathcal{N}(\mathbf{X}|\mu, \sigma^2)$.

Contrary to common text books (e.g. (Murphy 2021)) we derive the optimal action using the absolute loss, and let the square loss derivation as an exercise. This optimal action minimizes the error probability, which in this case is given by:

$$\begin{aligned} p(\text{error}) = \int |\alpha - \mathbf{X}| p(\mathbf{X}|\mu, \sigma^2) \mathrm{d}\mathbf{X} = \\ \int_{-\infty}^{\alpha} (\alpha - \mathbf{X}) p(\mathbf{X}|\mu, \sigma^2) \mathrm{d}\mathbf{X} + \int_{\alpha}^{\infty} (\mathbf{X} - \alpha) p(\mathbf{X}|\mu, \sigma^2) \mathrm{d}\mathbf{X} \end{aligned} \tag{2.5}$$

We just have to check the following two conditions:

$$\begin{aligned} \frac{\mathrm{d}p(\text{error})}{\mathrm{d}\alpha} = 0 \\ \frac{\mathrm{d}^2 p(\text{error})}{\mathrm{d}\alpha^2} > 0 \end{aligned} \tag{2.6}$$

Starting with the first derivative, we rely on Leibniz integral rule to obtain the derivative of the integration limits. This yields, for the first term in the RHS of Equation 2.5:

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}\alpha}\int_{-\infty}^{\alpha}(\alpha-\mathbf{X})p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}=\\&(\alpha-\alpha)p(\alpha)+\int_{-\infty}^{\alpha}\frac{\mathrm{d}}{\mathrm{d}\alpha}(\alpha-\mathbf{X})p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}=\\&\int_{-\infty}^{\alpha}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}\end{aligned} \tag{2.7}$$

Similarly, the second therm in RHS is given by $-\int_{\alpha}^{\infty}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}$. By plug in this derivation in Equation 2.5, and then equaling 0, we arrive at:

$$\int_{-\infty}^{\alpha}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}=\int_{\alpha}^{\infty}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X} \tag{2.8}$$

and the value at which both integrals are equal is the $\alpha = 0.5$ quantile also known as the median[3]. For the second derivative, we start from the result of the first derivative obtained above. Now making use of the first fundamental theorem in calculus:

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}\alpha}\int_{-\infty}^{\alpha}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}-\int_{\alpha}^{\infty}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}=\\&\frac{\mathrm{d}}{\mathrm{d}\alpha}\int_{-\infty}^{\alpha}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}-\left[1-\int_{-\infty}^{\alpha}p(\mathbf{X}|\mu,\sigma^2)\mathrm{d}\mathbf{X}\right]=\\&2p(\alpha|\mu,\sigma^2)\geq 0\end{aligned} \tag{2.9}$$

for any value of α, since the probability density function is always either zero or positive. This shows that the optimal action to take given the absolute loss is the median of the distribution, which is μ in this case since the median of the Gaussian distribution is given by its mean μ.

A similar analysis shows that for the squared error loss in Equation 2.2, the value of α that minimizes the error probability is given by the mean μ.

[3]Note that these integrals represent the definition of cumulative distribution functions.

2.2 Calibration and Refinement

In the introduction of BDR we have assumed that we have access to the data generating distribution $p(\mathbf{Y}, \mathbf{X})$. However, in practice, since this distribution $p(\mathbf{Y}, \mathbf{X})$ is unknown, a practitioner has to define a model $p_\theta(\mathbf{Y}, \mathbf{X})$ parameterized by θ and define $\mathcal{A}$ so that, in the ideal case, $p_\theta(\mathbf{Y}, \mathbf{X}) = p(\mathbf{Y}, \mathbf{X})$. The trivial case, implies having an infinite amount of data, i.e. the entire distribution, where the model is given by the empirical distribution:

$$p_\theta(\mathbf{Y}, \mathbf{X}) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \delta(\mathbf{X} - \mathbf{X}_n)\delta(\mathbf{Y} - \mathbf{Y}_n) = p(\mathbf{Y}, \mathbf{X}) \tag{2.10}$$

with $\delta(\mathbf{X})$ being the Dirac measure. This is obviously impractical in real scenarios.

In this subsection we introduce the concept of calibration and refinement through the lens of Proper Scoring Rules and information theory and, at the end of this section, we will show the relationship between many other techniques used in the learning process e.g. maximum likelihood, Bayesian inference, divergence minimization or empirical risk minimization, among others; showing they are all equivalent.

This calibration and refinement decomposition will allow us to analyze the process of approximating a distribution $p(\mathbf{X}, \mathbf{Y})$ with a model $p_\theta(\mathbf{X}, \mathbf{Y})$, with the purpose of being used as a motivation behind some of the techniques presented later in this thesis.

2.2.1 Proper Scoring Rules

Let's denote with $t \in \mathcal{P}$ to a probability distribution assigned to a particular $\mathbf{X}_n$, i.e. $t = p_\theta(\mathbf{Y}|\mathbf{X}_n)$, where $\mathcal{P}$ is the set of all probability distributions on $\mathcal{Y}$. In this case, $p_\theta(\mathbf{Y}|\mathbf{X}_n)$ just represents our guess about the probability distribution over $\mathbf{Y}$ in form of a vector. So this can be, for example, the output of a Deep Neural Network feed into a softmax activation function[4].

(Informally[5]) a scoring rule is a function $S : \mathcal{P} \times \mathcal{Y} \to \mathbb{R}$ that provides a reward based on your reported belief t and the true outcome $\mathbf{Y}_n$. Two examples of

[4]Note that we don't assume that $p_\theta(\mathbf{Y}|\mathbf{X}_n)$ is a probability distribution as the Categorical likelihood, but just a vector of probabilities obtained by our model θ.

[5]See (Gneiting et al. 2007) for an exact definition.

well known scoring rules for classification scenarios are the Brier Score (BS) and the Log Score (LS):

$$\begin{aligned} \text{LS} &= \log p_\theta(\mathbf{Y} = \mathbf{Y}_n | \mathbf{X}_n) \\ \text{BS} &= \sum_{c=1}^{C} (\mathbb{1}_{[c=\mathbf{Y}_n]} - p_\theta(\mathbf{Y} = c | \mathbf{X}_n))^2 \end{aligned} \tag{2.11}$$

Given another distribution $q(\mathbf{Y}) \in \mathcal{P}$, we can define the expected score as:

$$S(t, q(\mathbf{Y})) = \sum_{c=1}^{C} q(\mathbf{Y} = c) S(t, c) \tag{2.12}$$

where $q(\mathbf{Y})$ is an object representing optimality in some sense, in this case being the proportion of true outcomes[6]. So this is, basically, the average of all your scoring rules weighted by the frequency of a particular label $\mathbf{Y}$. It must hold that:

$$S(t, q(\mathbf{Y})) \leq S(q(\mathbf{Y}), q(\mathbf{Y})) \tag{2.13}$$

for the scoring rule to be proper and positive oriented. This means that the value at which the scoring rule is maximum is the value in which your predicted probabilities t are equal to the ground truth distribution $q(\mathbf{Y})$. If the Proper Scoring Rule (PSR) is negatively oriented then the above inequality changes the direction. The PSR is said to be strictly proper if the above inequality is strict. An example of a negative and positive oriented scoring rules are the Brier Score and the Logarithmic Score respectively.

Note that if we want to consider all possible inputs $\mathbf{X}$ then we must take the expected value, yielding the final expected score defined as:

$$S(t, q(\mathbf{Y})) = \int p(\mathbf{X}) \sum_{c=1}^{C} q(\mathbf{Y} = c | \mathbf{X}) S(t, c) \mathrm{d}\mathbf{X} \tag{2.14}$$

Since in practice we don't have access to the true state of nature represented by $p(\mathbf{X})q(\mathbf{Y}|\mathbf{X})$ we approximate the above expectation by taking the expected

[6]This has been so far denoted by $p(\mathbf{Y}|\mathbf{X})$ but we change the use of p by q to make the difference between the model and the data distribution more visually clear.

value over the empirical distribution $q(\mathbf{Y}|\mathbf{X})p(\mathbf{X}) = \frac{1}{N}\sum_{n=1}^{N}\delta(\mathbf{Y}-\mathbf{Y}_n)\delta(\mathbf{X}-\mathbf{X}_n)$[7], yielding the usual definition we see of these scoring rules in the literature:

$$\begin{aligned}\text{LS} &\approx \frac{1}{N}\sum_{n=1}^{N}\log p_\theta(\mathbf{Y}=\mathbf{Y}_n|\mathbf{X}_n)\\ \text{BS} &\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}(\mathbb{1}_{[c=\mathbf{Y}_n]}-p_\theta(\mathbf{Y}=c|\mathbf{X}_n))^2\end{aligned} \tag{2.15}$$

On the other hand, if for each sample $\mathbf{X}$, we are directly given the vector of probabilities $q(\mathbf{Y}|\mathbf{X}_n)$[8], then the expected score is given by:

$$\begin{aligned}\text{LS} &\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}q(\mathbf{Y}=c|\mathbf{X}_n)\log p_\theta(\mathbf{Y}=c|\mathbf{X}_n)\\ \text{BS} &\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}q(\mathbf{Y}=c|\mathbf{X}_n)\sum_{c'=1}^{C}(\mathbb{1}_{[c'=c]}-p_\theta(\mathbf{Y}=c'|\mathbf{X}_n))^2\end{aligned} \tag{2.16}$$

Note moreover that from Equation 2.13 one could be tempted to think that a PSR can be derived from a statistical divergence such as the Kullback-Leibler divergence (KLD). This is because statistical divergences are a measure of similarity between two probability distributions ($q(\mathbf{Y})$ and t here), which is only 0 when both distributions are the same. In fact, the reader is not misguided since Proper Scoring Rules are Bregman Divergences (Gneiting et al. 2007), being the KLD a member of this family. In fact the LS is derived from the KLD, while the Brier score is derived from the Euclidean distance. This generalization of scoring rules give us a way to generalize the concepts of entropy, cross-entropy and divergence, but more important, it allows us to construct any scoring rule from a Bregman Divergence of interest. For example since the KLD is non robust to outliers, we can use a robust divergence such as the β divergence that since it is also a Bregman Divergence, it allows us to construct a robust scoring rule.

From the definition of PSR, we can see that it is a way to evaluate the goodness of the predictions w.r.t. the target distribution, because the PSR is optimum if your predicted probabilities t equal the ground truth $q(\mathbf{Y})$. In practice, and

[7] This is equivalent to say that we approximate the expectation by a Monte Carlo where samples are obtained from $p(\mathbf{X}, \mathbf{Y})$.

[8] For example $\mathbf{X}$ is a tweet that has probability 0.8 of being misogynist and 0.2 of being racist.

in this particular discussion, t refers to the model probability $p_\theta(\mathbf{Y}|\mathbf{X})$ while the target distribution $q(\mathbf{Y})$ will be given by the data distribution $p(\mathbf{Y}|\mathbf{X})$. Thus it seems reasonable to optimize the expected score in order to learn a model that parameterizes our probability distribution of interest, because the optimal value of the PSR is the value at which your model has recovered the data distribution, and hence our decisions will be optimal attending to Bayes Decision Rule.

One of the interesting properties of PSR is that this goodness can be measured by three quantities into which the PSR can be decomposed, see Theorem 4 from (Degroot et al. 1983) and (Bröcker 2009). Due to their connection with information theory, we first define three information theory concepts:

Entropy: The entropy of a distribution $p(\mathbf{X})$ measures the information about the random variable $\mathbf{X}$, taking a minimum value of 0 when there is no randomness in $\mathbf{X}$. This quantity is defined as:

$$H(\mathbf{X}) = -\int p(\mathbf{X}) \log p(\mathbf{X}) \mathrm{d}\mathbf{X} \tag{2.17}$$

Conditional Entropy: The conditional entropy of a random variable $\mathbf{Y}$ given a realization of another random variable $\mathbf{X}$ measures the information about the random variable $\mathbf{Y}$ once we observe $\mathbf{X}$. If knowing $\mathbf{X}$ give us all the information about $\mathbf{Y}$ then the conditional entropy is minimum and takes a value of 0. It is defined as:

$$H(\mathbf{Y}|\mathbf{X}) = -\int p(\mathbf{Y}, \mathbf{X}) \log \frac{p(\mathbf{Y}, \mathbf{X})}{p(\mathbf{X})} \mathrm{d}\mathbf{X} \mathrm{d}\mathbf{Y} \tag{2.18}$$

Mutual Information: The mutual information measures the dependence between two random variables $\mathbf{X}, \mathbf{Y}$. In other words, it measures the amount of information obtained about one random variable when observing the other one. For example, if $\mathbf{X}, \mathbf{Y}$ are independent then the mutual information is zero. It can be decomposed as:

$$I(\mathbf{Y}, \mathbf{X}) = H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{X}) \tag{2.19}$$

2.2.2 Decomposing Proper Scoring Rules

We can now define the three quantities into which a PSR can be decomposed, see (Bröcker 2009):

Uncertainty: The uncertainty measures the inherent lack of knowledge around the label $\mathbf{Y}$, in other words, it is the entropy of the prior probability over the labels. Note that in the extreme case in which $p(\mathbf{Y} = c) = 1$, then our lack of knowledge is zero and the prediction done by a classifier will always be class c.

Resolution / Sharpness: The resolution measures how the probability assigned to a particular sample, given by $p(\mathbf{Y}|\mathbf{X}_n)$, differs from the average probability of the ground truth, i.e. from the prior probability $p(\mathbf{Y})$. In other words, if after observing $\mathbf{X}$ we don't get more information about our label $\mathbf{Y}$ beyond that provided by the prior probability, then our resolution will be zero. This gives us an interpretation of the resolution as the mutual information between $\mathbf{Y}$ and $\mathbf{X}$. The case in which the resolution is zero can be seen as a case in which knowing $\mathbf{X}$ does not provide further information about $\mathbf{Y}$, i.e. both random variables are independent and as a consequence do not share mutual information. An example of resolution being zero could be a case in which the prior probability of observation $p(\mathbf{Y} = 1) = 0.8$, and our model always predicts probability $p(\mathbf{Y}|\mathbf{X}) = 0.8$ for any input $\mathbf{X}$.

Calibration / Reliability: The calibration measures how the probability assigned by the model towards a class c, $p_c = p(\mathbf{Y}_n = c|\mathbf{X}_n)$, differs from the frequency of ground truth outcomes with assigned probability p_c. For example if given a set $\{\mathbf{X}_n\}_{n=1}^N$ the ground-truth outcome $\{\mathbf{Y}_n = 1\}_{n=1}^N$ is given with 0.8 frequency (i.e. out of $N = 10$ samples 8 belong to class 1) then a model is calibrated if $p(\mathbf{Y}_n = 1|\mathbf{X}_n) = 0.8$, $\forall n \in \{1, \ldots, N\}$.

From this decomposition, we can introduce the concept of refinement. The refinement is the difference between the uncertainty $H(\mathbf{Y})$ and the resolution $I(\mathbf{Y}, \mathbf{X})$, so it can be seen as the posterior entropy $H(\mathbf{Y}|\mathbf{X})$ about the label $\mathbf{Y}$, once the data $\mathbf{X}$ is observed. In other words, it measures how much uncertainty do we reduce about $\mathbf{Y}$ once $\mathbf{X}$ is observed. Thus, the refinement measures how separated the distributions are. For example, if the distributions being considered are totally overlapped (bad refinement), then observing a sample $\mathbf{X}$ will not provide any information about $\mathbf{Y}$ and hence our best guess is the prior probability about the labels $\mathbf{Y}$. In such a case, the refinement of our classifier cannot be better than the uncertainty because the mutual information between the variables is 0 as knowing $\mathbf{X}$ does not tell anything about $\mathbf{Y}$, due to the complete overlapping. On the other hand, if our distributions are totally

separated, then observing a sample $\mathbf{X}$ will provide us all the information about which class this sample belongs to, which means $H(\mathbf{Y}|\mathbf{X}) = 0$. In this case, all the uncertainty is reduced, i.e. the information gain $I(\mathbf{X}, \mathbf{Y})$ is maximal (equal to the entropy of the label $\mathbf{Y}$), which means that we gain all the information around our label $\mathbf{Y}$ once we observed $\mathbf{X}$, by reducing our prior uncertainty encoded in $H(\mathbf{Y})$. In this case, the refinement is the best possible. Finally, note that there would not be a way to completely define the label $\mathbf{Y}$ if there is some kind of overlap between the distributions, because in the regions where the distributions overlap, we would lose information (i.e. $H(\mathbf{Y}|\mathbf{X}) > 0$) and the refinement is affected. In this case, the mutual information could not reduce all the entropy over the labels. Another case in which the refinement is the best possible is when the entropy over the marginal distribution $p(\mathbf{Y})$ is zero, because in such a case only one class is presented and our lack of knowledge is zero ($H(\mathbf{Y}|\mathbf{X}) = 0$) even-though the mutual information is also 0.

2.2.3 *Illustrating* PSR *decomposition with an example*

The following visual example introduces the three concepts just discussed, in order to provide some light into the descriptions provided above. To do so we use a binary classification problem where $p(\mathbf{X}|\mathbf{Y} = c)$ is a Gaussian distribution. We also vary the prior over the classes $p(\mathbf{Y})$. In the case in which $p(\mathbf{Y} = 1) = 0$ then $H(\mathbf{Y}) = 0$ while $H(\mathbf{Y}) \approx 0.69$ when uncertainty around a label is maximal $p(\mathbf{Y} = 1) = 0.5$[9]. We have assumed a BDR in which a zero-one loss is used, which corresponds to the MAP decision threshold. In the Binary classification problem this corresponds to a threshold set by $p(\mathbf{Y} = 1|\mathbf{X}) = 0.5$. The entropy is computed analytically while the rest of the quantities, posterior entropy and posterior distribution, are computed numerically, using 1 million samples drawn from both distributions. We only display a subset of these samples in the figures. Note that the following example will hold for any number of arbitrary classes.

Resolution and Refinement

We start by providing some light into the concept of refinement and resolution. Figure 2.1 shows the toy example introduced above with different configurations, where one class is represented with red crosses and the other is represented with blue circles. On the top row the prior probability $p(\mathbf{Y} = 1) = 0.5$, while in the middle $p(\mathbf{Y} = 1) = 0.999$ and the bottom row is the case in which $p(\mathbf{Y} = 1) = 1$. From left to right we show distributions in which the class con-

[9] The values of entropies provided in the example are measured in nats.

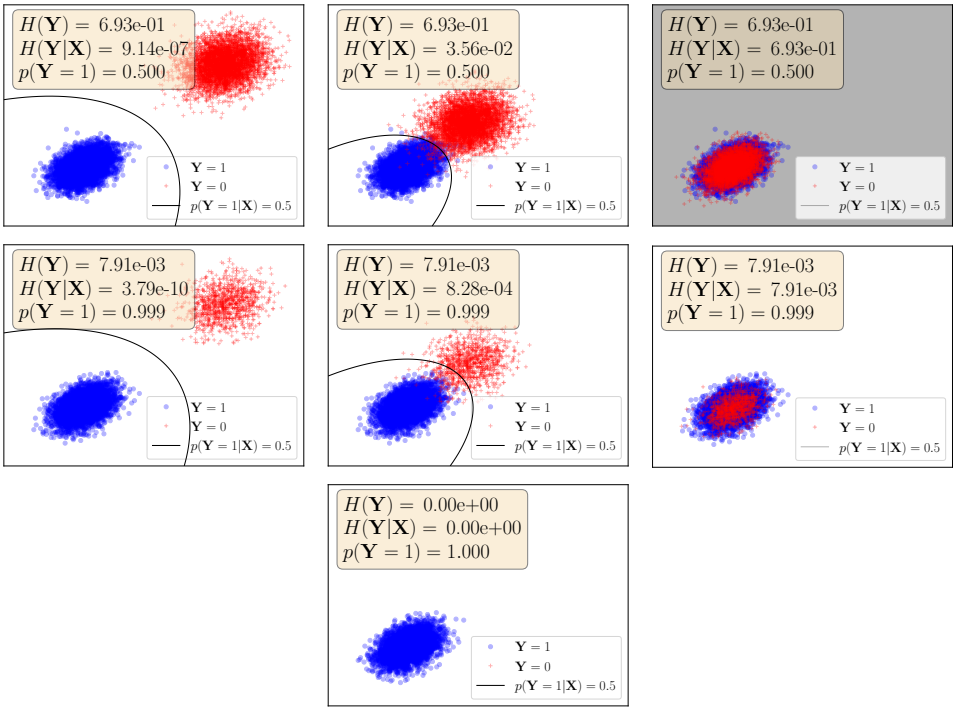


Figure 2.1: Toy example illustrating the concept of refinement and resolution for different prior probabilities and different data generating distribution $p(\mathbf{X}|\mathbf{Y})$. From left to right we present a figure showing good, bad and worst possible refinement.

ditional probability $p(\mathbf{X}|\mathbf{Y} = \text{red})$ has been varied in order to be more or less separated from the blue distribution. In black we show the decision threshold obtained by the optimal rule w.r.t. the zero-one loss function. In a box, we display the value of the entropies and conditional entropies. As will become clear later, it is important to note that the posterior probabilities used to compute these quantities has been obtained numerically using the data generating distributions $p(\mathbf{Y})$ and $p(\mathbf{X}|\mathbf{Y} = c)$.

First, note that the refinement measures how separated both distributions are. On each row, we can see that the refinement from the left plot is better than from the right, as measured by $H(\mathbf{Y}|\mathbf{X})$. This is because for a particular value of $\mathbf{X}$, we will be placed in one of the two distributions. When the distributions are very separated, there is no uncertainty around which label $\mathbf{Y}$ the feature $\mathbf{X}$ belongs to, i.e. we have removed all our lack of knowledge encoded in $H(\mathbf{Y})$,

while this uncertainty grows when the distributions start to overlap. When there is a total overlapping, the refinement is given by the uncertainty because in this case knowing $\mathbf{X}$ will not tell anything about $\mathbf{Y}$ beyond what the prior probability says and our refinement is the worst possible, given by $H(\mathbf{Y})$.

On the other hand, when the prior probability is more informative (middle row) the values of the posterior entropy $H(\mathbf{Y}|\mathbf{X})$ are lower than those of the top row comparing each of the columns. This is because the information that observing a particular value of $\mathbf{X}$ provides is the same for both rows, but because the prior in the middle row is more informative, there is an extra lowering of uncertainty coming from this knowledge and the overall refinement is better.

We can also see the connection between uncertainty, resolution and refinement from this example. The resolution measures how much information we gain about $\mathbf{Y}$ once $\mathbf{X}$ is observed. When the refinement is better, the resolution is also better because knowing $\mathbf{X}$ is more informative about $\mathbf{Y}$. When the distributions overlap the resolution of the classifier is 0 as we cannot do better than predict the prior probability, which is the reason why uncertainty and refinement are the same. When the distributions start to overlap (middle column), we lose resolution due to the lack of knowledge about which is the ground truth $\mathbf{Y}$ once $\mathbf{X}$ is observed.

The last case is the one given in the bottom row. Note that when the prior probability over one class is 1.0, then the refinement is perfect although the resolution is the worst possible. In this particular case knowing $\mathbf{X}$ does not provide information about $\mathbf{Y}$ because all the uncertainty in $\mathbf{Y}$ has already been reduced.

Finally, it is worth mentioning that a good or bad refinement implies that the error probability of our classifier is different. Even in the particular case in which we classify correctly all of our test data (100% accuracy) a better refinement implies a lower probability of error as can be seen by solving Equation 2.4. This is a very important fact when evaluating the performance of a classifier, as a finite test sample size can bias the conclusions we draw.

Calibration

Figure 2.2 illustrates the concept of calibration. To do so, we rely on the same example as above. In this case we pick the slice in which the posterior probability is within a confidence range, for example $0.9 \leq p(\mathbf{Y} = 1|\mathbf{X}) \leq 1.0$. Then we pick all the samples $\mathbf{X}$ that lie in this range and compute the frequency of those with $\mathbf{Y} = 1$. Note that we can say that a model is calibrated when $P(\mathbf{Y} = 1|t) = t$, $\forall t \in [0, 1]$, where remember t denotes the probability assigned by the classifier $t = p_\theta(\mathbf{Y}|\mathbf{X})$. Thus, a model is calibrated if this equality holds in expectation:

$$\mathbb{E}_{p(t)} \left[P(\mathbf{Y} = 1|t) - t \right] = 0 \tag{2.20}$$

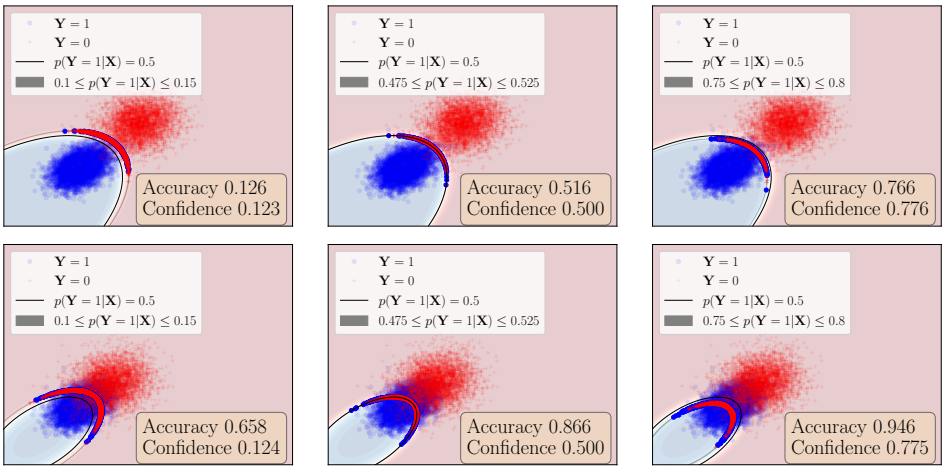


Figure 2.2: This figure illustrates the concept of calibration. From left to right, we evaluate the proportion of samples with $\mathbf{Y} = 1$ and the corresponding average confidence. We highlight the samples that belong to a posterior confidence region, marked in gray. The background color indicates the posterior probability assigned by the classifier. Top row illustrates a calibrated classifier while bottom row an uncalibrated one. We can see that when the model is perfectly calibrated (up to error introduced by finite sample estimation) for any t the proportion of samples with $Y = 1$ equals t.

As we can see, calibration is a way to measure how accurately do we model the uncertainty in our data distribution. When this uncertainty is correctly modeled, the calibration is perfect. This means that a model is calibrated when the probability assigned to the different regions in the input space $\mathbf{X}$ reflect the inherent overlapping of our data distributions. This is the same as

saying that a model is calibrated when the probability assigned by a model in a region reflects the proportion of samples of each class in that region. As will become clear in the next section, it is important to note that the confidences in the calibrated plot are computed using the posterior probability obtained by the data generating mechanism given by $p(\mathbf{Y})$ and a Gaussian distribution for $p(\mathbf{X}|\mathbf{Y})$ while the posterior confidences of the uncalibrated plots (bottom row) has been obtained by a non linear transformation applied to the posterior confidences: we have raised them to the power of 5 and then renormalize.

2.2.4 Calibration, Refinement and Bayes Decision Rule

We started section 2.2 by noting that the goal of a practitioner is to set a model $p_\theta(\mathbf{Y}|\mathbf{X})$ that should recover the data generating distribution $p(\mathbf{Y}|\mathbf{X})$. We have also seen that the refinement and the calibration are properties that characterize $p(\mathbf{Y}|\mathbf{X})$. Given some data, the best possible refinement we can have is given by the overlapping between the different classes, while the calibration error can always be reduced to zero as this just implies assigning probabilities that reflect this degree of overlapping. As highlighted previously, the refinement computation and the calibrated picture in the toy example was done using a numerical approximation to the data generating posterior distribution i.e. $p_\theta(\mathbf{Y}|\mathbf{X}) = p(\mathbf{Y}|\mathbf{X})$ while the uncalibrated one was computed using $p_\theta(\mathbf{Y}|\mathbf{X}) \neq p(\mathbf{Y}|\mathbf{X})$. This shows that when we use the PSR as our training objective, in its maximum $S(q(\mathbf{Y}), q(\mathbf{Y}))$, i.e. when $p_\theta(\mathbf{Y}|\mathbf{X}) = p(\mathbf{Y}|\mathbf{X})$ the calibration will be reduced to zero while the refinement will be intrinsic to $p(\mathbf{Y}|\mathbf{X})$.

Our goal is then, to set a model $p_\theta(\mathbf{Y}|\mathbf{X})$ that can separate the classes, i.e. it can recover the refinement, and that is able to assign the probabilities depending on the degree of uncertainty between our classes, i.e. it is able to have zero calibration error. When this is achieved, we know that our model will have recovered the data generating distribution and thus taking decisions using BDR will provide optimal performance. Figure 2.3 shows the decision thresholds, given a zero-one loss, obtained with a model where the calibration has been affected.

It is clear then that a model that recovers the refinement but has a bad calibration, will not provide the optimal model in order to make decisions. This means that by calibrating a model we can improve the error probability and thus the accuracy of our classifier. A nice example of this effect is provided by (Cohen et al. 2004).

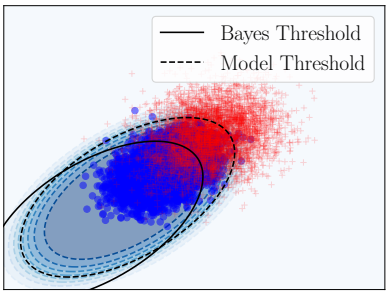


Figure 2.3: Comparison of decision rule obtained with the data generating distribution and a model that is uncalibrated. The uncalibrated model has been obtained by taking the data generating distribution and shifting the probabilities. Hence the refinement remains unaltered.

This contrasts with the common motivation in the recent literature of the necessity of having a calibrated classifier. Due to the overconfidence in DNN (Zhang et al. 2017) which has been shown to affect the calibration of these models (Guo et al. 2017), practitioners usually motivate the need of having a calibrated classifier in order to provide reliable predictions that reflect the proportion of outcomes, an attempt to reduce this overconfidence. However, calibrating a model implies both: having reliable predictions but also improving the accuracy of your classifier, since it implies approaching the data generating distribution and thus obtaining minimum error probability. *In other words, calibration implies optimality.*

Finally, it is important to note that the refinement and the calibration are decoupled properties, as having a good/bad calibration does not imply having a good/bad refinement. For example, a model with perfect calibration and worst refinement is a prior classifier, i.e. when the distributions totally overlap, while a distribution that has perfect refinement and worst possible calibration could be a model where the classes are totally separated but the probabilities assigned by the model have been rotated. It shall be noted that both properties must be recovered by our model, with the difference that the refinement can only be the best possible if the distributions are separated in origin, while the calibration error can always be reduced to 0. This point is the central observation of the contribution in chapter 3.

2.2.5 Measuring Calibration

Once the concepts of calibration and refinement have been introduced, we now present different ways of measuring both properties.

As we can deduce from this chapter, we can measure the calibration and the refinement by using a PSR. To this end in this thesis we use both the Logarithmic Score (LS) and the Brier Score (BS) given by Equation 2.15.

These PSR measure calibration and refinement in different ways. For example, the LS heavily penalizes strong errors. Note that just *one* totally incorrect sample (assign class c' with probability 1 when the true label is c) will make the LS saturate to $-\infty$, no matter if the rest of the samples are perfectly classified with confidence 1.0 (i.e. perfect calibration and refinement). This means that the LS is suitable for tasks in which making a very wrong prediction is highly risky. On the other hand, this effect is not present in the BS, as this scoring rule is bounded between 0 and 1.

In addition to these PSR we need a metric that only measures the calibration error. Note that, knowing that a PSR can be decomposed into calibration and refinement does not mean we can actually compute both quantities separately. For some cases, like the BS this is possible, but not for other PSR. For this reason, in this thesis we also use two ways of measuring calibration that have been proposed in the literature. These correspond to the Expected Calibration Error (ECE) and the Maximum Calibration Error (MCE) (Naeini et al. 2015).

The ECE is derived by discretizing Equation 2.20. To do so the expectation over $p(t)$ is approximated by partitioning the confidence range into M bins B_m each one having $|B_m|$ samples with confidence lying in that bin. On each of the bins, the average confidence $\text{AVGCONF}(B_m)$ approximates t while the accuracy $\text{ACC}(B_m)$ approximates $p(\mathbf{Y} = c|t)$. Then, a weighted sum of each of these bins is taken. Thus for a total of N samples:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{N} \left|\text{AVGCONF}(B_m) - \text{ACC}(B_m)\right| \tag{2.21}$$

On the other hand, in high risk applications rather than taking the expectation as in Equation 2.20, we might want the maximum of such difference. The MCE approximates this quantity as follows:

$$\text{MCE} = \max_{1 \leq m \leq M} \left|\text{AVGCONF}(B_m) - \text{ACC}(B_m)\right| \tag{2.22}$$

It shall be noted that these calibration metrics are not perfect but they have been widely adopted in the literature. For a discussion on its caveats and possible solutions see (Nixon et al. 2019).

Finally, note that in the four calibration metrics, the closer the value to 0, the better-calibrated model.

2.3 How does learning techniques meet model calibration and refinement

We end this chapter by showing the connection between PSR, refinement and calibration, with common techniques used to learn model parameters. We then briefly introduce the caveats of these techniques, motivating the use of uncertainty quantification around the parameters of the model.

2.3.1 Maximum Likelihood Estimation

One traditional way of learning the parameters of a model is by maximizing the log-likelihood function, a technique known as Maximum Likelihood Estimation (MLE). By specifying the probability distribution that $\mathbf{Y}$ follows given $\mathbf{X}$, for example a Student-t likelihood if $\mathbf{Y}$ is assumed to be some point corrupted with heavy-tailed noise, the log-likelihood function is given by:

$$\log \prod_{n=1}^{N} p_\theta(\mathbf{Y}_n|\mathbf{X}_n) \tag{2.23}$$

where $\mathcal{D} = \{\mathbf{Y}_n, \mathbf{X}_n\}_{n=1}^N$ is assumed to be drawn i.i.d. from the model distribution, allowing the factorization above. The use of the log function serves for several purposes such as allowing unbiased gradient estimates through mini batching, well-behaved gradients, numerical stability etc.

When the likelihood function is the categorical distribution, it can be easily shown that the log-likelihood function is proportional to the Logarithmic Scoring rule, while when $p_\theta(\mathbf{Y}|\mathbf{X})$ is a multivariate factorized Gaussian distribution, with the mean bounded between 0 and 1 (using a sigmoid link function for example), the log-likelihood function is proportional to the Brier Scoring

Rule (both defined in Equation 2.15), see (Bishop 1995)[10]. This can be easily shown as follows. Let's assume $p_\theta(\mathbf{Y}|\mathbf{X})$ is given by a categorical likelihood where the parameters μ of the categorical likelihood are given by some model (the output of a softmax function applied to the logit computed with a DNN). The log-likelihood function is given by:

$$\begin{aligned}
&\log \prod_{n=1}^{N} p_\theta(\mathbf{Y}_n|\mathbf{X}_n) \\
&\log \prod_{n=1}^{N}\prod_{c=1}^{C} \mu_c(\mathbf{X}_n,\theta)^{\mathbb{1}_{[\mathbf{Y}_n=c]}} = \\
&\sum_{n=1}^{N}\sum_{c=1}^{C} \mathbb{1}_{[\mathbf{Y}_n=c]} \log \mu_c(\mathbf{X}_n,\theta) = \\
&\sum_{n=1}^{N} \log \mu_{\mathbf{Y}_n}(\mathbf{X}_n,\theta) \propto \text{LS}
\end{aligned} \tag{2.24}$$

showing the equivalence with the LS, since in Equation 2.15 $p_\theta(\mathbf{Y}|\mathbf{X})$ referred directly to a vector of probabilities. Note that since the LS is scaled by a constant $1/N$, then it does not affect the result of the optimization so optimizing the LS is equivalent to maximizing the log likelihood function assuming a categorical likelihood. In a similar fashion, if $p_\theta(\mathbf{Y}|\mathbf{X})$ is given by a multivariate Gaussian distribution[11] with the mean vector μ given by the output of a model which is then bounded between 0 and 1 using for example the sigmoid function applied independently over each logit, and a constant diagonal covariance Σ , then the log function is given by:

$$\begin{aligned}
&\log \prod_{n=1}^{N} p_\theta(\mathbf{Y}_n|\mathbf{X}_n) \\
&\log \prod_{n=1}^{N}\prod_{c=1}^{C} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\mathbb{1}_{[c=\mathbf{Y}_n]} - \mu_c(\mathbf{X},\theta))^2}{2\sigma^2}\right) =
\end{aligned}$$

[10] It shall be noted that in Equation 2.15, $p_\theta(\mathbf{Y}|\mathbf{X})$ refer to a general probability bounded between 0 and 1, without assuming any density function, which didn't need to be defined to write the PSR. In this paragraph, however, $p_\theta(\mathbf{Y}|\mathbf{X})$ refer to a specific density form.

[11] Clearly, this noise model does not make sense for discrete/classification problems.

$$\underbrace{\sum_{n=1}^{N}\sum_{c=1}^{C}\log\frac{1}{\sqrt{2\pi}\sigma}}_{\text{Constant w.r.t. }\theta}+\sum_{n=1}^{N}\sum_{c=1}^{C}\left(-\frac{(\mathbb{1}_{[c=\mathbf{Y}_n]}-\mu_c(\mathbf{X},\theta))^2}{2\sigma^2}\right)=$$
$$-\sum_{n=1}^{N}\sum_{c=1}^{C}\left(\mathbb{1}_{[c=\mathbf{Y}_n]}-\mu_c(\mathbf{X},\theta\right)^2\propto -\text{BS}$$

where we have remove constants that does not affect the optimization beyond scaling the gradients, something that is absorbed into the learning rate of the optimizer. Note that maximizing this log-likelihood function is equivalent to minimizing the BS.

Hence, in many common situations, maximizing the log-likelihood function will optimize a PSR, providing calibrated outputs with a proper refinement. From an intuitive point of view that is to be expected: if we assume that our data follows a categorical distribution, then maximizing the log probability will be achieved when the samples are assigned with confidence 1.0 to their correct class. It is clear that if possible, the model will have a perfect refinement (all the samples are correctly separated) with perfect calibration (all the samples are classified with 1.0 confidence).

2.3.2 Empirical Risk Minimization

Another view point of this learning procedure is through the lens of empirical risk minimization (ERM). Remember from the beginning of this section that our goal is to minimize the error probability w.r.t. some loss function:

$$p(error)=\int\mathcal{L}(\mathbf{X},\mathbf{Y},\alpha)\mathrm{d}P(\mathbf{X},\mathbf{Y})\ , \tag{2.25}$$

Then it seems reasonable to learn a model that directly minimizes this error probability. In other words, the model learns to take actions α based on the target distribution and the loss associated with the predictions. However, since we only have access to a sample which we assume belongs to $P(\mathbf{X},\mathbf{Y})$, the above integral can be approximated with the empirical loss:

$$\frac{1}{N}\sum_{i=n}^{N}\mathcal{L}(\mathbf{X},\mathbf{Y},\alpha,\theta)\delta(\mathbf{X}-\mathbf{X}_n)\delta(\mathbf{Y}-\mathbf{Y}_n). \tag{2.26}$$

Empirical Risk Minimization is the name given to the learning procedure that minimizes the empirical loss. If, for example, the loss function is the squared loss: $(\alpha(\mathbf{X}_n, \theta) - \mathbf{Y}_n)^2$ then minimizing this objective is equivalent to maximizing the log Likelihood when we assume a factorized Gaussian observation model $p_\theta(\mathbf{Y}|\mathbf{X})$ with constant variance, hence equivalent to minimizing the Brier Scoring Rule if the mean is bounded between 0 and 1, as shown in the previous subsection. This states a clear connection between ERM and MLE for different loss functions and likelihood models (up to a $1/N$ factor).

Note that, in general, if we choose the loss function to be a PSR then ERM will provide a model where both the refinement and the calibration are optimized since optimum will occur at $p_\theta(\mathbf{Y}|\mathbf{X}) = p(\mathbf{Y}|\mathbf{X})$.

2.3.3 Divergence Minimization and Overfitting

When using an overparameterized model, such as a DNN, both MLE and ERM can suffer from overfitting, unless: 1) the model parameterized by θ is well-specified (i.e. it can parameterize the data generating distribution) and 2) we have access to an infinite amount of samples from $P(\mathbf{X}, \mathbf{Y})$[12].

My favorite way of explaining the source of this overfitting is by noting that both ERM and MLE are a surrogate of minimizing the Kullback–Leibler divergence (KLD) between the data generating distribution and the model distribution:

$$\begin{aligned} &\int p(\mathbf{X}, \mathbf{Y}) \log \frac{p(\mathbf{X}, \mathbf{Y})}{p_\theta(\mathbf{X}, \mathbf{Y})} \mathrm{d}\mathbf{X}\mathrm{d}\mathbf{Y} = \\ &\underbrace{\int p(\mathbf{X}, \mathbf{Y}) \log p(\mathbf{X}, \mathbf{Y}) \mathrm{d}\mathbf{X}\mathrm{d}\mathbf{Y}}_{\text{Entropy}} - \underbrace{\int p(\mathbf{X}, \mathbf{Y}) \log p_\theta(\mathbf{X}, \mathbf{Y}) \mathrm{d}\mathbf{X}\mathrm{d}\mathbf{Y}}_{\text{Cross-entropy}} \end{aligned} \tag{2.27}$$

We can then minimize this divergence w.r.t. θ, and since the entropy is constant w.r.t. θ and by approximating the expectation of the cross-entropy with the empirical distribution we arrive at the empirical loss or the MLE criteria, showing this equivalence.

[12] We obviously need, for example, a well-behaved learning algorithm, but these are the minimum requirements to recover the data generating distribution.

$$\begin{aligned}&\operatorname*{argmin}_{\theta} \int p(\mathbf{X},\mathbf{Y}) \log p(\mathbf{X},\mathbf{Y}) \mathrm{d}\mathbf{X}\mathrm{d}\mathbf{Y} - \int p(\mathbf{X},\mathbf{Y}) \log p_\theta(\mathbf{X},\mathbf{Y}) \approx \\ &\operatorname*{argmin}_{\theta} -\frac{1}{N}\sum_{n=1}^{N} \delta(\mathbf{X}-\mathbf{X}_n)\delta(\mathbf{Y}-\mathbf{Y}_n) \log p_\theta(\mathbf{Y}|\mathbf{X})\end{aligned} \tag{2.28}$$

On the other hand, instead of directly sampling from $p(\mathbf{Y},\mathbf{X})$ to approximate the empirical loss, we might be given for each sample $\mathbf{X}$ the probability $p(\mathbf{Y}|\mathbf{X})$ this $\mathbf{X}$ belongs to. For example $\mathbf{X}$ might represent a tweet and $p(\mathbf{Y}|\mathbf{X})$ is a vector saying this tweet has 0.8 probability of being misogynist and 0.2 of being racist. In this case the empirical loss is approximated by:

$$\begin{aligned}&\operatorname*{argmin}_{\theta} \int p(\mathbf{X},\mathbf{Y}) \log p(\mathbf{X},\mathbf{Y}) \mathrm{d}\mathbf{X}\mathrm{d}\mathbf{Y} - \int p(\mathbf{X},\mathbf{Y}) \log p_\theta(\mathbf{X},\mathbf{Y}) \approx \\ &\operatorname*{argmin}_{\theta} -\frac{1}{N}\sum_{n=1}^{N} \delta(\mathbf{X}-\mathbf{X}_n) \sum_{c=1}^{C} p(\mathbf{Y}=c|\mathbf{X}) \log p_\theta(\mathbf{Y}|\mathbf{X})\end{aligned} \tag{2.29}$$

If $p_\theta(\mathbf{Y}|\mathbf{X})$ is a categorical likelihood an with a similar derivation as that in Equation 2.24, we show this is equivalent to the LS when a vector of probabilities is given, as in Equation 2.16.

It is well known that this form of KLD force the model $p_\theta(\mathbf{X},\mathbf{Y})$ to cover all the target distribution, see (Bishop 2006, p. 469), which includes outliers or any unrepresentative source of variation from the objective $p(\mathbf{Y}|\mathbf{X})$. Other ways of explaining overfitting include, e.g. the Vapnik–Chervonenkis dimension (Vapnik et al. 1971). Obviously one can use a robust divergence to outliers, such as the β divergence, so that the model is not forced to cover the tails of the distribution. With this, we will arrive at a different training criteria.

To finish, we noted in a previous subsection that PSR are Bregman Divergences. From the KLD divergence we can derive the LS, while with the euclidean distance we arrive at the BS. It is important to note that we have arrived at both the LS and the BS through a different path by the maximum likelihood training criteria (equivalent to KLD minimization) by assuming two different observation models. As we noted, the difference between the Scoring rule and the maximum likelihood paths is the model θ being a probability vector or a probability density. Thus, in both cases, we can use alternatives to the KLD to create robust scoring rules, which again are derived or can be justified via different paths. What is important is that since all these different training

criteria are the same, all the ideas from scoring rules can be extrapolated into maximum likelihood and vice-versa. In other words, the refinement and calibration are things being optimized when doing maximum log likelihood, and so any reasoning around these two concepts extrapolates.

2.3.4 Regularization

One way to avoid overfitting and improve generalization is by employing a set of techniques known as regularization. This set of techniques can be divided into groups. For example parameter sharing, Convolutional Neural Networks (and more in general: inductive bias), max-norm etc. are techniques that restrict the set of possible functions that the model can parameterize. On the other hand, different stochastic optimization techniques can also improve generalization by performing a better search over the parameter space. This includes gradient clipping, learning rate scheduling, early stopping etc. Another way to improve generalization is by employing data augmentation, which aims at extending the set of samples used for training. This can be done by employing transformations that are likely to produce samples representative of $p(\mathbf{X}, \mathbf{Y})$, transformations that include samples in the vicinal of the empirical distribution (Chapelle et al. 2001) or transformations that include directions in the data space in which the model drastically change its predictions (Szegedy et al. 2014).

One of the most popular techniques to improve generalization relies on modifying the objective function by adding a regularizer. Norm penalties (such as the L_1 or L_2) over the parameters are quite popular choices. Depending on the norm we can induce a different effect. For example, the L_1 norm tends to shrink the parameters of the model towards zero, and thus have been popularized as a technique for model pruning, although is not a very good one and hierarchical priors have shown an improvement upon them (Louizos et al. 2017a). These norm penalties have a direct connection with Bayesian inference, perhaps one of the (in theory) greatest and best founded ways of regularizing our model.

2.3.5 Bayesian Learning

We have seen so far that our goal is to approximate a target distribution $p(\mathbf{X}, \mathbf{Y})$ with a model $p_\theta(\mathbf{X}, \mathbf{Y})$. However, the traditional learning techniques introduced above will provide a single optimal $\hat{\theta}$ which represents all our knowledge about the target distribution. In theory, the information contained in $\hat{\theta}$ will be enough under an ideal setting: 1) when the model is not misspecified and 2) when we have an infinite amount of data, which is not true in practice.

Then since we won't be able to characterize all our knowledge using a single $\hat{\theta}$, it seems reasonable to model the degree of uncertainty around the parameter θ, given the data. Bayes theorem provides all the theory we need to represent this degree of uncertainty. For that, we can introduce a prior distribution $p(\theta|\gamma)$ over the parameters θ, where γ parameterizes the prior. The purpose of this prior is to encode our prior beliefs about the parameters. If for example θ represents the mean of a population and we know that mean is within some range, the prior distribution can restrict the set of values that θ can have.

By combining the prior and the likelihood, we can compute the posterior distribution, given by:

$$p(\theta|\mathbf{Y}, \mathbf{X}, \gamma) = \frac{p_\theta(\mathbf{Y}|\mathbf{X})p(\theta|\gamma)}{\int p_\theta(\mathbf{Y}|\mathbf{X})p(\theta|\gamma)\mathrm{d}\theta}, \tag{2.30}$$

which will represent the degree of uncertainty around the parameter, given the data. Note that the combination of the likelihood and the prior leads to a proper quantification of uncertainty. For ease of understanding, suppose that our lack of knowledge about the parameter θ is maximal, i.e. we choose a uniform distribution over the parameter[13]. Then it is clear that the posterior will concentrate more density on parameters that agree more with the data, i.e. that explains better the data.

On the other hand, we can interpret the effect of the prior as being a regularizer. Thus, we could optimize this objective w.r.t. θ, a technique known as Maximum a Posterior optimization, which is equivalent to L_2 regularization for the choice of a (centered) Gaussian prior, and L_1 regularization for the choice of a (centered) Laplace prior.

[13] We won't discuss the particularities of improper priors and assume we always have a proper posterior.

Furthermore, we can extend this formulation, by additionally placing a prior over γ, leading to a hierarchical latent variable model. This allows us to encode uncertainty on the prior as well. For example, if we know that our prior should be a Gaussian distribution but we are not sure about the parameters that this Gaussian should have, then we can impose a hyperprior distribution on the parameters of this prior (that can be made uninformative). We can keep increasing the hierarchy as desired.

Bayesian Predictions

In this subsection we are not interested in how we can compute this posterior distribution (see chapter 4), but what are the appealing properties of using Bayesian Learning. The first one is that it allows us to make predictions taking into consideration the uncertainty around the set of possible parameters (also known as epistemic uncertainty), encoded through the posterior distribution.

This is a direct consequence of applying the product rule and the marginalization rule from probability theory. We start with the posterior predictive distribution $p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{X}, \mathbf{Y})$, which encodes the degree of uncertainty around a prediction $\mathbf{Y}^*$ given a test input $\mathbf{X}^*$, and the observed data $\mathbf{X}, \mathbf{Y}$[14]. We then incorporate the parameters as a random variable and apply the product rule. This is summarized in the following expression:

$$\begin{aligned} p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{Y}^*, \theta|\mathbf{X}^*, \mathbf{X}, \mathbf{Y})\mathrm{d}\theta = \\ \int p(\mathbf{Y}^*|\mathbf{X}^*, \theta)p(\theta|\mathbf{X}, \mathbf{Y})\mathrm{d}\theta. \end{aligned} \tag{2.31}$$

In this way, the prediction is based on the entire parameter space, where each of the conditional probabilities $p(\mathbf{Y}^*|\mathbf{X}^*, \theta)$ are weighted according to the posterior distribution. The parameters that explain the data better have a greater influence on the final prediction because the posterior distribution will assign high density to them.

This way of making predictions have several advantages. First note that because all the parameters (each one representing a model) contribute to the prediction, the posterior predictive distribution is more robust to overfitting and more representative of the data distribution than a point estimate selection

[14]Note that we are considering a class conditional setting but similar holds for unconditional modeling.

of θ. Moreover, the model will make more reliable predictions in underrepresented parts of the input space (where no data is available for learning), because the posterior distribution will assign low density to parameters that explain these regions in different ways, hence canceling the contributions between each of them ending up with a high entropy class conditional probability.

Regarding this last paragraph, note that models that are likely to represent a region of the input space where only samples from a particular class are present will end up assigning high confidence to that particular class in that region, because increasing the density towards other classes will not raise the likelihood from the numerator in Equation 2.30. On the other hand, models that are likely to explain regions where features from two or more classes overlap will be forced to increase the probability density of both classes, thus relaxing the ultimate confidence provided to those classes in that region of the input space[15]. This behavior will favor probabilities that closely reflect the patterns shown in the data, i.e. that are more representative of the data distribution. As we already know from the discussion previous to this subsection, this better representation leads to a better-calibrated model. This is the central point of the contribution in chapter 4.

Bayesian Model Selection

The second appealing property of Bayesian learning is that it gives a principled way for Model selection, which I prefer over the typical cross validation. Let's denote a model with M_1 (for example an 18-layer Residual Network) and with M_2 a different model (a fully connected Neural Network). We could use the posterior probability to choose which model to use, i.e. model the data with M_1 if $p(M_1|\mathbf{X}, \mathbf{Y}) > p(M_2|\mathbf{X}, \mathbf{Y})$. Assuming that we impose the same prior over choosing any of the models $p(M_1) = p(M_2) = 1/2$. Then the model chosen will be that in which $p(\mathbf{X}, \mathbf{Y}|M_x)$ is greater.

Note that this probability is the denominator in Equation 2.30, which is known as the marginal likelihood. As with any probability distribution, the marginal likelihood must integrate to 1. This means that highly parameterized models will have to spread their probability between the different datasets it can represent; and only when the given dataset is really well represented by this particular model (for example images and Convolutional Neural Networks), then the marginal likelihood of an expressive model will be greater than a simpler one. In other words, using the marginal likelihood for model selection

[15]Note that this total confidence must sum up to 1.

implies selecting a model using a trade off between fitting the data and model expressiveness, favoring simpler models. This implies a form of automatic Occam's Razor (MacKay 1992).

2.3.6 Model misspecification

So far in this chapter, we have assumed that either we have access to the data generating distribution $p(\mathbf{Y}|\mathbf{X})$ or that our model $p_\theta(\mathbf{Y}|\mathbf{X})$ is well specified in the sense that under some conditions previously outlined, the model can recover the data generating distribution.

In practice, however, this is far from being true, and the model is usually misspecified. This misspecification can happen at many levels. For example, the likelihood might be misspecified w.r.t. the data generating distribution: in practice, classifiers are usually trained by assuming that the observation model follows a categorical (multiclass) or a Bernoulli (binary) likelihoods, which impose independence between the classes, which is far from being true. On the other hand, beyond regularization, one can derive training criteria from robust divergences. This can make the model robust to outliers (Jewson et al. 2018), which is something a categorical likelihood in the absence of infinity data and/or being a misspecified model will suffer from.

In terms of Bayesian inference, model misspecification clearly affects the performance one can obtain when making predictions using the posterior predictive. In fact, the Bayesian arguments that make Bayesian inference the (probably best) tool to make predictions are only justified if the model is well specified (Walker 2013; Knoblauch et al. 2019). In other words, inferences using uncertainty quantification about a parameter θ are only justified if the prior and the likelihood are well specified, as we will discuss further in this thesis.

It turns out that modern machine learning techniques are usually misspecified, since they are complex black-box function approximators difficult to interpret i.e. it is difficult to understand which types of functions a given black-box approximator parameterizes. Hence, this lack of interpretability implies that specifying meaningful prior/likelihoods for our application remains a challenging task. On the other hand, high dimensional distributions are hard to analyze, which complicates the understanding of what kind of function parameterizes the distribution that represents this data.

Many alternatives have been proposed to overcome these limitations. On the side of misspecified priors, one of my favorites is accounting for uncertainty in a different way by changing the uncertainty quantifier, which is given by the KLD

in standard Bayesian inference (Knoblauch et al. 2019). This will be discussed later in this thesis. On the side of likelihood misspecification perhaps the work from (Bissiri et al. 2016) is one of the most prominent approaches. (Bissiri et al. 2016) shows that one can properly account for parameter uncertainty by defining loss functions that target our ultimate source of interest when making predictions. For example, if we are concerned with making predictions about the median, then rather than modeling the whole distribution (which can be very difficult), (Bissiri et al. 2016) shows that we can obtain proper uncertainty quantification in form of posterior distributions by just conducting inferences about the median, i.e. by defining a likelihood and a prior on this quantity of interest.

In any case, even in the misspecified setting being Bayesian usually provides better predictions than using a point estimate of the parameters. Figure 4.2 compares the predictions of a Bayesian Neural Network where samples from the posterior have been drawn using Hamiltonian Monte Carlo, vs the predictions performed by a Neural Network where a single parameter $\hat{\theta}$ is obtained by optimizing this posterior distribution. As discussed in subsection 4.2.1 we can see how the probabilities of the Bayesian Neural Network better reflect the data distribution, something not present in the point-estimate. In this problem the Neural Network has a great chance of being misspecified w.r.t. the data generating distribution since this data has been obtained using the same procedure as the examples in this chapter and the observation model of this Neural Network is categorical[16].

[16]Remember samples from this toy problem are drawn by assuming that $p(\mathbf{X}|\mathbf{Y})$ is a Gaussian distribution and $p(\mathbf{Y})$ is a Categorical distribution. With these choices, we cannot compute the posterior $p(\mathbf{Y}|\mathbf{X})$ analytically and hence we cannot assume is Categorical (the prior is not conjugate of the likelihood), which is what I have assumed to train/perform-inference over the Neural Network.

Chapter 3

Implicit Calibration of Deep Neural Networks using Mixup Training

The third chapter of this PhD thesis is focused on the discussion around model calibration and data augmentation. We first build upon the introduction about ERM *and formally introduce Vicinal risk minimization and Mixup. Then we use the ideas of chapter 2 to justify why Mixup, and any Data augmentation technique in general, will not provide calibrated distributions. We use these ideas to propose our solution which is finally evaluated. All the content in this chapter corresponds to (Maroñas et al. 2021a).*

3.1 From Empirical to Vicinal Risk minimization

In chapter 2 we have introduced Empirical Risk Minimization (ERM) as a learning technique of model parameters θ that minimize an empirical estimate of the error probability w.r.t. a loss function, which we know is the target of Bayes Decision Rule:

$$p(error) = \int \mathcal{L}(\mathbf{X}, \mathbf{Y}, \alpha, \theta) \mathrm{d}P(\mathbf{X}, \mathbf{Y}) \approx \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{X}, \mathbf{Y}, \alpha, \theta) \delta(\mathbf{X} - \mathbf{X}_n) \delta(\mathbf{Y} - \mathbf{Y}_n). \tag{3.1}$$

One of the consequences of ERM is that it clearly lacks of support in many different parts of the data distribution $p(\mathbf{Y}, \mathbf{X})$, which makes this learning paradigm present some limitations such as overfitting, memorization (Zhang et al. 2017), bad calibration (Guo et al. 2017), or sensitivity to adversarial examples (Szegedy et al. 2014).

In fact this is to be expected. Remember that optimizing a PSR aims at setting $p_\theta(\mathbf{Y}|\mathbf{X}) = q(\mathbf{Y})$. So if $q(\mathbf{Y})$ is a empirical distribution given by Dirac measures, an overparameterized model $p_\theta(\mathbf{Y}|\mathbf{X})$ can learn this empirical distribution so that $p_\theta(\mathbf{Y}|\mathbf{X}) = q(\mathbf{Y})$ which in turns implies that it learns to perfectly separate (perfect refinement) and assign extreme 1.0 confidences (perfect calibration).

Vicinal Risk Minimization (VRM) (Chapelle et al. 2001) is proposed to solve this lack of support in the input manifold. The idea is that the Dirac Delta empirical distribution $1/N \sum_{n=1}^{N} \delta(\mathbf{X} - \mathbf{X}_n)\delta(\mathbf{Y} - \mathbf{Y}_n)$ is replaced by a vicinity distribution, which aims at exploring different parts of the input space in the vicinity of the observed samples $\mathbf{X}_n, \mathbf{Y}_n$. An example of a vicinity distribution could be a Gaussian centered at each sample $\mathbf{X}_n$. In practice, we then sample from this Gaussian distribution and recover an unbiased estimate of the empirical loss.

$$\frac{1}{N} \sum_{n=1}^{N} \int \mathcal{L}(\mathbf{X}, \mathbf{Y}_n, \alpha_n, \theta) \mathcal{N}(\mathbf{X}|\mathbf{X}_n, \sigma^2) \delta(\mathbf{Y} - \mathbf{Y}_n) \mathrm{d}\mathbf{X} \approx \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{X}_{nj}, \mathbf{Y}, \alpha_n, \theta) \delta(\mathbf{Y} - \mathbf{Y}_n), \mathbf{X}_{nj} \sim \mathcal{N}(\mathbf{X}|\mathbf{X}_n, \sigma^2). \tag{3.2}$$

Thus, note that any Data Augmentation (DA) technique can be understood through the lens of Vicinal Risk Minimization.

3.1.1 Mixup Training

Many common DA techniques assume that the samples in the vicinity distribution belong to the same class. The idea behind Mixup (Zhang et al. 2018) relies on defining a vicinity distribution over the inputs $\mathbf{X}$ and over the labels $\mathbf{Y}$. This vicinity distribution is defined as the expected value of a linear interpolation between two input samples and their corresponding labels, where the parameter of the linear interpolation γ follows a beta distribution parameterized by ν:

$$\begin{aligned} &p(\mathbf{X}, \mathbf{Y} | \mathbf{X}_n, \mathbf{Y}_n) = \\ &\frac{1}{J} \sum_{j=1}^{J} \mathbb{E}_\gamma[\delta(\mathbf{X} - [\gamma \cdot \mathbf{X}_n + (1-\gamma) \cdot \mathbf{X}_j])\delta(\mathbf{Y} - [\gamma \cdot \mathbf{Y}_n + (1-\gamma) \cdot \mathbf{Y}_j])] \end{aligned} \tag{3.3}$$

The empirical loss is now evaluated by drawing samples from this vicinity distribution as follows:

$$\begin{aligned} &\mathbf{X}_n, \mathbf{Y}_n, \mathbf{X}_j, \mathbf{Y}_j \sim p(\mathbf{X}, \mathbf{Y}) \\ &\gamma \sim \text{Beta}(\nu, \nu) \\ &\widetilde{\mathbf{X}} = \gamma \cdot \mathbf{X}_n + (1-\gamma) \cdot \mathbf{X}_j \\ &\widetilde{\mathbf{Y}} = \gamma \cdot \mathbf{Y}_n + (1-\gamma) \cdot \mathbf{Y}_j \\ &\mathcal{L}(\widetilde{\mathbf{X}}, \widetilde{\mathbf{Y}}, \alpha, \theta) \end{aligned} \tag{3.4}$$

As a consequence, training with Mixup smooths the predictions performed by a model in the intersection between samples from the unknown distribution $P(\mathbf{X}, \mathbf{Y})$. This vicinity distribution showed a great improvement of the accuracy of different models (Zhang et al. 2018), and has been established as a common regularization technique, from which many different extensions have been proposed e.g. (Yun et al. 2019; Verma et al. 2019; Hendrycks et al. 2020).

3.2 Does Mixup Training really achieves Model Calibration?

The fundamentals and good performance of Mixup training has motivated the community towards exploring if this technique has other desirable properties beyond improving the accuracy of a DNN. The work from (Thulasidasan et al. 2019) studies the calibration and predictive uncertainty of Mixup against

other well-established calibration techniques. Beyond other properties, this work shows that Mixup improves the calibration of DNN.

In this chapter, however, we discuss and provide empirical evidence that DNN trained with Mixup do not necessarily improve calibration. To do so we rely on the ideas presented in chapter 2, using them to propose a new loss function that improves the calibration of DNN trained with Mixup.

The main takeaway from this chapter is not the uncalibration provided by Mixup or how the proposed loss function improves the calibration, but the ideas behind *how* do we arrive at the proposed loss function.

3.2.1 Data Augmentation and Model Calibration

We have seen in chapter 2 that the calibration and refinement of a probabilistic classifier rely on a correct modeling of the data uncertainty, i.e. the degree of overlap between the classes in the target distribution. Note that both properties directly depend on the data distribution to be modeled, and not the model itself. The refinement is directly concerned with how the data is separated, i.e. if the data presents overlapping then the model must learn this overlapping. The calibration measures how reliably do we represent this overlapping using probabilities.

We have also seen that DA can be framed through the lens of VRM, where the idea is that the input manifold is extended with transformations that are likely to provide samples closed to the training set, either through expert knowledge (e.g. rotations or translations when the inputs are images) or general purpose DA techniques such as Mixup. However, note that both Mixup and human-driven DA techniques share a common issue: they are not designed by analyzing or considering the properties of the input distribution $p(\mathbf{Y}|\mathbf{X})$ and the intersection of these with the probabilistic model $p_\theta(\mathbf{Y}|\mathbf{X})$. In other words, these DA techniques are not designed by considering how the inputs $\mathbf{X}$ of the different classes are distributed.

The reason is that modern instances of these models, such as DNN, are difficult to interpret, and high dimensional distributions are difficult to analyze. For that reason, the selection and performance of DA techniques depend, basically, on cross-validation; and there is no principled way to establish if a particular DA technique will boost the performance of a particular application or not. This does not mean, however, that DA techniques cannot improve the calibration properties or the robustness towards adversarial examples of an overparameterized model w.r.t. standard ERM (Hendrycks et al. 2020; Hendrycks et

al. 2019a; Hendrycks et al. 2019c; Hendrycks et al. 2019b), because the model will learn over a wider support of the input manifold.

We can thus conclude that there is no reason to establish that a particular DA technique will boost the calibration performance of a probabilistic model, beyond results obtained using model selection techniques e.g. (Wilk et al. 2018). This is because common DA techniques do not take into account the data uncertainty and hence the generated samples are not guaranteed to represent the uncertainty in the data distribution, as we shall see in the next subsection for the particular case of Mixup.

3.2.2 Mixup and Model Calibration

The main argument of the research around the idea that Mixup can improve the calibration and the robustness of probabilistic classifiers relies on the fact that this technique smooths the predictions performed by a model in the intersection between samples from the target distribution $P(\mathbf{X}, \mathbf{Y})$. However, even if this might reduce high-oscillations in the predictions performed in these regions of the feature space, or smooth the ultimate confidence assigned to these regions, this only ensures that the model will be less overconfident, which does not necessarily mean that the ultimate probability distribution will be calibrated.

Note that Mixup only ensures a linear-soft transition between the confidences assigned by the model in different parts of the input space, but it does not consider the proportion of samples present in different regions of the input distribution, which is at the core of a proper calibration. As a consequence, only if the data distribution presents a linear relation between their corresponding classes, one could expect to improve the calibration by using the generated samples.

In the experimental section, we show that some models trained with Mixup do not necessarily improve the calibration, as recently noted by (Thulasidasan et al. 2019). In fact, we show that Mixup tends to worsen the calibration in many cases.

Toy example

To illustrate the ideas discussed so far we make use of the following toy example illustrated in Figure 3.1.

The top row from the figure shows three common toy datasets employed by the community, which are generated using Scikit-Learn (Pedregosa et al. 2011). With circles and light red, we represent one of the classes, while in light blue and with triangles we represent the other class, both sets of samples being drawn from the data generating distribution $P(\mathbf{X}, \mathbf{Y})$.

In the bottom row, we represent the training data along with samples generated using Mixup[1]. The virtual samples are represented with a star. The position of the star represents the virtual location $\mathbf{X}$, while the color of the start represents the virtual label $\mathbf{Y}$. The more red means the virtual label corresponds to class red, and vice-versa. Thus purple represents labels with a confidence closed to 0.5.

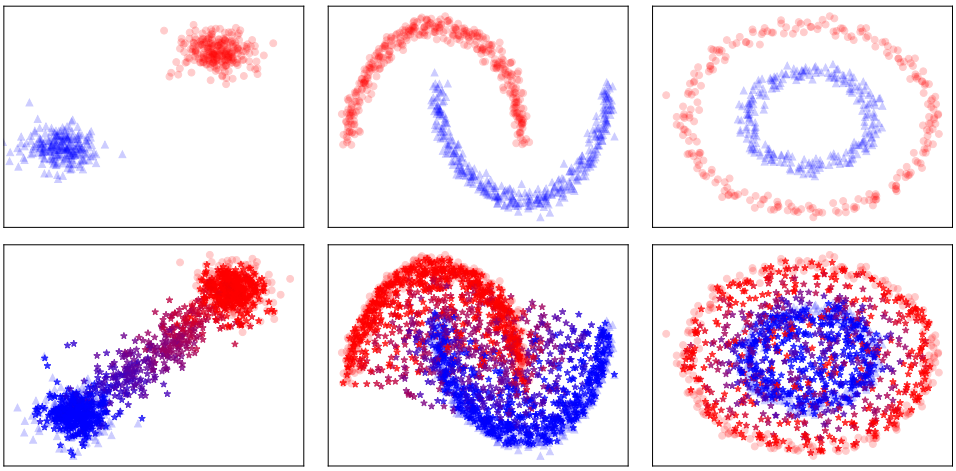


Figure 3.1: This figure illustrates the effect of Mixup on the data distribution. The top row shows samples from the data generating distribution $P(\mathbf{X}, \mathbf{Y})$, while the bottom row shows the training data alongside with virtual samples generated using Mixup. We can see how Mixup can sometimes generate samples representative of the data generating distribution (left plot) but that is not necessarily true for any distribution (middle and right plot).

[1]I recommend zooming in the pdf.

For ease of exposition, let's concentrate on the circles dataset (top left plot). A reasonable classifier for this dataset could be parameterized by a categorical distribution in which the parameters are given by a linear projection of the data locations $p(\mathbf{Y}|\pi = \mathbf{WX})$. If we think in a non-Bayesian setting, i.e. when we don't integrate out the model parameters $\mathbf{W}$, then the learned decision threshold around $p(\mathbf{Y}|\pi = \mathbf{WX}) = 0.5$, will be in the middle between the blue and red circles, going from the top left to the right bottom corner of the plot. Since both circles are totally separable, the model will learn to assign nearly extreme confidences (~ 1.0) towards the different classes around this decision threshold.

It seems reasonable to think that both set of samples have been sampled from two Gaussian distributions with small variance, hence there is no overlapping between both of them in general. However, we know Gaussian distributions have support over all $\mathbb{R}^2$, hence there is a chance that samples from different distributions can overlap. Moreover, we do not want extreme confidences around a decision threshold (unless the data is obviously totally separable). In this dataset, it seems reasonable to assign extreme confidences where most of the training data appears, and relax these confidences when we get close to the intersection between both distributions.

If we now take a look at the corresponding figure from the bottom row, we can see how Mixup is able to generate samples that *agree* with how this data is distributed. The virtual samples generated towards the intersection between both distributions starting from the red side are red and gradually become more purple. The same effect happens when starting on the blue class. Hence we can expect that a classifier learned with these generated samples will smooth the predictions, will improve generalization and will learn a calibrated distribution because the virtual samples are representative of the data distribution i.e. the *data uncertainty*.

If we, however, take a look at the middle and right plots we can see how there are virtual samples which are not representative of the data distribution. Of course, there are some that are, but we can see how in regions where the virtual sampler generates both purple and blue/red samples. Even more important we can see extreme (red) samples in regions where we would only expect blue samples, and vice-versa. This does not mean Mixup won't be able to improve the generalization capabilities of the classifier. We could expect the resulting classifier to provide smoother predictions (less overconfident) and to improve the performance, but the virtual samples are not representative of the data distribution, hence we cannot expect the resulting classifier to be calibrated.

In high dimensional distributions, this effect is harder to analyze, and it is impossible to know a priori how the data is distributed in order to use a vicinity distribution that can generate samples representative from the target distribution. This does not mean that data augmentation techniques cannot improve the calibration performance of a probabilistic model, but that can only be ensured using model selection techniques.

3.3 The Auto Regularized Confidence Loss Function

In this section, we introduce the proposed solution to the uncalibration of DNN trained with Mixup. We have seen so far that in order to have a calibrated probabilistic model we need the virtual data to be representative of the target distribution.

Note that the goal of a probabilistic classifier is to map any data distribution into a linear separable manifold, where a final linear projection and a link function is applied to provide the parameters of the likelihood distribution. This separability can only be achieved if: 1) the data is separable in its origins and 2) the model has enough capacity to do so. Thus, if 1) or 2) do not hold (which is something that we will not typically know in high dimensional distributions), then it seems unreasonable to force the model to learn towards $\{0, 1\}$ probabilities[2]; and we should expect an overparameterized model to experiment different pathologies such as overfitting (Vapnik 1998), memorization (Zhang et al. 2017), bad calibration (Guo et al. 2017), or sensitivity to adversarial attacks (Szegedy et al. 2014).

A very illustrative example of this pathology is: Why should we push probabilities towards 1.0 in a 1-dimensional input generative Gaussian classifier if Gaussians have support over $\mathbb{R}$? Based on this observation a training loss in a modern probabilistic model should somehow consider this inherent structure (uncertainty) in the data to reliably target the underlying distribution, and avoid the great ability of DNN to assign $\{0, 1\}$ probabilities when we do not know if the distribution to be modeled is or can be linearly separated after projecting the input $\mathbf{X}$. This is the core idea of our proposed loss function and in the previous section we used it to justify why Mixup should not necessarily provide calibrated distributions.

[2]Note that this what we force the model to learn when we use a categorical (multiclass) or Bernoulli (binary) likelihood to learn a probabilistic classifier, unless the rare case in which two exactly equal samples are labeled towards different classes; something we don't usually observe in the datasets used to evaluate the proposed approach in this chapter.

The solution we adopt encourages the probabilistic classifier to assign confidences based on its discriminative capabilities, through the incorporation of a super simple measure of data uncertainty. Thus, our loss function is inspired by how optimality is achieved in a Bayes Decision Rule (BDR) scenario, and I personally claim that this has to be done to achieve reliable probability distributions. As we mentioned at the beginning of this chapter, the main takeaway from this section is not the proposed loss function, or the fact that Mixup training does not always provide calibrated distribution, but the incorporation of a form of data uncertainty in the loss function in order assign probabilities, since that is what BDR tell us we must do. In fact, I will show that the proposed loss function is not perfect and that a lot of work can be still done in this direction.

3.3.1 Proposed Solution

The goal of the proposed solution is to benefit from the improved accuracy of Mixup training, but providing better-calibrated distributions. We do this by introducing a new loss function which is a weighted combination of our proposed loss, named Auto-Regularized-Confidence (ARC), and the categorical cross-entropy (CCE). The ARC loss is inspired by the Expected Calibration Error (ECE). The idea, as discussed before, is to incorporate data uncertainty in the predictions. This is done by first partitioning the confidences, $p = p_\theta(\mathbf{Y}|\mathbf{X})$, assigned to a batch of samples $\mathbf{X}$, into M equally spaced bins B_i; and matching these confidences to the accuracy μ_i in that bin, by means of any of these two loss variants:

$$\begin{aligned} \text{ARC\_V1} &= \frac{1}{M}\sum_{i=1}^{M}\left[\left(\frac{1}{|B_i|}\sum_{0<j\leq|B_i|} p_{ij}\right) - \mu_i\right]^2 \\ \text{ARC\_V2} &= \frac{1}{M}\sum_{i=1}^{M}\left[\frac{1}{|B_i|}\sum_{0<j\leq|B_i|} \left(p_{ij} - \mu_i\right)^2\right] \end{aligned} \tag{3.5}$$

where p_{ij} denotes the confidence of sample $\mathbf{X}_j$ that lies in bin B_i. The difference between these two losses lies in whether the average confidence (ARC_V1) or the individual confidences (ARC_V2) are forced to match the accuracy. If we set $M = 1$ then our loss function is computed over the entire batch instead of considering different bins. We make the accuracy μ_i a constant value so

learning gradients only depend on the confidence assigned by the model. We can do this by computing the accuracy and then detaching its value from the computational graph.

Our loss is combined with the CCE to avoid the local minimum in which the network parameterize a prior classifier (i.e. the one which assigns confidences given by the prior distribution to all samples), as we found in our initial analysis. This is because a prior classifier is useless, but the trivial way of optimizing calibration. Thus the overall loss is given by:

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}, \theta) = \frac{1}{N} \sum_{n=1}^{N} \text{CCE}(\mathbf{X}_n, \mathbf{Y}_n, \theta) + \beta \text{ARC}(\mathbf{X}_n, \mathbf{Y}_n, \theta) \tag{3.6}$$

where β is a hyperparameter that controls the relative importance given to each of the losses and is established with a validation set.

Note that this new loss targets the uncertainty of the learned representation, through the accuracy. The accuracy is used to summarize the proportion of samples from different classes that are being "mixed", i.e. the degree of overlap. So it somehow represents how the representations that the model can learn are distributed. It is clear that the accuracy is a very simple statistical summary of the data uncertainty and it is left to future work the search for other quantifiers that could encode more useful information such as how samples are distributed in the input space.

Consequently, we can expect that by evaluating the CCE loss on the Mixup image $\widetilde{\mathbf{X}}$, and the ARC loss on the mixing images $\mathbf{X}_1$ and $\mathbf{X}_2$, one can benefit from the improved discrimination as learned by the CCE, but the ultimate confidences are assigned by how the classifier classifies samples $\mathbf{X}_1$, $\mathbf{X}_2$, which are the ones really representative from the target distribution $P(\mathbf{X}, \mathbf{Y})$, and not those $\widetilde{\mathbf{X}}$ virtually generated by Mixup. It is then clear that ARC incorporates data uncertainty, which will improve the model representation of the underlying distribution, and thus its calibration. To validate this procedure, in our work we experiment with variants that compute ARC loss over $\mathbf{X}_1$ and $\mathbf{X}_2$; and over $\widetilde{\mathbf{X}}$. In general, all datasets benefit more from the first approach. A discussion is provided in the experimental section.

3.3.2 Motivation behind the two loss variants

On the other hand, the idea of experimenting with the two variants of our loss named ARC_V1 and ARC_V2 is based on the following observation. The only difference between the two variants is whether we force the average confidence of a set of samples to match the accuracy, as performed by ARC_V1, or we force each individual sample to match the accuracy, as done by ARC_V2. ARC_V2 is proposed to avoid solutions in which the set of confidences assigned by the model presents high variance. This will avoid solutions in which, for instance, the network present a 90% accuracy on a set of samples, and the model assigns 0.8 confidence to half of the samples and 1.0 to the other half. In such a setting, the loss being minimized will be 0, but the ultimate goal will not be achieved. The possibility of computing our loss over separate bins is incorporated to reduce this effect. However, in practice, we expect both losses to work, as the ideal behaviour of a good representation as learned by a model should be to map all the samples of a given class to the same (ideally linearly separable) representation. If this happens, the aforementioned variance on the confidence assigned by the model is reduced.

3.4 Experimental Evaluation

3.4.1 Experimental Details

We perform several experiments that illustrate the main claims of this chapter. The code and loss hyperparameters (e.g. if the model uses ARC_V1 or ARC_V2) are open-sourced in Github[3]. In the following tables, we show average results and provide the specific results used to compute this average values in Github alongside with the specific values of the loss hyperparameters. We evaluate different calibration metrics, detailed in subsection 2.2.5 plus the Logarithmic Score (LS) and the Brier Score (BS) defined in Equation 2.15. The ECE and MCE are evaluated with a partition of 15 bins, as is common in many works using these metrics, e.g. (Guo et al. 2017). Our approach is compared to a recent technique designed to implicitly calibrate a probabilistic DNN, which uses a measure called Maximum Mean Calibration Error (MMCE) (Kumar et al. 2018). The MMCE loss is a function that evaluates the calibration of the model using kernels, see section 3 in (Kumar et al. 2018) for the loss definition and how can it be estimated using training samples.

[3] https://github.com/jmaronas/calibration_MixupDNN_ARCLoss.pytorch.git

Datasets: We evaluate a collection of classical benchmarks for the image classification task: CIFAR10 (Krizhevsky et al. 2009a), CIFAR100 (Krizhevsky et al. 2009b), SVHN (Netzer et al. 2011); and we also evaluate our model on more realistic problems such as the ones provided by Caltech Birds (Welinder et al. 2010) and Standford Cars (Krause et al. 2013), which contain bigger and more realistic images. Due to computational restrictions, we did not evaluate our model on ImageNet.

Models: We experiment with state-of-the-art configurations of computer vision DNN: Residual Networks (He et al. 2016a), Wide Residual Networks (Zagoruyko et al. 2016) and Densely Connected Neural Networks(Huang et al. 2017). Moreover, for each variant, we evaluate several configurations and models with and without Dropout (Srivastava et al. 2014). We find this interesting since a dropout model can be used to quantify uncertainties (Kingma et al. 2015b; Gal et al. 2016). For the ResNet, we add a Dropout layer after the whole network. We set the Dropout values according to the ones provided in the original works, or in available implementations, except for the ResNet where we use a 0.5 Dropout rate. We use pre-trained models on ImageNet for Birds and Cars, which are obtained from the PyTorch API. On these pre-trained models, we add a Dropout layer at the end just before the last linear projection. Models are optimized with stochastic gradient descent with momentum and by placing a Gaussian prior over the parameters (L_2 regularization a.k.a. weight decay). The precision of this Gaussian prior is set according to the provided implementations or original works. For all the datasets except Birds and Cars, we use a learning rate starting from 0.1. For Birds and Cars the initial learning rate is set to 0.01. We use step learning rate scheduler that varies depending on the model. Additional details can be found in the code.

Data Augmentation Hyperparameters: Regarding Mixup hyperparameters we used the ones provided in the original work (Zhang et al. 2018). On the datasets where this technique was not evaluated, we searched for the optimal value on a validation set. This hyperparameter is then fixed for the rest of the experiments carried out. More details on Github.

ARC hyperparameters: Our loss hyperparameters: β, the number of bins M and the type of cost used (ARC_V1/ARC_V2) were searched using a validation set with the ResNet-18 (with and without dropout) on each dataset. The selected hyperparameter was then used with the rest of the networks. This way of searching hyperparameters is not optimal, since we should search within each particular experiment. The reason we did this was due to computational restrictions. Since our main goal was to extract conclusions on a possible good configuration of our loss function we need to do a big battery of experiments (we trained more than 1000 Neural Networks to evaluate the loss); and with our resources we could only do this in a feasible amount of time if we used the cheapest Neural Network we considered. We provide an additional discussion related to this point in Appendix A.

Our search includes all the possible combinations of: loss ARC_V1 and ARC_V2; number of bins: $M = 1$,$M = 15$ and $M = \{5, 15, 30\}$ (for this one the loss is computed three times, one per each value of M, and the three losses are then averaged); and evaluation of the ARC loss over the Mixup image $\widetilde{\mathbf{X}}$ or the separate images $\mathbf{X}_1$ and $\mathbf{X}_2$. This experiment was essential to validate our claim regarding data uncertainty and calibration, as exposed in subsection 3.3.1. We select the hyperparameters that provided a good accuracy with low calibration error. The specific loss hyperparameters of each of the models are provided in Github.

3.4.2 Reported Results

For the sake of illustration, we provide average results of all the DNN used for each dataset in Table 3.1 and the best performing model used in these averages in Table 3.2. The individual results used to compute these averages are provided in PDFs in the provided repository since I have realized that including them in this manuscript ends up messing things up (even in the appendix).

These tables show the accuracy (ACC) and the Expected Calibration Error (ECE) and the rest of calibration metrics are provided in Appendix A.

Table 3.1: Table showing average accuracy and ECE in (%) of all the models considered in this work

	CIFAR10		CIFAR100		SVHN		Birds		Cars	
	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE
Baseline (B)	94.76	3.41	77.21	11.57	96.32	1.90	78.51	2.39	86.74	2.06
Baseline + Mixup (B+M)	96.01	4.35	80.04	3.71	96.41	5.00	79.63	14.22	86.67	18.13
MMCE (M)	94.24	2.17	72.68	3.71	96.28	1.78	78.78	1.95	86.83	2.23
MMCE + Mixup (M+M)	91.90	5.69	78.52	5.48	96.59	2.83	79.99	12.37	86.03	13.07
ARC (A)	94.82	3.37	77.04	11.31	96.26	1.87	78.52	2.70	87.78	2.76
ARC + Mixup (A+M)	95.90	1.62	79.84	2.42	96.02	2.17	79.74	4.95	89.63	2.84

Table 3.2: Table showing the accuracy and ECE in (%) of the best model per task and technique.

	CIFAR10		CIFAR100		SVHN		Birds		Cars	
	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE
Baseline (B)	95.35	2.97	79.79	5.06	97.07	0.50	80.31	4.34	89.13	2.57
Baseline + Mixup (B+M)	97.19	4.65	82.34	1.42	96.97	4.91	82.09	10.14	89.45	18.10
MMCE (M)	95.58	1.21	74.98	7.04	96.90	0.49	80.64	3.28	89.40	2.70
MMCE + Mixup (M+M)	97.02	1.11	81.31	4.46	97.17	3.69	82.41	10.93	88.47	11.56
ARC (A)	95.99	2.01	80.77	4.73	97.08	0.37	80.32	4.44	90.09	1.92
ARC + Mixup (A+M)	97.09	1.03	82.02	0.98	96.82	2.20	82.45	1.28	91.13	2.40

3.4.3 Analysis of Results

First, as shown in rows B (Baseline) and B+M (Baseline+Mixup) we see how Mixup degrades the calibration except in CIFAR100. By comparing with the results reported in (Thulasidasan et al. 2019) we can conclude that Mixup behaves particularly well in CIFAR100, probably because the intersection between classes can be explained through a linear relation. However, our tables demonstrate that this is not a general behaviour of Mixup as shown in the rest of datasets.

It is surprising how Mixup degrades calibration in Birds and Cars, even though the DNN used for these datasets are pre-trained models which have been shown to provide better-calibrated distributions (Hendrycks et al. 2019b). In general, our results contrast with those reported in (Thulasidasan et al. 2019) where they provide general improvement in calibration performance due to Mixup. We can explain this difference with the fact that different models are used. For instance, while they use a VGG-16 and a ResNet-34, we are using much

deeper models, such as a ResNet-101 or a DenseNet-121. The difference can be connected to the observation in (Guo et al. 2017) where they show that calibration is further degraded by deeper architectures. Moreover, we shall emphasize that our results on CIFAR10 are on the state-of-the-art ($\sim$ 97% ACC) and much better calibrated (1.03 top ECE and 1.62 average ECE) than in (Thulasidasan et al. 2019), as they report a 2.00 value of ECE.

Analyzing our loss function, we see how it can correct the miscalibration introduced by Mixup training. In CIFAR10 and CIFAR100 A+M is the best performing approach. In SVHN we see that A+M corrects the calibration error introduced in B+M, but the approach behaves worse than the baseline. SVHN is a dataset that presents good calibration in many models over the test set, as noted also in (Guo et al. 2017; Maroñas et al. 2020). Finally, regarding Birds and Cars we see how our loss can largely correct the miscalibration introduced by Mixup. This means that our approach also performs well with pre-trained models on ImageNet. It should be noted that in this case, we do not achieve the same ECE error in Birds and Cars as with the baseline model. However, we have much better accuracy (over 3% on average results in Cars). In fact, our work reports nearly state of the art accuracy in Cars using a Dense-Net, where the best performing reported model has an accuracy only two points above but using much more complex architectures such as efficient net (Tan et al. 2019) or inception (Szegedy et al. 2016) (which are also pre-trained on ImageNet). On the other hand, our method is better than the recently proposed MMCE (Kumar et al. 2018). We found this method to be unstable in some cases, as some models saturated during training or tended to degrade the accuracy, as shown in the tables.

Regarding the parameterization of the loss function, we found that most of the time the best configuration of hyperparameters was obtained with ARC_V1. We also found that $M = 1$ is a reliable choice for this hyperparameter. This can be explained by the fact that DNN typically learn invariant representations and thus, we avoid the pathological behaviour that ARC_V1 can present, which is discussed in subsection 3.3.2. Besides, we found that only in Birds and some CIFAR100 models, the ARC loss computed over the Mixup image $\widetilde{\mathbf{X}}$ worked better than when computed over $\mathbf{X}_1$ and $\mathbf{X}_2$, even though this configuration also improved the calibration. Thus, as we claim in subsection 3.3.1, it seems reasonable that a loss function that takes into account, separately, the underlying structure present in the data distribution can provide better-calibrated uncertainties. To enhance this last claim, we emphasize that in some cases the β hyperparameter was 40 times greater than the CCE loss. This enhances the beneficial influence that our loss function can have on several problems.

Table 3.3: This table shows the results of applying the ARC loss just to a validation set.

	CIFAR100				CIFAR10				SVHN			
	validation		test		validation		test		validation		test	
β	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE
0.5	82.74	4.32	77.97	9.48	97.28	1.29	95.29	2.82	98.50	1.21	96.45	2.32
1.0	86.46	4.29	78.74	10.65	97.92	0.88	94.92	3.39	98.86	0.54	96.48	2.33
2.0	91.26	2.17	79.49	9.19	99.08	0.27	95.25	3.05	99.08	0.33	96.54	2.37
4.0	94.30	1.61	79.61	8.86	99.74	0.21	95.60	2.63	99.24	0.24	96.50	2.33
8.0	96.26	1.08	78.85	9.73	99.84	0.18	95.58	2.72	99.32	0.18	96.67	2.13

3.4.4 A final insight on the experiments

Finally, we discuss one drawback of our proposal as being used as a general-purpose calibration tool. Note that, if applied on a DNN that presents near 100% accuracy on the training dataset (which is the case in many of the standard databases tested) then the ARC loss will provide the same learning signal as the CCE, because it will force the average confidences to be 1.0. This means that it will not work in datasets where the training error is overfitted, as in CIFAR100. This explains why applying ARC loss over the Baseline model (A in the tables) does not significantly improve the calibration over the baseline (B in the tables).

A possible solution could be to apply the ARC loss on a separate validation set. To do so, we experiment with the following variant. We take a validation split from the training dataset where the DNN presents uncalibrated over-confidences. Let say that this validation set presents an 80% accuracy, with a 0.99 average confidence. Thus, we use the validation set to compute the ARC loss while the training dataset is only used for the CCE.

Surprisingly, the DNN learns to minimize the ARC loss by increasing the accuracy and the confidence assigned to this validation set rather than by relaxing the confidences assigned (even though the ARC loss does not directly push the probabilities to be 1.0). This is shown in Table 3.3 where results are reported by varying the hyperparameter β. We can see how when β grows so does the accuracy on the validation set rather than decreasing the calibration on the test set. This means that the model is learning to correctly classify the vali-

dation set (perfect accuracy) assigning extreme 1.0 confidences (nearly perfect calibration). Note in the table how the ECE decreases when β grows.

3.5 Conclusions

This chapter has shown and motivated why Mixup does not ensure calibrated distributions. The results and theory presented suggest that a similar analysis should be employed over different DA techniques, which is left for future work. We have also opened a new perspective to reduce overconfidence in DNN. As we cannot control how a model might overfit the dataset to achieve high discriminative performance, a good practice is to auto-regularize the model to incorporate the uncertainty of the learned representations. This work has shown a way of doing this on Mixup training, reporting state-of-the-art results in accuracy and calibration. Future work is concerned with the exploration of new loss functions for this purpose.

Finally, as noted by one of the reviewers of this manuscript, a nice experiment to provide more light into the results and conclusions discussed in this chapter is to substitute the ARC loss by the categorical cross-entropy, i.e. to apply the CCE both over the mixup image and over the separated images. The motivation is the following: the CCE is a proper scoring rule, so, as we noted in the previous chapter, it should recover both discrimination and calibration. Hence it seems reasonable to check whether the ARC can be substituted by the CCE to fix the ultimate confidences adequately. Note that the CCE can still overfit, since in the case where different datapoints $\mathbf{X}, \mathbf{Y}$ do not overlap, it will end up assigning a $\{0, 1\}$ confidence, as we have discussed previously in this chapter. Moreover, we should also check if other PSR such as the Brier Score can replace the ARC as well.

Chapter 4

Recalibration of Deep Probabilistic Models using Bayesian Neural Networks

This chapter presents a methodology to recalibrate the output of a DNN *using a Bayesian Neural Network (*BNN*). The uncalibrated logits from a* DNN *are passed through a* BNN *which maps the uncalibrated logits to calibrated ones. We present and analyze the proposed approach and discuss potential lines of improvement.*

In the first chapter of this thesis, we motivated the use of Bayes Decision Rule in order to take optimal decisions and introduce some related concepts. Then, in the second chapter, we make use of this theory to motivate why Mixup training does not necessarily provide calibrated distributions and, even more important, to show how a simple modification of the loss function that takes into account the uncertainty in the learned representation can fix the miscalibration that Mixup can introduce.

However, one of the main problems of these implicit calibration techniques, i.e. techniques that aim at providing a model that is directly calibrated, is the high cost of training and experimentation when the model is parameterized by a DNN. An alternative solution to implicit techniques is to design calibration techniques which takes the output probability of an uncalibrated model and map it to a calibrated one.

In this chapter, we show how to implement this post-calibration technique using a Bayesian Neural Network (BNN), i.e. a Neural Network in which the parameters of the model are inferred and then marginalized out to make predictions.

We start this chapter by reviewing the basic idea of post calibration. We then review the state of the art emphasizing pros and cons of different approaches. We then motivate the use of a Bayesian Neural Network and the chosen (approximate) inference algorithm. We finally present results, conclusions and future research directions. Part of this future work has already been started by one of my MSc students which is now pursuing a PhD in this research direction, trying to answer some of the open questions of this chapter. Most of the ideas presented in this chapter correspond to those described in (Maroñas et al. 2020).

4.1 Introduction to Post Calibration

The goal of post-calibration techniques is to map the prediction of a model[1] to calibrated class probabilities. Perhaps one of the most well known classical algorithms for this purpose in the context of Binary Classification is Platt Scaling (Platt 1999).

If we denote with $\mathbf{X}$ the output of the model, then the idea behind Plat Scaling is to learn a logistic regression model from $\mathbf{X}$ to $\mathbf{Y}$ with parameters $\mathbf{w}, \mathbf{b} \in \mathbb{R}$. In other words, Plat Scaling models the outputs $\mathbf{Y}$ with a Bernoulli likelihood $p(\mathbf{Y}|\sigma(\mathbf{w}\mathbf{X} + \mathbf{b}))$ with $\sigma()$ being some link function, for example the Sigmoid function.

This method can be extended to a multiclass setting with C classes by considering the following model: $p(\mathbf{Y}|\sigma(\mathbf{W}\mathbf{X} + \mathbf{b}))$ with $\mathbf{b} \in \mathbb{R}^C$ and $\mathbf{W} \in \mathbb{R}^{C \times C}$, given $\mathbf{X} \in \mathbb{R}^C$. In this case the link function is usually the Softmax function and the likelihood is given by a Categorical distribution.

These post-calibration models are trained by using a different set of samples to train the model and the Plat Scaling calibration stage.

[1] Note that it does not have a probabilistic model but just a model used to make predictions such a Support Vector Machine.

4.1.1 Deep Neural Networks are Uncalibrated

The work from (Niculescu-Mizil et al. 2005) showed that Neural Networks usually produce well-calibrated probabilities in binary classification problems, which is to be expected since we have already seen that the training criteria of Neural Networks are proper scoring rules. However, (Guo et al. 2017) analyzes the calibration of modern state of the art Deep Neural Networks, which are radically different from those used by (Niculescu-Mizil et al. 2005). To their surprise, they showed that modern DNN are badly calibrated, a phenomenon that links to the over parameterization of these models as discussed in (Guo et al. 2017) and section 3.3 in this thesis.

In their study, (Guo et al. 2017) analyzes how different modern regularization techniques affect calibration. This includes weight-decay, dropout, model capacity, batch normalization etc. Moreover they popularize the use of the MCE and ECE calibration measures.

One of the most important parts of this work is the study of different post-calibration techniques for the purpose of recalibrating the output of a DNN. The idea is that the logits of a DNN play the role of $\mathbf{X}$ in section 4.1, and the calibration technique maps this uncalibrated logit to a calibrated one. The authors compare different classical post-calibration techniques including the aforementioned Plat Scaling, but also Histogram binning (Zadrozny et al. 2001), Isotonic regression (also known as PAV) (Zadrozny et al. 2002) and Bayesian Binning into Quantiles (Naeini et al. 2015). These binary post-calibration methods are extended to the multiclass setting.

Among all these techniques the authors propose a novel post-calibration stage named Temperature Scaling (TS). The idea is to replace the parameters $\mathbf{W}$ and $\mathbf{b}$ in Plat Scaling, by a single parameter $\mathbf{T}$ which is multiplied by all the logits. The authors show that on DNN this post-calibration technique consistently improves the calibration among the other classical techniques with the improvement that TS does not change the accuracy of the DNN.

The results from this work let the authors conclude that the calibration space is inherently simple since the calibration is usually corrected by simpler models. This is the starting point of the contribution of this chapter. Our main hypothesis is that we can actually use complex models as long as we turn them into Bayesian models. As we have seen in the last section from chapter 2, Bayesian inference provides a principled way to account for uncertainty in the parameters of a model, making the predictions more robust to overfitting.

4.1.2 The benefits of post-calibration techniques

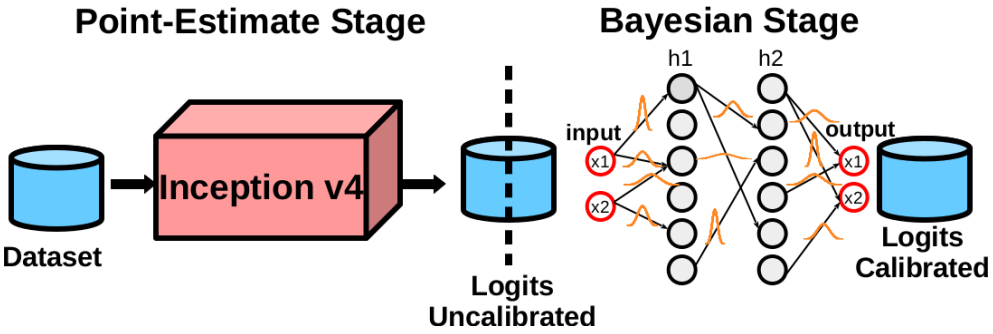
Before introducing the methodology of this section, we motivate why post-calibration (explicit) techniques can be useful since I have found that some researchers argue that what is interesting is to directly have a calibrated model (implicit techniques). However, as I mentioned at the beginning of this chapter, this can have a high computational cost in a modern DNN.

Other benefits of post-calibration techniques include the fact that the post-calibration stage is only compromised by the dimensionality of the logit space, no matter how challenging the initial task is, or the type and complexity of the pre-trained DNN. Moreover, these approaches are efficient, since the initial DNN does not need to be re-trained for re-calibration. Some approaches that attempt to directly train a deep calibrated model (Kumar et al. 2018; Seo et al. 2019) increase the training time over the initial DNN. In this sense, hyperparameter search is quicker with post-calibration stages, as one only need to focus on getting good accuracy from the DNN. Obviously one can benefit from combining explicit and implicit calibration techniques. For instance, the best results reported by (Kumar et al. 2018) are a combination of their method with TS. On the other hand, post-calibration techniques do not compromise the architecture of the DNN. For example, implicit techniques such as (Gal et al. 2016; Seo et al. 2019) require certain architectures in the previous stage. Finally, any future improvement can be incorporated into the post-calibration stage. For example in our case, we can train BNN post calibrators using alternative inference algorithms without affecting the previous stage.

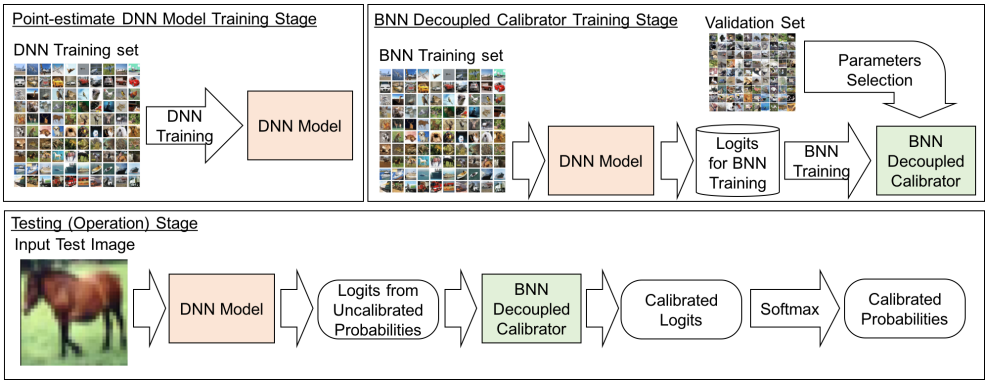
4.2 Bayesian Neural Networks as Post-Calibration technique

Since the work from (Guo et al. 2017) many techniques have been proposed to solve this miscalibration in DNN. Perhaps one of the most popular ones are model ensembles (Lakshminarayanan et al. 2017; Wenzel et al. 2020; Mariet et al. 2021; Havasi et al. 2021) but they are clearly inefficient in nature, although much work is being done in making them more efficient. There are other proposed techniques which range from different applications (classification vs regression) to different modeling design (implicit vs explicit), see e.g. (Kumar et al. 2018; Kuleshov et al. 2018; Seo et al. 2019).

All these techniques share something in common: their design is based on point estimate approaches, e.g. maximum likelihood. However, as we will justify in the next section, a proper address of uncertainty, as done by Bayesian approaches, is a clear advantage towards reliable probabilistic modeling; a fact



(a) An example of the architecture of our proposed model. On the left top figure, an expensive DNN is trained on a dataset. Then, the (uncalibrated) output of such DNN is the input to the BNN calibration stage. The inputs and outputs of the Bayesian stage have the same dimensionality (given by the number of classes). Orange Gaussians on each arrow represent the variational distributions on each parameter.



(b) This figure represents a description of the training, validation and test stages of the proposed model.

Figure 4.1: A graphical description of the proposed architecture

that has been recently shown for example in the context of computer vision (Kendall et al. 2017). Despite these well-known properties of Bayesian statistics, they have received major criticisms when they are used in DNN pipelines, mainly due to important limitations such as prior selection, memory and computational costs, and inaccurate approximations to the distributions involved (Lakshminarayanan et al. 2017; Kumar et al. 2018; Kuleshov et al. 2018), although there has been some work trying to solve these problems and understand these limitations for model calibration and robustness (Dusenberry et al. 2020; Wilson et al. 2020; Izmailov et al. 2021).

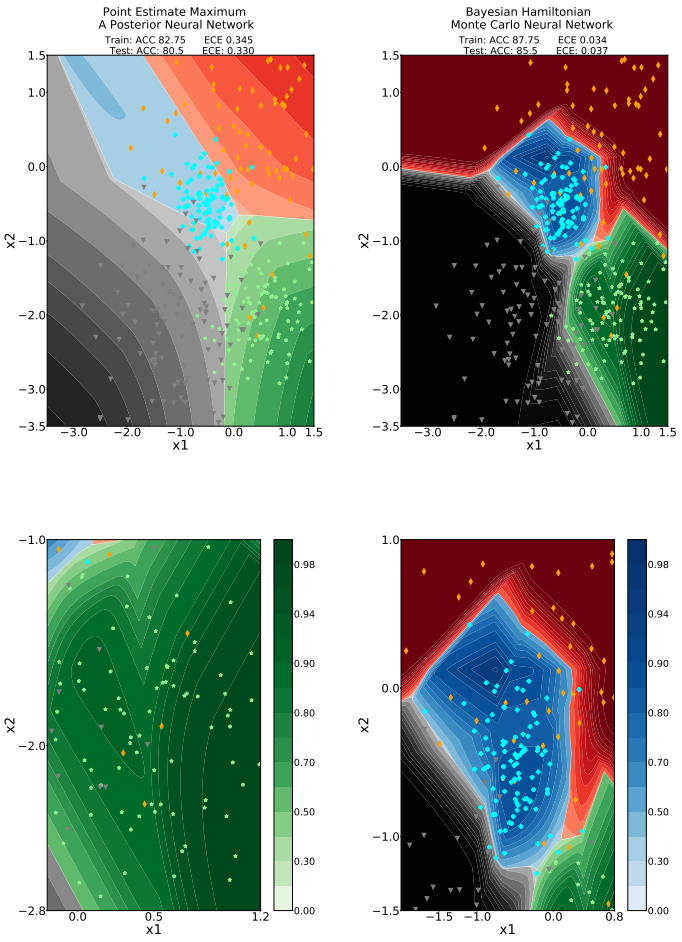


Figure 4.2: Decision thresholds learned by a Neural Network on a 2-D toy dataset problem where four classes are considered, each one represented with a different colour and marker style. The plot represents the confidence assigned by the model towards the most probable class, in each region of the input space. Darker colours represent higher confidences. The subfigure on the top row left corner represents the decisions learned by a point-estimate model obtained by minimizing the loss function given by the posterior distribution (MAP optimization) in Equation 2.30; and the figure on the top row, right corner, represents the confidences assigned by a Bayesian model that uses Hamiltonian Monte Carlo to draw samples of the posterior distribution, which are used to approximate the posterior predictive, see (Brooks et al. 2011) for details. Bottom rows represent zooms to different regions of the input space, showing the decision thresholds learned by the Bayesian model. Each figure represents the Accuracy (ACC) (the higher the better); and the Expected Calibration Error (ECE) (the lower the better). With markers, we plot the observed data $\mathbf{X}$ with their corresponding label $\mathbf{Y}$ being represented by the marker color. Figure best viewed in color.

In this work we aim at bridging this gap, i.e. being able to combine the state-of-the-art accuracy performance provided by DNN, with the good properties of Bayesian approaches towards principled probabilistic modeling. Following this objective, we propose a new procedure to use Bayesian statistics in DNN pipelines, without compromising the whole system performance. The main idea is to re-calibrate the outputs (in the form of logits) of a pre-trained DNN, using a decoupled Bayesian stage which we implement with a Bayesian Neural Network (BNN), as shown in Figure 4.1.

4.2.1 Bayesian Modeling and Calibration

In subsection 2.3.5 we have provided a discussion around why making Bayesian predictions are likely to provide more robust and calibrated predictions than point estimate networks, even under model misspecification.

The purpose of this subsection is to analyze what happens in a toy example by comparing point-estimate with Bayesian predictions. To do so, let's take a look at Figure 4.2.

This figure shows the training points and the confidences respectively assigned by a Neural Network with a given topology on a 2-D toy dataset, where four classes are considered, each one represented with a different color. The darker the color, the higher the confidence assigned in that region, as illustrated by the vertical color bars in the figures from the bottom row. The likelihood model is quite possibly misspecified w.r.t. the data generating distribution since this data distribution is obtained from a linear Gaussian model $p(\mathbf{X}|\mathbf{Y})p(\mathbf{Y})$ (the same one used in chapter 2) and hence we know that the posterior distribution $p(\mathbf{Y}|\mathbf{X})$ is not the Categorical distribution.

The only difference between the figure of the top row relies on whether the predictions are done by optimizing the posterior distribution (top left figure) or by drawing samples using a custom implementation of Hamiltonian Monte Carlo (top right figure), see chapter 5 in (Brooks et al. 2011). This custom implementation is provided by clicking on the following link: HMC_implementation. The bottom row zooms in different parts of the top right figure.

We can see that the Bayesian model assigns better probabilities in the sense that it better reflects the inherent structure in the data, thus being closer to the optimal decision rule. This is quantitatively reflected by the values of the accuracy and the expected calibration error (ECE) showed at the top of the plots. Moreover, it can be seen how the different models assign different confidences to each region of the input space. For the sake of illustration, in

the bottom row, we present two different concrete parts of the input space. We can clearly see how the Bayesian model assigns confidence being coherent with what the input distribution presents: highest confidence (close to 1.0) in regions where only one class is presented and moderate probabilities in regions where the data from different classes overlap. The point estimate does not present this behavior.

4.2.2 Proposed Solution

Figure 4.1 shows the main architecture of the proposed solution, which is shared with other post-calibration techniques. First, we train a DNN on a specific task. After training is finished, we project each input sample to the logit space, i.e. the pre-softmax, by forwarding the data through the DNN. Second, we train a Bayesian stage, which is responsible for mapping the uncalibrated logit vector of values provided by the DNN, to a calibrated one. Note that once the DNN is trained, and the forward step is done for a given sample, the Bayesian stage does not require further access to the previous DNN to be trained.

The post-calibration technique is implemented with a Bayesian stage, which we decided to be a Bayesian Neural Network, rather than a Gaussian Process. We now justify the selected method, and how do we perform inference.

Bayesian inference as an infinite-dimensional optimization problem

To justify the proposed solution, it is convenient to present Bayesian inference as the solution of an infinite-dimensional optimization problem (Zellner 1988). Given the set of all probability distributions $\mathcal{P}(\Theta) = \{p(\theta) : \theta \in \Theta\}$, Bayesian inference can be restated as the solution to an infinite-dimensional optimization problem where the posterior $q(\theta) \in \mathcal{P}(\Theta)$ is given as the solution to the following minimization problem:

$$q^*(\theta) = \underset{q(\theta)\in\mathcal{P}(\Theta)}{\operatorname{argmin}} \int -\sum_{n=1}^{N} \log p(\mathbf{Y}_n|\mathbf{X}_n, \theta) q(\theta) \mathrm{d}\theta + \mathrm{KLD}[q(\theta)||p(\theta)] \qquad (4.1)$$

with $p(\theta)$ being the prior and $\sum_{n=1}^{N} \log p(\mathbf{Y}_n|\mathbf{X}_n, \theta)$ being the log likelihood.

One of the many interesting discussions in the work from (Knoblauch et al. 2019) is the interpretation of the elements in the loss function being minimized.

The left element is the data-driven term or the empirical risk minimizer, while the KLD$[q(\theta)||p(\theta)]$ is the uncertainty quantifier, i.e. the element that determines how uncertainty about θ is quantified with the resulting posterior $q(\theta)$. This observation plays a super important role in the approach we take in this chapter later on, and in the future lines of research.

On the other hand, the second relevant point to this thesis of the work from (Knoblauch et al. 2019) is the observation that Variational Inference (VI) can be seen as the same minimization problem of Bayesian Inference but restricting the space of possible solutions, i.e. by searching over a family $\mathcal{Q} \subset \mathcal{P}(\Theta)$. This implies that VI is the optimal procedure to approximate the posterior distribution using an element from $\mathcal{Q}$ when this posterior is analytically intractable.

This means that, from this perspective, using alternatives to VI produce suboptimal results. Why is then that alternatives to VI proposed in the literature work better than VI in certain problems?. There is an excellent answer in (Knoblauch et al. 2019) which attributes these performance gaps to the Rule Of Three which includes: how the prior is specified, the definition of a loss function i.e. how the likelihood is specified, and the set of possible posteriors, which includes the computational resources available to search over the set of possible solutions.

Thus, rather than using alternatives to VI, (Knoblauch et al. 2019) propose to solve any of these misspecifications in different ways. The computational one can be, in principle, solved using MCMC algorithms, or more expressive approximations to the posterior distribution that increases the subset $\mathcal{Q}$. The misspecification about the prior can be solved by using an alternative divergence to the KLD so that the way we quantify uncertainty changes. Note that a misspecified prior, i.e. a prior that does not really play the role that the prior should play, can bias our posterior towards a suboptimal quantification of uncertainty, which in turn will affect our final predictions, as already discussed in subsection 2.3.6. Finally, alternative loss functions can be used to solve the way in which the data is related to the parameters of the model. This in turn has been popularized as Generalized Bayesian Inference (Bissiri et al. 2016). Special cases of this last point include loss functions derived from alternative divergence criteria (Jewson et al. 2018), since the log-likelihood is related to the KLD, and we have already seen in subsection 2.3.3 that this divergence tends to cover the tails of the distributions (i.e. it will be non-robust to outliers).

Architecture of the Bayesian Neural Network

Having introduced the ideas from (Knoblauch et al. 2019) relevant to this thesis we can now describe and justify the proposed solution.

We start by describing the choice of the elements involved in the Bayesian Neural Network. The likelihood model $p(\mathbf{Y}|\mathbf{X}, \theta)$ is implemented with fully connected Neural Networks with ReLU activations for the hidden layers, and softmax activation for the output layer. Note that one can adapt the complexity and flexibility of this stage depending on the context, for instance by using recurrent architectures.

In this work we discard the use of Gaussian Processes (GP) since the number of cubic operations scales linearly with the number of classes. Also because GP need further approximations to the BNN such as the optimization of the inducing points. We shall point however, to a recent work presented after the publication of these work in where they have a fantastic idea to use GP for this task (Wenger et al. 2020) and solve many of the limitations I have pointed out. The problem with this approach is that it is only particularly suitable for this task and not a general-purpose technique that can be applied for GP in the task of classification. Either way, this contribution is brilliant for this particular task.

The prior distribution $p(\theta)$ of the Neural Network parameters is given by a standard Gaussian distribution, which is the standard choice in the literature. We will provide a discussion around this choice further in this chapter, since this choice implies analytical evaluation of the objective function. For the moment note that the role that the prior plays in a Bayesian Neural Network has to do with the prior over functions that it induces, since Neural Networks are used to model functions through a computational graph with parameters θ and the relation of them with the input $\mathbf{X}$.

Thus our ultimate interest is the prior over functions that the prior over the parameters and the computational graph induce. However, the prior over functions that BNN implement is difficult to interpret from the prior over the parameters, since the computational graphs are black-box function approximations difficult to analyze. In other words, since it is hard to analyze how parameters sampled from the prior distribution interact with the input $\mathbf{X}$ within the computational graph, it is difficult to analyze, interpret and specify a suitable prior over functions.

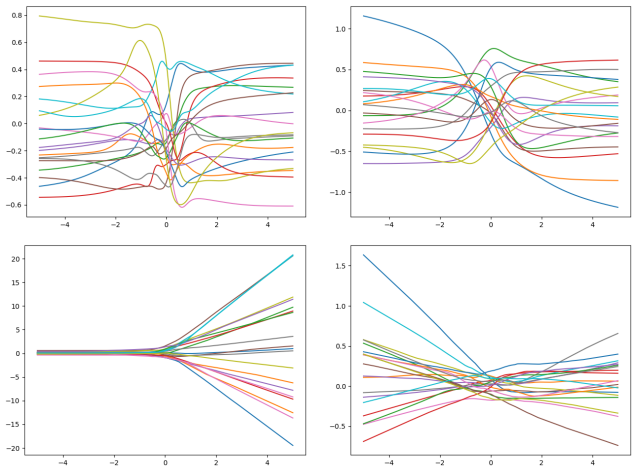


Figure 4.3: This figure shows the prior over functions that different BNN likelihood models induce when a standard Gaussian prior is placed over its parameters. On the top figure left we show a 10-layer Neural Network with residual connections and hyperbolic tangent activation function. Top figure right shows a one layer Neural Network with hyperbolic tangent activation. The bottom row shows Neural Networks with ReLU activation. The left plot shows a 10 layers residual network Neural Network and the right shows a one layer.

When this happens, it is likely that the kind of functions we model a priori are not representative of the task at hand, and the posterior distribution is biased towards a wrong quantification of uncertainty as noted by (Knoblauch et al. 2019). This point is central to the discussion and the future work presented at the end of the chapter. For the moment, and for the sake of illustration, Figure 4.3 shows the prior over functions of a 1-dimensional problem that different Neural Network architecture and a Gaussian prior induces. As we see, the differences are notable depending on the computational graph.

Predictions over test samples

In order to make predictions over a test point label $\mathbf{Y}^*$ given the corresponding input $\mathbf{X}^*$ we rely on a Monte Carlo integrator. Note that the form of the likelihood model as described above makes unfeasible the analytic computation of the posterior predictive distribution. However, this integral can be approximated by drawing samples from the posterior distribution and approximating the integral by a mixture of likelihood models parameterized by the computational graph:

$$p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{Y}, \mathbf{X}) = \int p(\mathbf{Y}^*|\mathbf{X}^*, \theta)p(\theta|\mathbf{X}, \mathbf{Y})\mathrm{d}\theta \approx \frac{1}{K}\sum_{k=1}^{K} p(\mathbf{Y}^*|\mathbf{X}^*, \theta_k); \theta_k \sim p(\theta|\mathbf{X}, \mathbf{Y}) \tag{4.2}$$

Since we use a categorical likelihood $p(\mathbf{Y}^*|\mathbf{X}^*, \theta_k)$ the resulting mixture relies on averaging the softmax output from the different K forward steps. Note that if we would make a DNN Bayesian, this would imply K forward through a super expensive DNN. However, with this approach, we only need one forward through the DNN and K much cheaper forwards through the BNN.

Inference

So far, we have described how the likelihood model is implemented and predictions are done. Note that the only thing we have not specified is how do we draw samples from the posterior distribution, i.e. how do we perform the operation $\theta_k \sim p(\theta|\mathbf{X}, \mathbf{Y})$. It turns out that the non-conjugacy of the prior and the likelihood does not allow us to compute this posterior analytically. Thus we need an approximation.

The gold standard, in this case, would be to run a variant of the Hamiltonian Monte Carlo algorithm, probably either by using the No U-Turn Sampler (Hoffman et al. 2014) or a stochastic gradient version of this algorithm (Chen et al. 2014), so that we can draw samples using mini batches. Other MCMC possibilities include stochastic Langevin dynamics (Welling et al. 2011) or more recently proposed ChEES-HMC (Hoffman et al. 2021). However, in practice, MCMC algorithms are super hard to diagnose, even more when the number of parameters to inspect increases. Moreover, the unidentifiability and multimodal posterior typical of ReLU networks would make the convergence to the target distribution even more harder, and the typical convergence diagnosis such as the Rhat will probably fail (Vehtari et al. 2021; Izmailov et al. 2021). On the other hand, MCMC approximations require storing all the samples that are used to make predictions, and drawing samples using HMC is computationally expensive, since one needs several reverse mode operations of the computational graph to get one sample.

For this reason, and also based on the ideas from (Knoblauch et al. 2019) (already introduced above) we decided to use Variational Inference. The idea behind VI is that we approximate the posterior distribution with an approxi-

mate distribution by minimizing the KLD between this approximate posterior $q_\phi(\theta)$ parameterized by ϕ and the true posterior $p(\theta|\mathbf{X}, \mathbf{Y})$. This, in turn, is equivalent to maximizing the Evidence Lower Bound (ELBO):

$$\begin{aligned}
&\text{KLD}[q_\phi(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})] = \int q_\phi(\theta) \left[\log q_\phi(\theta) - \log p(\theta|\mathbf{X}, \mathbf{Y})\right] \mathrm{d}\theta = \\
&- \int q_\phi(\theta) \log \frac{p(\mathbf{X}, \mathbf{Y}|\theta)p(\theta)}{p(\mathbf{X}, \mathbf{Y})} \mathrm{d}\theta + \int q_\phi(\theta) \log q_\phi(\theta) \mathrm{d}\theta = \\
&\log p(\mathbf{X}, \mathbf{Y}) - \int q_\phi(\theta) \log p(\mathbf{X}, \mathbf{Y}|\theta) \mathrm{d}\theta + \text{KLD}[q_\phi(\theta)||p(\theta)].
\end{aligned} \tag{4.3}$$

Rearranging terms we note that this minimization problem w.r.t. ϕ is equivalent to maximizing the ELBO:

$$\begin{aligned}
&\underbrace{\log p(\mathbf{X}, \mathbf{Y})}_{\text{Log marginal likelihood}} \quad -\text{KLD}[q_\phi(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})] = \\
&\underbrace{\int q_\phi(\theta) \log p(\mathbf{X}, \mathbf{Y}|\theta) \mathrm{d}\theta - \text{KLD}[q_\phi(\theta)||p(\theta)]}_{\text{ELBO}}
\end{aligned} \tag{4.4}$$

This objective function has two purposes. First, it minimizes a divergence between the approximate posterior and the target posterior distribution, but also maximizes the log marginal likelihood which we have also seen is the Bayesian criteria for model selection. Moreover, note that as we stated before, it is clear from Equation 4.4 that one can see Bayesian inference (see Equation 4.1) as the same minimization problem but over a family of distributions parameterized by ϕ, rather than the whole set of probability distributions.

In practice, gradients need to be obtained w.r.t. the variational parameters in order to maximize the above objective. For the case in which this posterior is Gaussian $q_\phi(\theta) = \mathcal{N}(\theta|\mu_\phi, \sigma_\phi^2)$, the KLD term of the ELBO can be computed analytically since the prior is also Gaussian. However, the expected log-likelihood needs to be approximated using Monte Carlo, leading to the final objective:

$$\begin{aligned}
&\text{ELBO} = \int q_\phi(\theta) \log p(\mathbf{X}, \mathbf{Y}|\theta) \mathrm{d}\theta - \text{KLD}[q_\phi(\theta)||p(\theta)] \approx \\
&\frac{1}{K} \sum_{k=1}^{K} \log p(\mathbf{X}, \mathbf{Y}|\theta_k) - \text{KLD}[q_\phi(\theta)||p(\theta)]; \theta_k \sim q_\phi(\theta)
\end{aligned} \tag{4.5}$$

Since gradients w.r.t. the variational parameters depend on this sampling procedure[2] $\theta_k \sim q_\phi(\theta)$, we apply the reparametrization trick to allow for unbiased and low variance gradient estimators, because this trick allows us to push the gradient into the expectation (in contrast to the reinforce gradient estimator). This trick can be applied for certain parametrizations of the variational posterior. For the case of a Gaussian distribution, the reparametrization trick applies as follows:

$$\begin{aligned} &\epsilon \sim \mathcal{N}(\epsilon|0, I) \\ &\theta_k = \mu_\theta + \epsilon\sigma_\theta \\ &\log p(\mathbf{X}, \mathbf{Y}|\theta_k) - \mathrm{KLD}[q_\phi(\theta)||p(\theta)] \end{aligned} \tag{4.6}$$

Thus the overall procedure to obtain gradients is given by:

$$\begin{aligned} &\nabla_\phi \mathrm{ELBO} = \nabla_\phi \int q_\phi(\theta) \log p(\mathbf{X}, \mathbf{Y}|\theta) \mathrm{d}\theta - \nabla_\phi \mathrm{KLD}[q_\phi(\theta)||p(\theta)] = \\ &\nabla_\phi \int p(\epsilon) \log p(\mathbf{X}, \mathbf{Y}|\mu_\theta + \epsilon\sigma_\theta) \mathrm{d}\epsilon - \nabla_\phi \mathrm{KLD}[q_\phi(\theta)||p(\theta)] = \\ &\int p(\epsilon) \nabla_\phi \log p(\mathbf{X}, \mathbf{Y}|\mu_\theta + \epsilon\sigma_\theta) \mathrm{d}\epsilon - \nabla_\phi \mathrm{KLD}[q_\phi(\theta)||p(\theta)] \approx \\ &\frac{1}{K} \sum_{k=1}^{K} \nabla_\phi \log p(\mathbf{X}, \mathbf{Y}|\theta_k) - \nabla_\phi \mathrm{KLD}[q_\phi(\theta)||p(\theta)] \end{aligned} \tag{4.7}$$

The trick that allows us to compute the expectation w.r.t. ϵ rather than θ (first to second row) is known as the LOTUS rule, with the theoretical background given by the change of variables of the pushforward measure. This trick is important also in the next chapter. It implies that expectations w.r.t. a distribution that is given by a transformation of samples from another distribution can be written as expectations w.r.t. the base distribution, provided that the transformation is measurable, as linear transformations involved in the case of Gaussian distributions are. A more extensive explanation of the procedure described here so far can be found in (Kingma et al. 2014; Rezende et al. 2014; Blundell et al. 2015). Note moreover that since we assume a factorized likelihood $\log \prod_{n=1}^{N} p(\mathbf{X}_n, \mathbf{Y}_n|\theta_k)$, the above objective can be evaluated without bias using a subset (minibatch) of the whole dataset.

[2] We need gradients w.r.t. ϕ since we are maximizing the ELBO.

Finally, we experiment with the local reparameterization trick (Kingma et al. 2015b). This trick aims at reducing the noise in the Stochastic evaluation of the Expected Log-Likelihood in the ELBO. The idea is to sample w.r.t. the distribution of the preactivation that is induced by the linear combination of the weights and the features of the Neural Network at each layer, rather than sampling directly the Neural Network weights. Since the variational distribution is Gaussian, the distribution of the preactivation is also Gaussian, with mean and variance given by a linear combination of the variational mean and variance with coefficients given by the data features. In this chapter, we will refer to the standard mean-field approach with MFVI and with the modified version that uses the local reparameterization trick as MFVILR. Note that the only difference between both approaches relies on the convergence of the algorithm to the optimal solution, since the Local reparameterization trick provides a variance reduction in the stochastic gradients used to compute the ELBO due to sample activities rather than weights, thus leading to faster convergence.

Training the Bayesian Neural Network with the training set

One difference between our decoupled BNN and other calibration stages is that we use the same set to train the DNN and the BNN since the validation set is used for another purpose (see next subsection). One could think, in principle, that this is methodologically wrong because if the DNN provides overconfident logits with 100% accuracy then the BNN will learn a posterior distribution for this set, instead of a validation set that would be representative of the miscalibration.

However, we found in a pilot study that training the BNN with the training and validation dataset was (surprisingly) equivalent. We didn't put too much effort into analyzing these kind of things, and hence I pursued this analysis, beyond other facts, with a Master student (Alvarez-Balanya 2020). We are now extending it and willing to publish it soon.

One consequence of training the BNN using the training set is that that the BNN will be usually biased towards providing overconfident predictions since the logits provided by the DNN on the training set have 100% accuracy with 1.0 confidence. To this end, and building upon the work from (Knoblauch et al. 2019) we introduce a parameter β that weights the influence of the KLD and the Expected Log-Likelihood (ELL) in the training objective:

$$\text{ELBO} = \int q_\phi(\theta) \log p(\mathbf{X}, \mathbf{Y}|\theta) \mathrm{d}\theta - \beta \, \text{KLD}[q_\phi(\theta)||p(\theta)] \tag{4.8}$$

Note that since the KLD is the uncertainty quantifier, we can use a value of $\beta \gg 1$ to *force* the approximate posterior to quantify more uncertainty from the prior. In this way the initial approximate posterior that is concentrated around weights that map overconfident logits to overconfident logits is forced to quantify more uncertainty around this set of parameters, i.e. is forced to consider other weights beyond the MAP solution. Our experiments and the work from (Alvarez-Balanya 2020) shows that the value of β is generally greater than 1, and that helps to improve the calibration of the model.

However, this β is related to Bayesian Inference using a power likelihood: $p(\mathbf{Y}|\mathbf{X}, \theta)^{\beta}$, which in turns can be used to solve, for example, model misspecification (Holmes et al. 2017). One clear consequence of raising the value of β is that the model will be more sensible to prior misspecification (Knoblauch et al. 2019). We build on this observation further in the chapter.

Variance Under Estimation

Before diving into the experiments that demonstrate how BNN can calibrate the output of a DNN, we describe one of the well-known issues of variational inference since we will use it as a possible explanation of some of the performance issues of BNN in this particular task.

Variational inference suffers from posterior variance underestimation (VUE), see again (Bishop 2006, p. 469). This makes the resulting variational distribution $q_{\phi}(\theta)$ avoid placing high density over regions where the posterior presents low density. Or, in other words, if the posterior is highly multimodal and the variational distribution is unimodal, then the resulting variational distribution will tend to cover only one mode from the true posterior. This effect is also known as mode collapse.

In practice, we hypothesize that this effect can sometimes affect the performance of the proposed approach in two ways. On one side, consider, for example, a highly multimodal intractable posterior that presents a single high-density mode, alongside with different bumps over the parameter space. As a result of the optimization process, if the variational distribution accounts for this high mode, the set of weights sampled could resemble those of MAP estimation, and thus we will be providing over-confidence predictions. A similar effect could happen if the posterior distribution is unimodal but it has a funnel shape with heavy-tailed marginals, see (Carpenter et al. 2017) user guide section 21.7. In this case, the variational posterior would concentrate around the mode since accounting for the tails (in the case of a Gaussian vari-

ational posterior) would imply placing density over regions with 0 posterior mass. To overcome this limitation, and beyond the introduction of β, we propose to select the optimal value of K in Equation 4.2 on a validation set. While this approach contrasts with the theory, which states that K should tend to infinity, we find it an effective solution to overcome this limitation in our experiments for this particular mean-field approach. We illustrate this later in the experiment section.

On the other hand, if our intractable posterior presents several bumps with equal probable density, or our approximate distribution accounts for a non-highly probable mode of the intractable posterior, the set of weights sampled could not be enough representative of the data distribution. The confidences assigned by models parameterized with this set of sampled weights could affect the accuracy and the calibration error. This can only be solved by using more sophisticated approximations of the variational distribution as the current approach can only recover unimodal Gaussian distributions. We realized that this effect only affects the most complex tasks. For complexity, we refer, on one side, to the particular task to solve (which will mainly depend on the number of classes and number of samples) and, on the other to how well the variational distribution is able to fit the intractable posterior (i.e. how unimodal the shape of the posterior is).

Note that since the posterior depends on the choice of the likelihood, the prior and the set of observations, the issues described above are task-specific and will depend on the number of classes, the representations learned by the DNN, the topology of the DNN and the number of training points. As we will see in the experiment section, these issues are just present in some of the experiments, and these just serve as a hypothesis on what could be happening in order to provide some light into future research directions.

4.2.3 Chapter Summary

Before presenting the experiments, I expose a brief backup of what we have seen so far and which elements are involved in the experiments.

One of the things that we do not experiment with in the following section is the use of MCMC algorithms to (better) characterize the true posterior[3]., and hence validate the possible conclusions. The reason was simple: I heard and learn about Hamiltonian Monte Carlo (HMC) nearly before submitting

[3]As I already note in this chapter, even using MCMC will not probably characterize the true posterior since diagnosing convergence in Bayesian Neural Networks is hard due to unidentifiability and multimodality of the true posterior.

this work for publication. Moreover, how to diagnose HMC, the NUTS HMC variant, or the Stan software which implements many useful algorithms to run NUTS (such as windowed warm-up), were things I heard about after, and hence I could not include them in the experiments presented below.

The other important thing not presented in the experiments is the use of alternative divergences to the KLD as uncertainty quantifiers. In the original publication in (Maroñas et al. 2020), I used subsubsection 4.2.2 to explain some of the issues we will see this approach can suffer from. After the publication of this work, I read the work from (Knoblauch et al. 2019). This work was an inspiration to understand, explain and solve these possible issues, and also to have further understanding of this approach.

Of course, both the Generalized Variational Inference and the use of MCMC algorithms form the base of the future work that my MsC student started in his Master Thesis (where he analyzed the influence of alternative divergences), and that is continuing at the moment. Why I did not do this and I do not present it in this chapter?. The answer is simple: I was already involved in the project that led to the next chapter of this thesis.

In summary, we focus this work from the perspective of performing a big set of experiments to validate if a standard mean-field variational inference BNN could serve for this task and analyze some general aspects of its performance. Right after we are performing a more detailed analysis of what is going on which will be hopefully available by the end of this current year.

4.3 Experiments

We conduct several experiments to illustrate the different properties of the proposed approach. We provide code for reproducibility and supplementary material for details on different specific results in Github[4].

4.3.1 Experiments set up

Datasets: We choose datasets with a different number of classes and sizes to analyze the influence of the complexity of the calibration space and the robustness of the model. In parenthesis, we provide the number of classes: Caltech-BIRDS (200)(Welinder et al. 2010), Standford-CARS (196)(Krause et al. 2013), CIFAR100 (100)(Krizhevsky et al. 2009b), CIFAR10 (10)(Krizhevsky

[4]https://github.com/jmaronas/DecoupledBayesianCalibration.pytorch

et al. 2009a), SVHN (10)(Netzer et al. 2011), VGGFACE2 (2)(Cao et al. 2018), and ADIENCE (2)(Eidinger et al. 2014). We use all the training set to train the Bayesian models except for VGGFACE, where we use a random subset of 200000 samples, which is 15 times fewer than the original. This was enough to outperform the state-of-the-art.

Models: We evaluate our model on several state-of-the-art configurations of computer vision Neural Networks, over the mentioned datasets: VGG (Simonyan et al. 2015), Residual Networks (He et al. 2016a), Wide Residual Networks (Zagoruyko et al. 2016), Pre-Activation Residual Networks (He et al. 2016b), Densely Connected Neural Networks (Huang et al. 2017), Dual Path Networks (Chen et al. 2017), ResNext (Xie et al. 2017) , MobileNet (Sandler et al. 2018) and SeNet (Hu et al. 2018).

Performance metrics: To evaluate the performance of the proposed approach we report values for accuracy (ACC) and Expected Calibration Error (ECE) computed with 15 bins.

BNN Training specifications: We optimize the ELBO using Adam optimization (Kingma et al. 2015a) as it performed better than Stochastic Gradient Descent (SGD) in a pilot study, and we select β from the set $\{10^{-i}\}_{i=0}^{4}$, depending on the BNN architecture. We use a batch size of 100 and both step and linear learning rate annealing. More details are provided in the supplementary material in Github.

Calibration Techniques: We evaluate our model against recently proposed calibration techniques. Regarding explicit techniques, we compare against Temperature Scaling (TS) (Guo et al. 2017) as to our knowledge is the state-of-the-art in decoupled calibration techniques. We also compare with a modified version of Network Ensembles (NE) (Lakshminarayanan et al. 2017). This is an implicit calibration technique that proposes to average the output of several DNN with adversarial noise (Szegedy et al. 2014) regularization, different random initialization and randomized training batches. Due to the high computation cost, we train decoupled NE, i.e, NE that maps the logit from the DNN.

On the other hand, regarding implicit calibration techniques, we compare against NE in their original form; against MMCE (Kumar et al. 2018), which proposes a calibration cost which is computed using kernels; and with Monte Carlo Dropout (Gal et al. 2016), that averages several stochastic forward passes through a Neural Network trained with Dropout (Srivastava et al. 2014).

4.3.2 Bayesian vs Non-Bayesian Linear Regression

In this section, we compare Bayesian and non-Bayesian Linear Logistic Regression under the proposed framework. With this, we show the benefits of being Bayesian in this setting, which let us conclude, in contrast to (Guo et al. 2017), that the source of miscalibration is not that the calibration space is simple, but the proper addressing of uncertainty in the model parameters.

To do this, we train several DNN on different datasets and then use a Linear Logistic model with a Bayesian and a Non-Bayesian approximation. In this setting, the likelihood is given by:

$$p(\mathbf{Y}|\mathbf{X}, \theta) = f(\mathbf{X}^T \cdot \mathbf{W} + \mathbf{b}), \tag{4.9}$$

where $\mathbf{W}$ and $\mathbf{b}$ are parameters, $f()$ is the softmax function and $\mathbf{X}$ represents the logit computed from the DNN.

Table 4.1 shows a comparison of both methods where it is clear that the Bayesian model provides better performance both in accuracy and calibration. It should be noted that the solution of this optimization problem under the non-Bayesian estimation is unique[5], while the MFVILR admits several steps of improvement just by using a more sophisticated approximated distribution, that could capture non-Gaussian or multimodal posteriors. Thus, it is clear that our main claim, combining the powerfulness of DNN and BNN can be achieved.

Table 4.1: Calibration ECE (%), and accuracy (ACC) (%) performance for averages of several logistic models trained for three of the databases considered in this work. ACC the higher the better, ECE the lower the better.

	CIFAR100		**SVHN**		**CARS**	
	ECE	ACC	ECE	ACC	ECE	ACC
Point Estimate	33.90	62.67	1.13	96.72	23.50	**76.14**
Bayesian	**3.66**	**72.36**	**1.03**	**96.72**	**1.88**	74.31

[5]The model is non-identifiable but several parameters provide the same predictive model. This is easy to see since the softmax is invariant under a shift, but the problem itself is convex.

4.3.3 Selecting optimal K on validation

We then illustrate why selecting the optimal value of the number of Monte Carlo predictive samples with a validation set is necessary. One of the problems of VUE is that we can fit our approximation to a high-probable mode of the intractable posterior density, sampling set of weights that could resemble those of MAP estimation, with overconfident probability estimates as a result. In this work we show that this effect can be controlled by searching for the optimal value of Monte Carlo predictive samples, K in Equation 4.2, using a validation set.

As an illustration of this over-sampling effect, **??** shows the calibration error when increasing the number of MC samples. By looking at the figure in the middle and the left we can see how the calibration error is kept constant (or even increased) when more samples are drawn. This suggests that the variational distribution is coupled to a particular part of the intractable posterior. As a consequence, the ultimate confidence assigned by the model is not being consistent with the ideal estimation. In the case of being coupled to high probability regions of the intractable posterior, the generated samples could resemble those of MAP estimation, having overconfident predictions as a consequence, which links with the observations provided by (Guo et al. 2017) in which complex models provide overconfident predictions. However, this effect can be more or less present, as seen for instance in the right figure, where the behavior resembles what one should expect, i.e. non-degraded performance when increasing the number of MC samples. However, even without selecting for the optimal value of K on validation, we observed that most of the models outperformed the baseline uncalibrated DNN and provide competitive or even better results than the state-of-the-art as K increases.

4.3.4 Calibration performance of BNN

In this subsection, we discuss the calibration performance of the proposed framework. We start by evaluating the proposed method against a baseline uncalibrated network in several datasets. Results are shown in Table 4.2, where we compare the results with MFVILR and MFVI. For VGGFACE2 we only run the experiments with MFVILR due to computational restrictions. These average results are computed from more than 40 trained DNN across different datasets, showing the suitability of the proposed approach

As shown in the table, the proposed technique improves the calibration performance by a wide margin over the baseline even though we are using a mean-

field approximation to the intractable posterior distribution with well-known established limitations. Regarding the accuracy performance, we see a slight accuracy degradation which is only relevant in highly complex tasks, such as CIFAR100, BIRDS and CARS. We hypothesise that this degradation is not due to a limitation of the BNN algorithm, but due to inaccurate approximations to the true posterior in some settings. In fact, in some cases, we improve the accuracy over the baseline, as in the two-class problem. This degradation can also give us further insight into the complexity of the calibration task.

As we stated, accuracy degradation can be explained by mode collapse. To illustrate this claim, we compare the performance provided by MFVI and MFVILR, as both these approximations only differ in the convergence of the training criteria given by Equation 4.5, i.e. both approximations provide factorized Gaussian approximations $q_\phi(\theta)$ as approximate distributions. As shown in the table, better results were obtained by the MFVILR, both regarding calibration and accuracy performance, which means that an inaccurate approximation to the true posterior is responsible for this degradation. This is justified by the fact that, as the MFVILR provides better speed convergence, we are able to fit a better approximation to the intractable posterior. This same effect is showed when one trains the same DNN using SGD and SGD with momentum. Even the models and the initialization can be the same, the results provided by SGD with momentum are better due to improved convergence.

On the other hand, as we see from the results, this degradation is noticeable in more complex tasks. This suggests that the complexity of the intractable posterior increases with the complexity of the task, and thus, a mean-field approximation is not able to provide the same performance as it does in sim-

Table 4.2: Average ECE 15(%) and ACC (%) on the test set comparing the uncalibrated model, and the model calibrated with MFVI and MFVILR for each database. ECE the lower the better, ACC the higher the better. "degr" means degraded

	uncalibrated		MFVI		MFVILR	
	ACC	ECE	ACC	ECE	ACC	ECE
CIFAR10	94.81	3.19	94.70	0.58	94.64	**0.50**
SVHN	96.59	1.35	96.50	0.87	96.55	**0.85**
CIFAR100	76.36	11.39	73.87	**2.52**	74.44	**2.52**
VGGFACE2	96.19	1.33	-	-	96.20	**0.37**
ADIENCE	94.25	4.55	94.28	0.53	94.27	**0.51**
BIRDS	76.27	13.22	degr	degr	74.32	**1.88**
CARS	88.79	5.81	degr	degr	85.34	**1.59**

pler ones. This follows our claim that complex techniques overfit due to a bad uncertainty treatment and not because the calibration space is inherently simple, as noted in (Guo et al. 2017). To provide further insight, Table 4.3 compares MFVI and MFVILR with different models and CIFAR100. The first two rows of the table show how the accuracy degradation is clearly improved just by using MFVILR, which is a general tendency in the experiments (see the supplementary material). However, one can not expect that using MFVILR should always achieve better results, as a good convergence of MFVI should make us recover similar approximate posteriors, reflected as no performance increases. This is shown in the third and fourth rows. Moreover, if the approximate posterior is a bad approximation to the true posterior, we can dig into an undesirable local minimum, as shown in the fifth and sixth rows. We found that models where MFVILR worsened the performance w.r.t. MFVI were those more difficult to calibrate in general, which can be explained by the fact that the complexity of the true posterior cannot be captured by the factorized Gaussian approximation, and more sophisticated approximations need to be employed.

On the other hand, we can also provide evidence on the complexity of the calibration space as being dependent on the complexity of the task by analyzing another effect observed in the experiments carried out. Again, and only in complex tasks: CIFAR100, BIRDS and CARS, we experimented an accuracy degradation during training with the MFVI. This means that even although the ELBO was correctly maximized, i.e. the likelihood correctly increases over the course of learning, the accuracy provided was totally degraded. In CIFAR100 we solve it by progressively increasing the expressiveness of the likelihood model for the MFVI, as illustrated in the supplementary material. However, on BIRDS and CARS it could only be solved when using MFVILR,

Table 4.3: MFVI compared to MFVILR in CIFAR100. * means best model on validation

	CIFAR100			
	MFVI		MFVILR	
	ACC	ECE	ACC	ECE
DenseNet 169	75.58	2.39	77.22*	2.45
ResNet 101	68.59	1.61	70.31*	1.75
Wide ResNet 40x10	76.17	1.88	76.51*	1.79
Preactivation ResNet 18	74.30	1.76	74.51*	1.59
Preactivation ResNet 164	70.77*	1.46	71.16	2.20
ResNext 29_8x16	73.97*	2.58	71.13	3.77

as shown in Table 4.2 where "degr" stands for degradation, and it refers to this effect. This suggests that the factorized Gaussian is unable to give a reasonable approximation to the intractable posterior under noisier gradients. As this effect is only present in a more complex task, this again suggests that when the complexity of the task increases, so does the calibration space.

On the other hand and based on the previous observation, one could argue that accuracy degradation is due to a lack of expressiveness in the likelihood model. However, we still emphasize that VUE is responsible for this effect. This is because first increasing the expressiveness of the likelihood model in MFVI on BIRDS and CARS did not solve the problem. Second is because we observed that by using MFVILR we were able to reduce, in general, the topologies, of the likelihood model as compared with MFVI. This is illustrated in Table 4.4 where we show a comparison between the average number of parameters used for each task[6].

To end with, we surprisingly found that in some models that achieved good calibration and accuracy properties, both the negative-log-likelihood and the accuracy increased over the course of learning. This means that the network is unable to correctly raise the probability toward the correct class for the miss-classified samples.

4.3.5 Comparison Against state-of-the-art calibration techniques

We then compare the calibration performance of our method against other proposed techniques for calibration, both implicit and explicit. For the comparison, we use the hyperparameters as provided in the original works. Results are shown in Table 4.5 for explicit methods and in Table 4.6 for implicit methods. Results on the same dataset might differ as due to the high computational cost of some of the explicit calibration techniques, we only perform a subset

Table 4.4: Average number of parameters (in thousands).

	MFVI	MFVILR
CIFAR100	24018.7	430.5
CIFAR10	696.6	65.6
SVHN	606.9	7.6
ADIENCE	0.470	4.482
average	6331.2	126.1

[6]In ADIENCE MFVILR was not able to reduce the topologies due to instabilities when computing derivatives. We provide a justification in the supplementary material in Github.

of the experiments. Details on the models used to compute these results are provided in the supplementary material in Github.

Explicit calibration techniques

Comparing against explicit calibration techniques we first see that all the methods increase the calibration performance over the baseline (see Table 4.2), with a clear improvement of the BNN over the rest in all the tasks. These results demonstrate the two main hypotheses of this work: Bayesian statistics provide more reliable probabilities, and complex models improve calibration over simple ones. This observation is consistent in all the experiments presented, where the ECE is the lowest for the proposed model, manifesting the robustness of the BNN approach in terms of calibration. Therefore, our results support the hypothesis that point-estimate complex approaches for re-calibration overfit (Guo et al. 2017) because uncertainty is not incorporated and not because calibration is inherently a simple task. This conclusion can also be supported by the fact that as the complexity of the task increases, the number of parameters of the Bayesian model that yields better results also increases. For instance, the calibration BNN for CIFAR100 needs many more parameters than the BNN for simpler tasks such as CIFAR10, as shown in Table 4.4. Second, it is important to remark that in some models TS has degraded calibration by a factor of three in the worst case while BNN do not, as seen in the results provided in the supplementary material. On the other hand, Bayesian model average clearly outperforms standard model averaging as performed by NE. In fact, NE are not suitable for the calibration of deep models, because training directly an ensemble of DNN is computationally hard and training NE over the logit space does not perform as well as TS. In addition, NE is the one that uses more parameters.

All these observations manifest the suitability of the proposed decoupled Bayesian stage for recalibration, as even a mean-field approximation to the intractable posterior performs better in terms of calibration than the state-of-the-art in many scenarios. This motivates future work to study more complex variational

Table 4.5: Average ECE results compared against explicit calibration techniques.

	CIFAR10	CIFAR100	SVHN	BIRDS	CARS	VGGFACE2	ADIENCE
NE decoupled	2.55	10.17	1.02	5.25	5.51	0.79	2.64
TS (Guo et al. 2017)	0.90	3.29	1.04	2.41	1.80	0.55	0.87
ours	**0.50**	**2.52**	**0.85**	**1.88**	**1.59**	**0.37**	**0.51**

approximations and different Bayesian-based stages, in order to mitigate the accuracy degradation observed in these experiments.

To end with, one important aspect we observed is the robustness of BNN. We obtained a calibration improvement over TS on the first hyperparameter search in many of the experiments performed. Only some exceptions require further hyperparameter search, which is explained by having to approximate more complex posterior distributions. However, in general, the mean-field approach provides good results, as illustrated in Figure 4.4, where we show how many of the tested configurations outperformed TS. More figures are provided in the supplementary material.

Implicit calibration techniques

We then compare against implicit calibration techniques. Looking at the results in Table 4.6 we see that Network Ensembles provide competitive results but at a higher computational cost. This is because this method requires to train several DNN to search for the optimal parameters (number of ensembles, the factor of adversarial noise, topologies of the ensembles...), while we only require to reach good discrimination as provided by the DNN, and then search hyperparameters on a much lighter model.

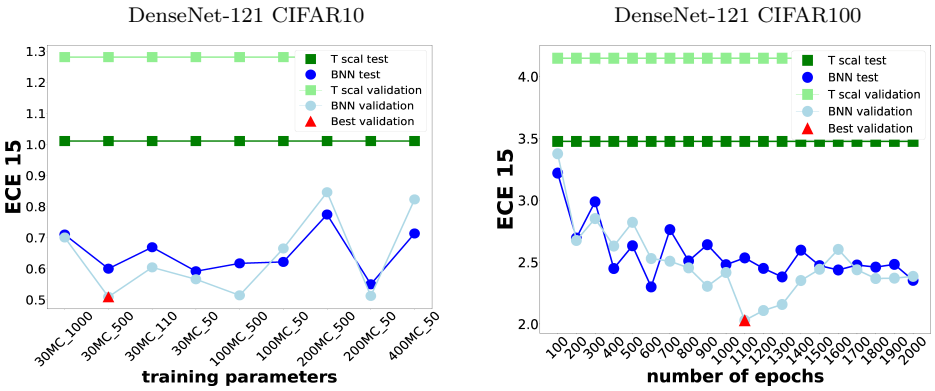


Figure 4.4: Comparison of ECE performance between TS and BNN in test and validation. On the left (CIFAR10) we show the performance of models trained with different parameters. As an example, 30MC 500 means that the ELBO was optimized using 30 MC samples to estimate expectation under $q_\phi(\theta)$ and 500 epochs of Adam optimization. On the right (CIFAR100) we show the performance of a BNN trained with a different number of epochs up to 2000, showing the performance against the course of learning.

On the other hand, we briefly discuss other potential advantages of our method against implicit techniques. First, we see how our Bayesian method outperforms the other Bayesian method provided, named Monte Carlo dropout (MCDROP). We should expect these results as the main authors clearly state in their work that the probabilities provided by this method should not be necessarily calibrated as the dropout parameter has to be adapted as a variational parameter depending on the data at hand (Gal et al. 2017). In fact, many works that aim at reporting that Bayesian methods do not provide calibrated outputs (Lakshminarayanan et al. 2017; Kuleshov et al. 2018) only provide results comparing with MCDROP. However, this work has clearly shown that Bayesian methods are able to improve the calibration performance over point estimate techniques.

Moreover, while our method does not compromise the previous DNN architecture, both MCDROP and VWCI require sampling-based stages, e.g dropout, to be applied to the DNN. Despite the improvement of (Seo et al. 2019) over a baseline uncalibrated model, our method is clearly better, as shown in the table. Moreover, it seems unclear how scalable this method is when applied to Deep Learning models, as to compute the cost function, this approach requires several forwards through the DNN. While their deeper model is a DenseNet-40 we provide results here for a DenseNet-169. On the other hand, our method is clearly more efficient than MCDROP or other Bayesian implicit methods as these requires performing several forwards through the DNN.

In addition, developing techniques to recalibrate the outputs of a model is indeed interesting, as they can be combined with implicit techniques. As an example, the best results reported by (Kumar et al. 2018) are a combination with their method with TS. Furthermore, (Lee et al. 2018) also uses TS as

Table 4.6: Average ECE results compared against implicit calibration techniques. * indicates that the results are taken from the original works. We also include TS. Results from TS and our approach differ from Table 4.5 as we only pick the DNN used in the explicit techniques.

	CIFAR10	CIFAR100	SVHN
VWCI (Seo et al. 2019)*	-	4.90	-
MMCE (Kumar et al. 2018)	1.79	6.72	1.12
TS (Guo et al. 2017)	0.82	3.84	1.11
MCDROP (Gal et al. 2016)	1.38	3.49	0.92
NE (Lakshminarayanan et al. 2017)	0.61	3.27	**0.71**
ours	**0.43**	**2.28**	0.83

the calibration technique, and (Kuleshov et al. 2018) proposes a method for re-calibrating outputs in regression problems; which manifest the interest and power of developing techniques that aim at re-calibrating outputs of a model.

We end up this subsection by discussing the accuracy performance of the explicit and implicit techniques. On the explicit calibration techniques, we know that Temperature Scaling does not change the accuracy of the DNN and we found that decoupled Network Ensembles didn't consistently improve or degrade it. Regarding the implicit calibration techniques, Network ensembles usually provide $2-3$ points of boost in accuracy w.r.t. the single network, but at a much higher computational cost. MCDROP or VWCI does not present a clear boost or accuracy degradation, while MMCE sometimes degrade accuracy as we illustrated in the previous chapter.

4.3.6 Qualitative Analysis

We have also performed a qualitative analysis of the output of the Bayesian model in comparison with TS. We realized that on the misclassified samples made by TS and BNN, the BNN assigns lower confidence than TS, which is a desirable property. On the other hand, regarding the correctly classified samples, the BNN not only adjusts the confidence better but also classifies these samples with higher confidence than TS. This may mean that TS calibrates by pushing samples to lower confidence regions, an observation that has been also noted in previous works (Kumar et al. 2018). Moreover, we analyzed the samples where the BNN decided a different class w.r.t. the DNN. On the one hand, we analyzed the set of these samples where the class assigned by the BNN was correct, i.e. 100% accuracy. First, in this set, the original decision made by the DNN was incorrect, i.e. 0% accuracy. Second, the DNN assigned very high incorrect confidence (over 0.9) to some of these miss-classified samples. Third, the new confidence assigned by the BNN was not extreme, which means that the BNN "carefully" changes the decision made by the DNN. On the other hand, we analyze the set of samples where the BNN assigned a different class from the DNN, and this newly assigned class was incorrect. First, we realize that the DNN only had a 50% of accuracy on this set. Second, the original confidence assigned by the DNN to these samples was below 0.5. This means that the BNN does not make wrong decisions on a set of high-confidence, well-classified samples by the DNN.

4.4 Discussion

The disadvantages discussed in subsubsection 4.2.2 are not a limitation of our approach. We can still improve the approximate posterior by applying normalizing flows (Rezende et al. 2015), auxiliary variables (Agakov et al. 2004), combinations of all of them (Louizos et al. 2017b) etc. Finally, a potential line of research considers robustification by means of Generalized Variational Inference (Knoblauch et al. 2019), and the use of MCMC algorithm to decouple the effect of the approximate inference algorithm in the results. However, including all these improvements is not the aim of this work, but to show the adequacy of the proposed decoupled BNN and its potential for future improvements. This is because the true posterior distribution can be highly variable, as it not only depends on the parameterization of the likelihood model and the prior but also on the observed dataset, which itself depends on the input training distribution and the set of representations learned by the specific DNN. Thus we decided to validate our proposal restricting ourselves to the Gaussian approximation and to show it works in a numerous set of different configurations.

4.5 Conclusions and Future Work

This work has shown that Bayesian Neural Networks with mean-field variational approximations can robustly provide state-of-the-art calibration performance in Deep Learning frameworks, overcoming the limitations of applying Bayesian techniques directly to them. This suggests that using more sophisticated approximations to the intractable posterior should even yield better results than the ones reported in this work.

We have also shown that as long as uncertainty is properly addressed we can make use of complex models that do not overfit, showing that probability assignments of DNN outputs suppose a more complex task than what previous works argued. Also, we have shown that, in contrast to previous works, Bayesian models parameterized with Neural Networks can be successfully used for the task of calibration. Moreover, our approach is a clear alternative to the development of Bayesian techniques directly applied to DNN, as we do it at a much lower computational cost.

On the other hand, we have analyzed and justified the drawbacks found in this work: slight accuracy degradation in complex tasks and the selection of the number of Monte Carlo predictive samples using a validation set. Future work will be focused on the exploration and analysis of different Bayesian

models for the task of calibration, different approximations to the intractable posterior distribution, the use of MCMC algorithms to decouple the influence of the approximate inference algorithm, and the use of alternative divergences to the KLD to account for prior misspecification. With all this, we aim at reducing and deeply analyze the influence of the aforementioned drawbacks.

Chapter 5

Transformed Gaussian Process as a new prior over functions

This last chapter of the thesis introduces the Transformed Gaussian Process. It is a new prior over functions that is constructed by transforming samples from a Gaussian process using an invertible transformation. The chapter first introduces Gaussian Processes and the variational inducing points approximation, which serves as the base for introducing the proposed model. Among other nice properties, we show that this model provides better calibration properties than the sparse variational Gaussian Process and similar performance to a Deep Gaussian Process at a fraction of the computational cost. The ideas presented in this chapter correspond to those described in (Maroñas et al. 2021b).

In the previous chapter, we briefly discuss the importance of specifying a suitable prior to conduct meaningful Bayesian inferences. We saw that in a Bayesian Neural Network we usually specify priors over the parameters for computational convenience since that allows us to compute the Kullback Leibler Divergence between the prior and the variational posterior in the ELBO in closed form, which drastically speeds up the convergence and stability of the optimization process.

However, since the computational graphs of Neural Networks are black-box function approximators, it is difficult to interpret the prior over functions that the prior over the parameters, the computational graph, and the data distribu-

tion $p(\mathbf{X})$ induce, which is our ultimate object of interest. This means that a prior over the parameters that is suitable for computational convenience might not induce a suitable prior over functions, and the problem is that due to its black-box nature, it is hard to characterize whether the prior over functions is correctly specified for the application at hand. In fact, one of the lines of improvement of the usage of Bayesian Neural Networks for decoupled calibration is precisely the robustification against prior misspecification.

Gaussian Processes (GP) are (for the moment) mathematical objects that allow us to specify a prior distribution directly over the space of functions, where the attributes of the functions are specified via very few interpretable parameters. This means that in contrast to Neural Networks, we can easily specify and interpret priors and thus avoid biasing inferences due to prior misspecification.

The parameters of GP allow us to easily define functions that are periodic, that are smooth[1], that present some linear tendency etc. Thus in practice GP allow us to easily encode inductive biases in the space of possible functions if we have background knowledge about our task at hand. Of course, GP have their limitations as any machine learning model and, as always, the application will usually drive the kind of machine learning model we want to deploy.

We start this chapter by introducing standard GP regression, their limitations, and how to overcome them.

5.1 Standard Gaussian Process

5.1.1 Introduction

A Gaussian Process is a stochastic process (i.e. a family of random variables indexed by some mathematical set that is classically associated with time) such that any finite collection have a joint Multivariate Normal distribution. The mean and covariance of this Multivariate Normal distribution are given by the mean and covariance functions evaluated at the index set of the stochastic process. In our particular problem, this index set corresponds to the features $\mathbf{X}$.

An example of a sample drawn from a Gaussian process is given Figure 5.1. This figure illustrates the difference between GP with different covariance functions and different parameterization. The top row GP have a covariance func-

[1] Infinetly differentiable.

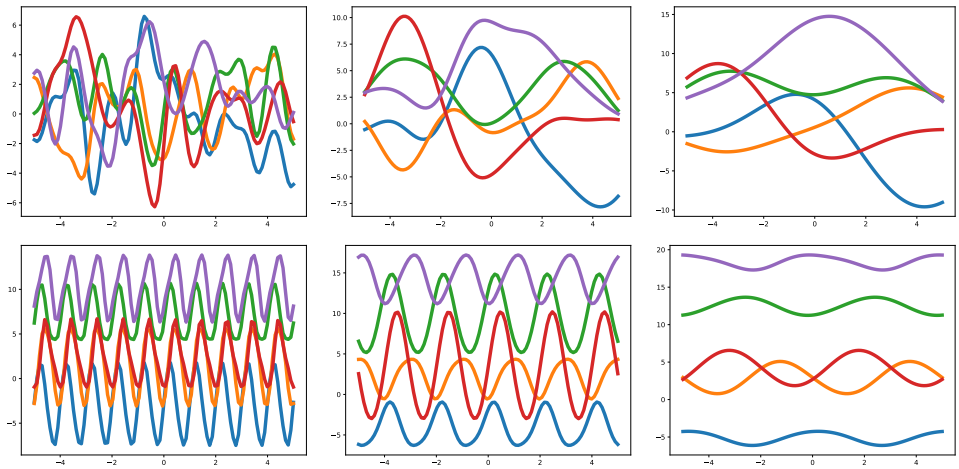


Figure 5.1: This figure illustrates five samples drawn from a zero mean Gaussian Process with different covariance functions with varying hyperparameters. Top row shows samples from an RBF kernel with parameter given by a value of 0.3, 1.0 and 2.0 (left to right) and bottom row shows a periodic kernel with lengthscale given by 1.0 and periodic parameter given by 1.0, 2.0 and 5.0.

tion given by a RBF kernel while the bottom row has a periodic one. Samples drawn from a RBF kernel are smooth, with the parameter controlling the length of the wiggles in the functions. Samples from a GP with RBF kernel are equivalent to samples from a linear regression model with a Gaussian prior on the parameters and an infinite number of basis functions $\phi(\mathbf{X})$. The RBF kernel is usually the *standard* kernel when no additional information about the data (periodicity, tendency etc.) is provided. On the other hand, the bottom row shows samples from a GP with a periodic kernel. The parameter from this kernel controls the period of the function, which can range from high (left) to low (right) frequencies. For an extensive study and explanation on different kernels and how can we combine them to create functions with more attributes see (Rasmussen et al. 2005; Duvenaud 2014).

GP can be seen as a distribution over functions since the family of random variables can be arbitrarily large and hence we can see each of these random variables as the function evaluation $f(\mathbf{X})$ at each point $\mathbf{X}$. Moreover, since the mean and covariance specifies the kind of functions we can sample, through few interpretable parameters, it is easy to specify meaningful priors if one wishes to use the GP as the prior over the functions in a Bayesian setting. This contrasts with Neural Networks where, as we saw, specifying meaningful priors is hard.

For example, if we know that our data has some periodic behaviour, we can encode this knowledge using a periodic kernel.

5.1.2 Bayesian predictions using GP

For our interests, the role of the GP is to encode the degree of belief about a set of functions being responsible of generating some data. With $\mathbf{f}$ we will denote samples drawn from a GP prior:

$$\mathbf{f} \sim \text{GP}(\mathbf{f}|m(\mathbf{X}), K_\nu(\mathbf{X}, \mathbf{X})), \tag{5.1}$$

which is linked to the observations $\mathbf{Y}$ through some likelihood function, which we will assume factorizes over data points: $\prod_{n=1}^{N} p(\mathbf{Y}_n|\mathbf{f}_n)$. For the moment and for explanatory purposes we will assume that the likelihood is Gaussian: $p(\mathbf{Y}_n|\mathbf{f}_n) = \mathcal{N}(\mathbf{Y}_n|\mathbf{f}_n, \sigma^2)$, with an homoscedastic variance (shared variance between all datapoints).

Since the likelihood and the prior are both Gaussian distributions, the posterior is also Gaussian. We can derive the expression of the posterior distribution starting from the joint distribution, and then conditioning on one of the observations since the Gaussian distribution is closed under marginalization and conditioning. Most of the expressions (if not all) used in this chapter can be derived using chapter 2 from (Bishop 2006), more precisely those on page 87 and page 93. The posterior distribution of the latent functions $\mathbf{f}$ given $\mathbf{X}$ at any other location $\mathbf{X}^*$ is given by:

$$\begin{aligned} p(\mathbf{f}^*|\mathbf{X}, \mathbf{Y}, \mathbf{X}^*) = \mathcal{N}(\mathbf{f}^*|K_{\mathbf{X}^*,\mathbf{X}} \left[K_{\mathbf{X},\mathbf{X}} + \sigma^2 I\right]^{-1} \mathbf{Y}, \\ K_{\mathbf{X}^*,\mathbf{X}^*} - K_{\mathbf{X}^*,\mathbf{X}} \left[K_{\mathbf{X},\mathbf{X}} + \sigma^2 I\right]^{-1} K_{\mathbf{X}^*,\mathbf{X}}^T), \end{aligned} \tag{5.2}$$

where from now on, $\mathbf{Y}$ and $\mathbf{X}$ denotes matrices where each row represents a training point and $K_{\mathbf{X},\mathbf{X}}$ is a matrix where each position corresponds to the kernel evaluation between all the possible pairs of both matrices $\mathbf{X}$. This means that $K_{\mathbf{X},\mathbf{X}}$ is an $N \times N$ matrix. We also drop the explicit notation of the kernel parameters ν^2 from both the kernel and the conditional distributions.

This posterior distribution is a GP (i.e. another distribution over functions) that express which functions are likely to be representative of the new data

[2] We now denote the parameters with ν rather than θ since θ will be used to denote the parameters of the proposed model.

and the kind of attributes encoded in the prior. As always, this posterior distribution is integrated out to obtain the posterior predictive, which is also Gaussian and can be derived using simple integrals from linear Gaussian models:

$$\begin{aligned} p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{Y}, \mathbf{X}) = \int p(\mathbf{Y}^*|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{X}, \mathbf{Y}, \mathbf{X}^*)\mathrm{d}\mathbf{f}^* = \\ \mathcal{N}(\mathbf{Y}^*|K_{\mathbf{X}^*,\mathbf{X}}\left[K_{\mathbf{X},\mathbf{X}} + \sigma^2 I\right]^{-1}\mathbf{Y}, \\ K_{\mathbf{X}^*,\mathbf{X}^*} - K_{\mathbf{X}^*,\mathbf{X}}\left[K_{\mathbf{X},\mathbf{X}} + \sigma^2 I\right]^{-1}K_{\mathbf{X}^*,\mathbf{X}}^T + \sigma^2 I) \end{aligned} \tag{5.3}$$

5.1.3 Bayesian Model Selection

In the second chapter of this thesis, we talk about the benefits of the marginal likelihood for model selection. One of the appealing properties of Gaussian Processes with a Gaussian Likelihood, in contrast to Neural Networks, is that we can compute the log marginal likelihood given the hyperparameters θ, which is a quantity that is usually intractable in Neural Networks since we cannot integrate out the parameters $\mathbf{W}, \mathbf{b}$ beyond simple cases. In GP however, we have:

$$\begin{aligned} p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})\mathrm{d}\mathbf{f} = \\ \int \prod_{n=1}^{N} \mathcal{N}(\mathbf{Y}_n|\mathbf{f}_n, \sigma^2)\mathcal{N}(\mathbf{f}|0, K_{\mathbf{X},\mathbf{X}})\mathrm{d}\mathbf{f} = \\ \mathcal{N}(\mathbf{Y}|0, \sigma^2 I + K_{\mathbf{X},\mathbf{X}}) \end{aligned} \tag{5.4}$$

Thus we can optimize this quantity using gradient descent to select the optimal set of hyperparameters. Of course, this is not the exact marginal likelihood, since the kernel hyperparameters θ cannot be integrated out analytically and we would need to resort to an approximation[3]. Both (MacKay 1992) and (MacKay 2002) provide a wide discussion around the implications of using this pseudo marginal likelihood for model selection. Beyond automatic Occam's Razor, optimizing this quantity to select the optimal θ is a good approximation similar to a fully Bayesian treatment of the hyperparameters.

[3]We can lower bound it using an approximate posterior $q(\theta)$, a process similar to that used for BNN in the previous chapter.

5.1.4 Benefits of Bayesian Learning Using Gaussian Processes

The Bayesian nature of GP (i.e. the fact that we integrate out $\mathbf{f}$ in order to make predictions) and the use of a pseudo-marginal likelihood for model selection and selection of few hyperparameters, make GP more robust to overfitting than Neural Networks, yet being super expressive models (the RBF kernel is equivalent to a linear model with infinity features). Moreover, since they are Bayesian models they properly quantify the uncertainty around a prediction, a property also available in BNN, with the advantage that we don't have to resort to approximate inference algorithms (at least for the moment) that can bias our predictions and we don't suffer from prior misspecification as well.

As a side note it shall be noted that the breakthrough of Neural Networks in machine learning and the fact that is much more useful than GP in many settings is not because Neural Networks are more expressive models, but due to the inductive biases that can be encoded in the Neural Network architecture and that make them more suitable to represent some kinds of distributions such as images.

5.1.5 Drawbacks of Bayesian Learning Using Gaussian Processes

Of course, GP have their limitations. Beyond the unavailability to encode inductive biases in a similar way to Neural Networks, many limitations make GP hard to deploy in practice. The first one is that when the likelihood $p(\mathbf{Y}|\mathbf{X})$ is not Gaussian, none of the expressions derived above can be computed, something that could happen if one wants to use GP for classification. Another drawback of GP is that predictions and model learning require to invert and $N \times N$ matrix, which has a cubic cost. Keeping this matrix in memory has a quadratic cost. This makes GP unfeasible for applications with tons of data, even more model learning since we need a cubic cost operation per gradient update. Note moreover that we cannot use stochastic gradient descent for learning, since Equation 5.4 does not factorize over data, as the GP incorporates correlations between data points once $\mathbf{f}$ is integrated out. Another problem is that GP might lack enough expressiveness due to the use of simple (stationary) kernels or because we assume that the joint distribution in the random process is Gaussian. Also if we want to model positive constrained signals, GP with a Gaussian observation model is clearly misspecified.

5.2 Sparse Gaussian Process

In this section, we introduce Sparse Gaussian Processes as a solution to many of the drawbacks seen so far. This is also a necessary step towards presenting the inference algorithm used in the Transformed Gaussian Process.

The initial idea behind sparse GP is the use of a subset of the data $\mathbf{Z}$ which acts as a summary statistics of the original data $\mathbf{X}$. We will refer to $\mathbf{Z}$ with inducing points or inducing inputs. If we have M inducing points with $M <<< N$, then the $\mathcal{O}(N^3)$ complexity can be reduced to $\mathcal{O}(M^3)$, drastically reducing the computational cost.

Classical approaches to sparse GP can be found in (Quiñonero-Candela et al. 2005). In this thesis, we will focus on the approach from (Titsias 2009), which builds upon the ideas from (Snelson et al. 2006), where $\mathbf{Z}$ are learned through gradient-based optimization, instead of being selected from $\mathbf{X}$ directly. We would however describe the model as in (Hensman et al. 2013) since the ideas presented in this work extend those in (Titsias 2009) so as training can be scaled to large datasets.

Traditional sparse GP approaches rely on selecting these inducing points using a marginal likelihood on a modified GP prior see e.g. (Snelson et al. 2006). This makes the inducing points model parameters and hence these approaches can overfit. The idea of (Titsias 2009) is brilliant and radically different. I try now to briefly expose the key idea behind his approach.

Our ultimate goal is to find a way to compute the posterior distribution $p(\mathbf{f}|\mathbf{X}, \mathbf{Y})$ using less than $\mathcal{O}(N^3)$ operations. An idea would be to define a new distribution $q(\mathbf{f}|\mathbf{Z})$, that depends just on M points. Then, we can optimize the inducing point's locations to minimize a distance between these distributions, for example by using the Kullback Leibler divergence. As we have seen in the previous chapter, this corresponds to variational learning since we want to target the true posterior using an approximate posterior. Note here that the use of a variational distribution is not due to intractabilities when computing the posterior distribution, but as a way of finding an approximation to the true posterior distribution using fewer points.

The problem however is that if our variational distribution is directly defined as $q(\mathbf{f}|\mathbf{Z})$, then there is no way we can yield a training criteria that avoids a $\mathcal{O}(N^3)$ complexity, since one always would have to evaluate the KLD between the prior and this approximate posterior, and that has $\mathcal{O}(N^3)$ operations due to the computation of the determinants need to compute this KLD. (Titsias

2009) proposed to perform variational inference in an augmented space. We can associate each inducing point location $\mathbf{Z}$ with its corresponding function evaluation $\mathbf{u}$. Then, since inducing point locations $\mathbf{Z}$ live in the same space as the data $\mathbf{X}$ we can augment the GP prior to consider these points $p(\mathbf{f}, \mathbf{u}|\mathbf{X}, \mathbf{Z})$. The idea then is to approximate the posterior distribution on this augmented space $p(\mathbf{f}, \mathbf{u}|\mathbf{X}, \mathbf{Y})$ using a variational distribution $q(\mathbf{f}, \mathbf{u}) = p(\mathbf{f}|\mathbf{u})q(\mathbf{u}|\mathbf{m}, \mathbf{S})$, where $\mathbf{m}, \mathbf{S}$ are variational parameters as $\mathbf{Z}$. These distribution take the following form:

$$
\begin{aligned}
p(\mathbf{f}, \mathbf{u}) &= \mathcal{N}\left(\begin{array}{c|cc} \mathbf{f} \\ \mathbf{u} \end{array} \right. \left. \begin{array}{cc} 0, & K_{\mathbf{X},\mathbf{X}} \quad K_{\mathbf{X},\mathbf{Z}} \\ & K_{\mathbf{Z},\mathbf{X}} \quad K_{\mathbf{Z},\mathbf{Z}} \end{array} \right) \\
p(\mathbf{f}|\mathbf{u}) &= \mathcal{N}(\mathbf{f}|K_{\mathbf{X},\mathbf{Z}}K_{\mathbf{Z},\mathbf{Z}}^{-1}\mathbf{u}, K_{\mathbf{X},\mathbf{X}} - K_{\mathbf{X},\mathbf{Z}}K_{\mathbf{Z},\mathbf{Z}}^{-1}K_{\mathbf{X},\mathbf{Z}}^T) \\
q(\mathbf{u}|\mathbf{m}, \mathbf{S}) &= \mathcal{N}(\mathbf{u}|\mathbf{m}, \mathbf{S}); \mathbf{m} \in \mathbb{R}^{M\times 1}, \mathbf{S} \in \mathbb{R}^{M\times M}
\end{aligned}
\tag{5.5}
$$

Thus, the idea behind (Titsias 2009) approach is to target the posterior distribution over the inducing points $p(\mathbf{u}|\mathbf{Y})$ with the variational distribution $q(\mathbf{u}|\mathbf{m}, \mathbf{S})$ and approximate the posterior distribution over the rest of points $\mathbf{f}$ using the conditional's model prior $p(\mathbf{f}|\mathbf{u})$. The fact that $p(\mathbf{f}|\mathbf{u}, \mathbf{Y})$ is approximated with $p(\mathbf{f}|\mathbf{u})$ relies on the assumption that $\mathbf{u}$ are sufficient statistics of the data, hence conditional independence applies on $p(\mathbf{f}|\mathbf{u}, \mathbf{Y})$. This has two consequences, the first one is that the inversion $K_{\mathbf{Z},\mathbf{Z}}$ requires $\mathcal{O}(M^3)$ computations and the second is that the conditional's model prior gets canceled avoiding a $\mathcal{O}(N^3)$ computation when computing the KLD in the Evidence Lower Bound, as we shall briefly see. The training criteria is derived as follows:

$$
\begin{aligned}
&\mathrm{KLD}[q(\mathbf{f}, \mathbf{u})||p(\mathbf{f}, \mathbf{u}|\mathbf{Y})] = \\
&\int q(\mathbf{f}, \mathbf{u}) \log q(\mathbf{f}, \mathbf{u}) \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} - \int q(\mathbf{f}, \mathbf{u}) \log \frac{p(\mathbf{Y}|\mathbf{f}, \mathbf{u})p(\mathbf{f}, \mathbf{u})}{p(\mathbf{Y})} \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} = \\
&\int q(\mathbf{f}, \mathbf{u}) \log \frac{q(\mathbf{f}, \mathbf{u})}{p(\mathbf{f}, \mathbf{u})} \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} - \int q(\mathbf{f}, \mathbf{u}) \log p(\mathbf{Y}|\mathbf{f}, \mathbf{u}) \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} + \log p(\mathbf{Y}) \\
&\int q(\mathbf{u}) \log \frac{\cancel{p(\mathbf{f}|\mathbf{u})}q(\mathbf{u})}{\cancel{p(\mathbf{f}|\mathbf{u})}p(\mathbf{u})} \mathrm{d}\mathbf{u} - \int q(\mathbf{f}) \log p(\mathbf{Y}|\mathbf{f}) \mathrm{d}\mathbf{f} + \log p(\mathbf{Y})
\end{aligned}
\tag{5.6}
$$

which can be rewritten as:

$$\underbrace{\log p(\mathbf{Y})}_{\text{log marginal likelihood}} \quad -\text{KLD}[q(\mathbf{f},\mathbf{u})||p(\mathbf{f},\mathbf{u}|\mathbf{Y})] = \underbrace{\int q(\mathbf{f}) \log p(\mathbf{Y}|\mathbf{f}) \mathrm{d}\mathbf{f} - \text{KLD}[q(\mathbf{u})||p(\mathbf{u})]}_{\text{ELBO}} \tag{5.7}$$

We can see how the conditional's model prior $p(\mathbf{f}|\mathbf{u})$ gets canceled, avoiding the cubic computation involved in the KLD. Also note that since the data $\mathbf{Y}$ only depends on $\mathbf{f}$, the likelihood $p(\mathbf{Y}|\mathbf{f},\mathbf{u}) = p(\mathbf{Y}|\mathbf{f})$. Once this is observed, we just integrate out $\mathbf{f}$ or $\mathbf{u}$ from the integrals where there is no dependency on these terms, leading to the final ELBO. The distribution $q(\mathbf{f})$ is Gaussian and can be easily obtained:

$$\begin{aligned} &q(\mathbf{f}) = \int p(\mathbf{f}|\mathbf{u}) q(\mathbf{u}) \mathrm{d}\mathbf{u} = \\ &\mathcal{N}(\mathbf{f}|K_{\mathbf{X},\mathbf{Z}} K_{\mathbf{Z},\mathbf{Z}}^{-1} \mathbf{m}, K_{\mathbf{X},\mathbf{X}} - K_{\mathbf{X},\mathbf{Z}} K_{\mathbf{Z},\mathbf{Z}}^{-1} \left[K_{\mathbf{Z},\mathbf{Z}} + \mathbf{S}\right] K_{\mathbf{Z},\mathbf{Z}}^{-1} K_{\mathbf{Z},\mathbf{X}}) \end{aligned} \tag{5.8}$$

Finally, since the likelihood factorizes across data points, we can take the sum over data points out of the integral, which allow us to compute unbiased gradients from the training objective allowing to train the model using stochastic gradient guided methods, scaling the method to large datasets:

$$\text{ELBO} = \sum_{n=1}^{N} \int q(\mathbf{f}_n) \log p(\mathbf{Y}_n|\mathbf{f}_n) \mathrm{d}\mathbf{f}_n - \text{KLD}[q(\mathbf{u})||p(\mathbf{u})] \tag{5.9}$$

where we have applied the marginalization property over $q(\mathbf{f})$ so that the integral per datapoint is w.r.t. a univariate Gaussian distribution $q(\mathbf{f}_n)$. This integral can be computed in closed form for certain observation models, or efficiently with one-dimensional quadrature. Predictions can be computed by replacing the posterior with the variational distribution when computing the posterior predictive distribution:

$$\begin{aligned} p(\mathbf{Y}^*|\mathbf{X}^*,\mathbf{X},\mathbf{Y}) = \int p(\mathbf{Y}^*|\mathbf{f}) p(\mathbf{f}|\mathbf{X},\mathbf{Y}) \mathrm{d}\mathbf{f} \approx \\ \int p(\mathbf{Y}^*|\mathbf{f}) q(\mathbf{f}) \mathrm{d}\mathbf{f} \end{aligned} \tag{5.10}$$

The presented derivation overcomes many of the limitations presented so far. The first one is that the computational complexity of the training objective can be reduced from $\mathcal{O}(N^3)$ to $\mathcal{O}(M^3)$. Second, with this approach, we can target the analytically intractable posterior distribution that arises when the likelihood is not conjugate of the GP prior, for example in a classification context. Third, the training objective can be evaluated using a subset of the training set without bias.

However, what happens if a GP with a stationary kernel does not provide an enough expressive model for our application?. In the following section, we introduce the Transformed Gaussian Process (TGP), which is a new prior over functions constructed by transforming samples from a GP yielding a more expressive model.

5.3 Transformed Gaussian Processes

So far we have introduced and motivated Gaussian Processes as prior over functions, and sparse Gaussian Processes as a way to overcome some of their limitations. However, we haven't seen how can we make these models more expressive, for example, if we want to model a non-stationary process with a non-Gaussian joint distribution.

To my knowledge, there are three ways in which we can overcome these limitations. The first one is to use non-stationary covariance functions (Paciorek et al. 2004; Heinonen et al. 2016), however current inference proposals can become hard and slow. Another option is to use Deep Gaussian Processes (Damianou et al. 2013), which are compositions of GP (see Figure 5.2) that allow us to model non-stationary and non-Gaussian processes. Finally, we have

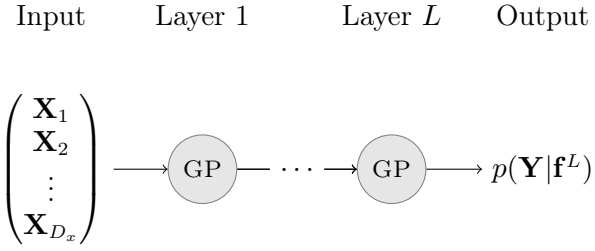


Figure 5.2: Graphical depiction of a Deep Gaussian Process with one GP per layer. The output of one GP is the input to the next GP in the hierarchy.

the Warped Gaussian Process (Snelson et al. 2003), described in section 5.5, which learns an invertible transformation that warps the outputs $\mathbf{Y}$ so that they are well modeled by a GP. Another option is the Bayesian Warped Gaussian Process (Lázaro-Gredilla 2012), which can be seen as a DGP with only one hidden layer.

DGP are certainly a good alternative to create more expressive prior processes. Perhaps their biggest limitation is that inference is hard and slow since one usually uses dozens of GP per layer, but it shall be noted that they work well in practice. In this chapter, we will evaluate our proposed model against DGP with the Double Stochastic Variational Inference (DSVI) algorithm (Salimbeni et al. 2017), which is one of the most popular inference algorithms for these models, and thus a good baseline.

5.3.1 Model Description

We start by describing the proposed TGP model. Given a GP specified via its mean and covariance functions, the TGP is a new process defined by transforming samples from a GP with K invertible parametric transformations $\{\mathbb{G}_{\theta_k}\}_{k=0}^{K-1}$. More precisely, we define for all $k = 0, \dots K-1$ the functions $\mathbb{G}_{\theta_k} : \mathcal{F} \to \mathcal{F}$ as the individual transformations, $\mathbb{G}_\theta = \mathbb{G}_{\theta_0} \circ \mathbb{G}_{\theta_1} \circ \cdots \circ \mathbb{G}_{\theta_{K-1}}$ as their composition and $\theta = \{\theta_0, \theta_1, \dots, \theta_{K-1}\}$ as the parameterization of this composition[4]. Transformations of this kind have recently been popularized in a different context as flows (Rezende et al. 2015). While our model applies for $\theta \in \mathbb{R}^d$, it also accommodates the case of function-valued (i.e. input-dependent) parameters $\theta : \mathcal{X} \to \mathbb{R}^d$ parameterized in our case by Neural Networks.

Taking $\mathbf{f}_0 \sim \text{GP}(\mu(\mathbf{X}), K_\nu(\mathbf{X}, \mathbf{X}'))$ as a sample from the base GP, we then define the TGP as $\mathbf{f}_K = \mathbb{G}_\theta(\mathbf{f}_0)$. For simplicity, this chapter restricts attention to element-wise mappings. Because such mappings produce diagonal Jacobians, they only affect the marginals of the GP, so that for any fixed $\mathbf{X}' \in \mathcal{X}$, $\mathbf{f}_K(\mathbf{X}') = \mathbb{G}_\theta(\mathbf{f}_0(\mathbf{X}'))$. Thus, we will often refer to them as *diagonal/marginal* transformations/flows. Note that the resulting TGP $\mathbf{f}_K$ can be seen as an input-dependent generalization of the Gaussian Copula Process discussed in (Wilson et al. 2010).

[4] I have changed the notation on the parameterization. I will use θ for the flow parameters and ν for the covariance parameters. Also $\mathbf{f}$ changes to $\mathbf{f}_0$.

5.3.2 Input-dependent Flows

A simple example for a marginal flow is given by stacking K Sinh-Arcsinh-Linear (SAL) flows (Rios et al. 2019):

$$\begin{aligned} \mathbf{f}_1 &= d_1 \cdot \sinh(b_1 \cdot \operatorname{arcsinh}(\mathbf{f}_0) - a_1) + c_1 \\ &\dots \\ \mathbf{f}_K &= d_K \cdot \sinh(b_K \cdot \operatorname{arcsinh}(\mathbf{f}_{K-1}) - a_K) + c_K \end{aligned} \tag{5.11}$$

In this example, $\mathbb{G}_\theta$ is not input-dependent and $\theta_{k-1} = \{a_k, b_k, c_k, d_k\}$. Figure 5.3 illustrates the effect of such a transform on a base GP for $K = 3$ as

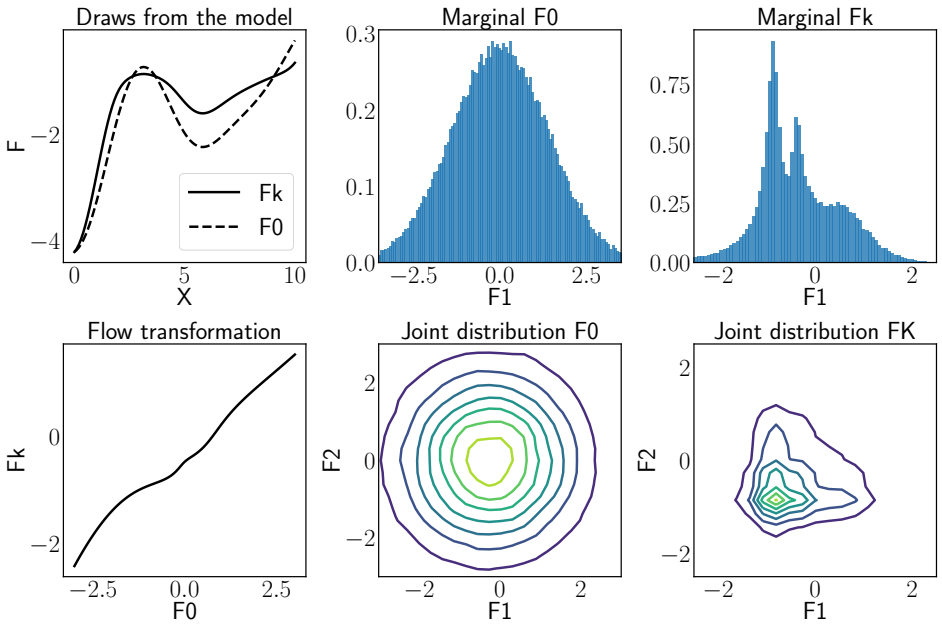


Figure 5.3: Flow constructed as in Equation 5.11 with $K = 3$. The parameters of the flow were obtained from one of the experiments ran in this work. We can see the effect of the warping function in the marginal distribution (top) and joint distribution (bottom). The flow allows us to learn multimodal marginal distributions. It shall be noted that since the flow is an element-wise mapping, only the marginal distributions are changed, i.e. the dependencies of $\mathbf{f}_K$ are driven by the GP even though the plot of joint distribution might let us conclude something different.

learned in one of our experiments. We can see that the flow changes the Gaussianity of the GP both in the marginal and joint distribution. It shall be noted that contrary to what we could think, the dependencies between the random variables from the GP and the TGP are the same, a property derived from Sklar's theorem (Sklar 1959; Wilson et al. 2010).

The above transformation can be made input-dependent. The only thing required is a reparameterization. In particular, one only has to replace the scalar parameters a_k, b_k, c_k, and d_k with the function-valued parameters $\alpha_k, \beta_k, \gamma_k, \delta_k : \mathcal{X} \to \mathbb{R}$.

We achieve this via Neural Networks with L layers so that for any fixed $\mathbf{X} \in \mathcal{X}$, the transformation's parameters are $\{\alpha_k(\mathbf{X}), \beta_k(\mathbf{X}), \gamma_k(\mathbf{X}), \delta_k(\mathbf{X})\}_{k=0}^{K-1}$. Thus, if the Neural Network's weights $\{\mathbf{W}^l\}_{l=1}^{L}$ are fitted without accounting for parameter uncertainty, $\theta = \{\mathbf{W}^l\}_{l=1}^{L}$. Note that a model of this form will be able to model non-stationary processes. This is illustrated in Figure 5.4 where we illustrate how the marginal distribution at different locations is different

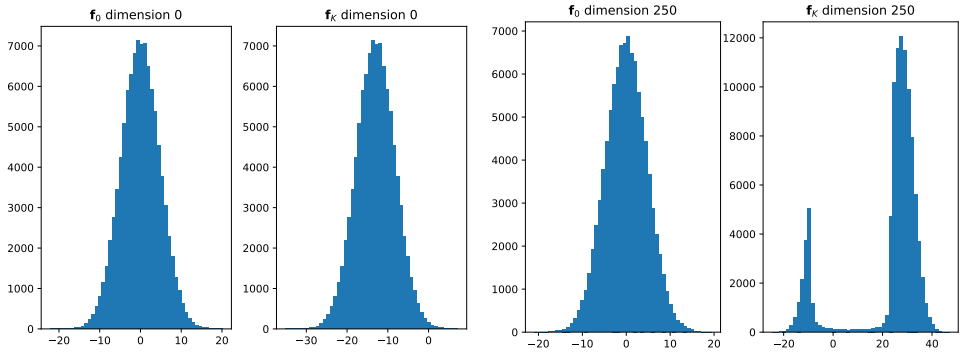


Figure 5.4: This figure illustrates the effect of an input dependent flow on the resulting marginal distributions. The two left figures represent the marginal distribution at the first element of the index set of the stochastic process, while the two on the right represent the element 250. From the two left figures, the one on the left represents the marginal distribution of the GP, while the second represents the marginal distribution of the TGP $\mathbf{f}_K$. The same applies to the two right figures. Since the flow is input dependent, a different flow parameterization applies on each marginal. We can see that $p(\mathbf{f}_0)$ is the same on both elements of the index set, but $p(\mathbf{f}_K)$ and hence the marginals are different. On the first index of the distribution, the TGP just shifts $p(\mathbf{f}_0)$, while on the right the distribution is turn into a multimodal distribution. Clearly and even though the dependencies of these random variables are the same, the TGP can model non-stationary processes.

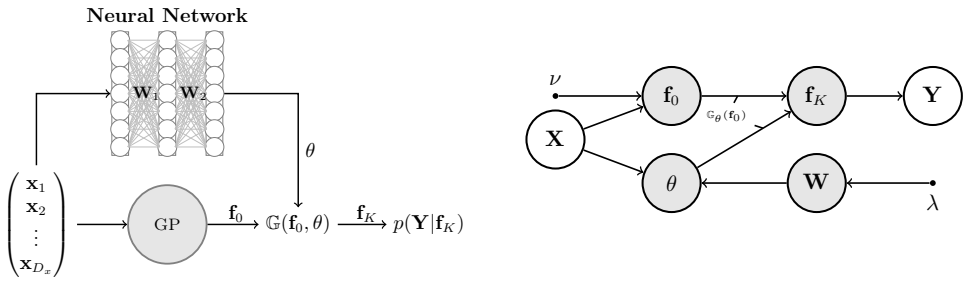


Figure 5.5: A pictorial representation of our general formulation that highlights the role of the Neural Network (left). As seen in the figure, the output of the neural network gives the parameters of the flow $\mathbb{G}$. We can further incorporate uncertainty into this parameters by defining a prior $p(\mathbf{W})$ over the Neural Network parameters. The complete graphical model of the proposed approach is given in the right figure.

when using an input dependent flow. A graphical depiction of our proposed model is provided in Figure 5.5.

5.3.3 Bayesian Priors on Flows

In this work, we find that a Bayesian treatment of $\{\mathbf{W}^l\}_{l=1}^L$ (i.e. rather than using a Neural Network we use a Bayesian Neural Network to encode input dependency) significantly improves test set performance. This is hardly surprising: input-dependent flows in the form of Neural Networks introduce a considerable number of additional hyperparameters, making a naive implementation prone to over-fitting even though we optimize the marginal likelihood. The reason for this is that enriching GP priors with non-Bayesian flows provide additional flexibility via hyperparameters which are not regularized via a complexity penalty at inference time. By placing a Bayesian prior $p(\mathbf{W})$ on the network weights $\mathbf{W} = \{\mathbf{W}^l\}_{l=1}^L$, we effectively regularize the network weights and avoid this issue. This means that we integrate over $\{\mathbf{W}^l\}_{l=1}^L$ at test time, accounting for uncertainty in θ. Though the prior could be chosen arbitrarily, we consider the fully factorized normal prior $p_\lambda(\mathbf{W}) = \mathcal{N}(\mathbf{W}; 0, \lambda^{-1} I^{|\mathbf{W}|\times|\mathbf{W}|})$. The complete generative model of the proposed approach is given by:

$$
\begin{aligned}
&\mathbf{f}_0|\mathbf{X} \sim \mathrm{GP}(\mu(\mathbf{X}), C_\nu(\mathbf{X}, \cdot)) \\
&\mathbf{W} \sim p_\lambda(\mathbf{W}) \\
&\theta(\mathbf{X}, \mathbf{W}) = \mathrm{DNN}(\mathbf{X}, \mathbf{W}) \\
&\mathbf{f}_K|\theta, \mathbf{X}, \mathbf{W} = \mathbb{G}_{\theta(\mathbf{X},\mathbf{W})}(\mathbf{f}_0)
\end{aligned}
\tag{5.12}
$$

Unlike in previous work (Wilson et al. 2010; Wauthier et al. 2010), we not only quantify uncertainty about the parameter θ, but also make it an input-dependent function.

5.3.4 Induced Distributions

One of the nice properties of the proposed model is that we can write the probability distribution induced by the transformation. By virtue of an iterated application of the change of variable formula and the inverse function theorem we have:

$$p(\mathbf{f}_K|\mathbb{G}, \mathbf{X}) = p(\mathbf{f}_0|\mathbf{X}) \prod_{k=0}^{K-1} \left| \det \frac{\partial \mathbb{G}_{\theta_k}(\mathbf{f}_k)}{\partial \mathbf{f}_k} \right|^{-1}. \tag{5.13}$$

By using a marginal flow, Sklar's theorem (Sklar 1959) implies that the dependencies in $p(\mathbf{f}_0)$ and $p(\mathbf{f}_K)$ are driven by the same Copula—the GP in our case. Though the copula is the same, $\mathbf{f}_K$ will generally have non-Gaussian marginals (see Figure 5.3). While this chapter restricts attention to diagonal mappings for simplicity, the presented derivations and methods may be extended to non-diagonal transformations such as those in (Rios 2020). In practical terms, non-diagonal transformations could be used to model arbitrary copulas and correlation structures; and we elaborate on this version of the model in the next section.

Whether $\mathbb{G}_\theta$ is a diagonal or non-diagonal transformation, we require that the resulting $\mathbf{f}_K$ is a valid stochastic process (and thus a valid function prior). This amounts to checking whether the resulting collection of random variables satisfies the necessary consistency conditions, which holds by simple arguments for marginal flows (Rios 2020). In order to employ flows such as e.g. Real NVP (Dinh et al. 2017), one needs to prove that these conditions are still satisfied. We leave this for future work, as the associated theory is highly dependent on the exact flow in question.

5.4 Inference in the Transformed Gaussian Process

So far we have presented the proposed TGP showing that it can be more expressive than a GP. In this section, we present the inference algorithm, which is the other main contribution of this work. Note that for general constructions of $\mathbb{G}$ both the posterior and the posterior predictive distributions are analytically intractable. The works from (Wilson et al. 2010; Wauthier et al. 2010) rely on a Laplace approximation to the posterior distribution. However, this approximation still requires $\mathcal{O}(N^3)$ computations so it is not suitable for training and making predictions with large datasets. Moreover, Laplace approximation is not amenable to stochastic variational inference. In this subsection, we present a sparse variational inference algorithm similar to that presented for standard GP that is also amenable to stochastic optimization.

5.4.1 Sparse Prior

Since the transformation $\mathbb{G}$ induces a valid stochastic process (i.e. the finite dimensional distributions are consistent) we can introduce inducing points in the joint distribution, which give us the general joint distribution for any valid $\mathbb{G}$ (diagonal or non diagonal):

$$p(\mathbf{f}_K, \mathbf{u}_K) = p(\mathbf{f}_0, \mathbf{u}_0|\mathbf{X}, \mathbf{Z}) \prod_{k=0}^{K-1} \underbrace{\left| \det \begin{pmatrix} \frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{f}_k} & \frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{u}_k} \\ \frac{\partial \mathbb{G}_\theta(\mathbf{u}_k)}{\partial \mathbf{f}_k} & \frac{\partial \mathbb{G}_\theta(\mathbf{u}_k)}{\partial \mathbf{u}_k} \end{pmatrix} \right|^{-1}}_{\mathbf{J}_{\mathbf{f}_k, \mathbf{u}_k}} \tag{5.14}$$

Where each element $\frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{f}_k}$ is itself the Jacobian of the transformation of function evaluations at the elements of $\mathbf{X}$. By noting that the determinant of a block diagonal matrix can be computed as:

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det \left(A - BD^{-1}C\right) \det \left(D\right), \tag{5.15}$$

the joint distribution $p(\mathbf{f}_K, \mathbf{u}_K)$ factorizes as follows:

$$
\begin{aligned}
p(\mathbf{f}_K, \mathbf{u}_K) &= p(\mathbf{f}_K|\mathbf{u}_K)p(\mathbf{u}_K) \\
p(\mathbf{f}_K|\mathbf{u}_K) &= p(\mathbf{f}_0|\mathbf{u}_0) \prod_{k=0}^{K-1} \left| \det \left[\underbrace{\frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{f}_k}}_{\mathbf{J}_{\mathbf{f}_k}} - \underbrace{\frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{u}_k}}_{\mathbf{J}_{\mathbf{f}_k|\mathbf{u}_k}} \left(\underbrace{\frac{\partial \mathbb{G}_\theta(\mathbf{u}_k)}{\partial \mathbf{u}_k}}_{\mathbf{J}_{\mathbf{u}_k}} \right)^{-1} \underbrace{\frac{\partial \mathbb{G}_\theta(\mathbf{u}_k)}{\partial \mathbf{f}_k}}_{\mathbf{J}_{\mathbf{u}_k|\mathbf{f}_k}} \right] \right|^{-1} \\
p(\mathbf{u}_K) &= p(\mathbf{u}_0) \prod_{k=0}^{K-1} \left| \det \frac{\partial \mathbb{G}_\theta(\mathbf{u}_k)}{\partial \mathbf{u}_k} \right|^{-1}
\end{aligned}
\tag{5.16}
$$

where we make use of $p(\mathbf{f}_K|\mathbf{u}_K) = {}^{p(\mathbf{f}_K, \mathbf{u}_K)}/_{p(\mathbf{u}_K)}$ to derive the expression for the conditional distribution, with $p(\mathbf{u}_0)$ and $p(\mathbf{f}_0|\mathbf{u}_0)$ given by:

$$
\begin{aligned}
p(\mathbf{u}_0) &= \mathcal{N}(\mathbf{u}_0|0, K_{\mathbf{Z},\mathbf{Z}}) \\
p(\mathbf{f}_0 \mid \mathbf{u}_0) &= \mathcal{N}(\mathbf{f}_0|K_{\mathbf{X},\mathbf{Z}}K_{\mathbf{Z},\mathbf{Z}}^{-1}\mathbf{u}_0, K_{\mathbf{X},\mathbf{X}} - K_{\mathbf{X},\mathbf{Z}}K_{\mathbf{Z},\mathbf{Z}}^{-1}K_{\mathbf{Z},\mathbf{X}})
\end{aligned}
\tag{5.17}
$$

Furthermore, note that for marginal flows, B and C are evaluated to zero and so the above factorization is given by:

$$
\begin{aligned}
p(\mathbf{f}_K, \mathbf{u}_K) &= p(\mathbf{f}_K|\mathbf{u}_K)p(\mathbf{u}_K) \\
p(\mathbf{f}_K|\mathbf{u}_K) &= p(\mathbf{f}_0|\mathbf{u}_0) \prod_{k=0}^{K-1} \left| \det \frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{f}_k} \right|^{-1} \\
p(\mathbf{u}_K) &= p(\mathbf{u}_0) \prod_{k=0}^{K-1} \left| \det \frac{\partial \mathbb{G}_\theta(\mathbf{u}_k)}{\partial \mathbf{u}_k} \right|^{-1}
\end{aligned}
\tag{5.18}
$$

where now $\frac{\partial \mathbb{G}_\theta(\mathbf{f}_k)}{\partial \mathbf{f}_k}$ is a diagonal matrix where the elements of the diagonal are given by $\frac{\partial \mathbb{G}_\theta(\mathbf{f}_{k,n})}{\partial \mathbf{f}_{k,n}}$.

5.4.2 Choice of the Variational Distribution

For computational convenience, we define the approximate posterior such as the conditional distribution cancels. This is achieved by defining a base variational distribution in the original GP space, which is warped with the same flow as the prior. This gives:

$$q(\mathbf{f}_K|\mathbf{u}_K) = p(\mathbf{f}_K|\mathbf{u}_K)q(\mathbf{u}_K) \tag{5.19}$$

where

$$q(\mathbf{u}_K) = \mathcal{N}(\mathbf{u}_0|\mathbf{m}, \mathbf{S})\mathbf{J}_{\mathbf{u}_K} \tag{5.20}$$

with $\mathbf{m} \in \mathbb{R}^{M\times 1}$ and $\mathbf{S} \in \mathbb{R}^{M\times M}$ being the variational parameters and $p(\mathbf{f}_K|\mathbf{u}_K)$ is given by Equation 5.16.

5.4.3 Evidence Lower Bound

Following a similar procedure as in standard GP, we approximate the true posterior using the above variational distribution:

$$\begin{aligned}
&\mathrm{KLD}[q(\mathbf{f}_K, \mathbf{u}_K)||p(\mathbf{f}_K, \mathbf{u}_K|\mathbf{Y})] = \\
&\int q(\mathbf{f}_K, \mathbf{u}_K) \log q(\mathbf{f}_K, \mathbf{u}_K) \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K \\
&- \int q(\mathbf{f}_K, \mathbf{u}_K) \log \frac{p(\mathbf{Y}|\mathbf{f}_K, \mathbf{u}_K)p(\mathbf{f}_K, \mathbf{u}_K)}{p(\mathbf{Y})} \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K = \\
&\int q(\mathbf{f}_K, \mathbf{u}_K) \log \frac{q(\mathbf{f}_K, \mathbf{u}_K)}{p(\mathbf{f}_K, \mathbf{u}_K)} \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K \\
&- \int q(\mathbf{f}_K, \mathbf{u}_K) \log p(\mathbf{Y}|\mathbf{f}_K, \mathbf{u}_K) \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K + \log p(\mathbf{Y}) = \\
&\underbrace{\int q(\mathbf{u}_K) \log \frac{\cancel{p(\mathbf{f}_K|\mathbf{u}_K)}q(\mathbf{u}_K)}{\cancel{p(\mathbf{f}_K|\mathbf{u}_K)}p(\mathbf{u}_K)} \mathrm{d}\mathbf{u}_K - \int q(\mathbf{f}_K) \log p(\mathbf{Y}|\mathbf{f}_K) \mathrm{d}\mathbf{f}_K}_{-\mathrm{ELBO}} + \log p(\mathbf{Y})
\end{aligned} \tag{5.21}$$

Before deriving the final expression, we shall compute the only unknown distribution $q(\mathbf{f}_K)$. It turns out that for marginal flows we can compute this expression analytically as follows:

$$
\begin{aligned}
q(\mathbf{f}_K) &= \int q(\mathbf{f}_K, \mathbf{u}_K) \mathrm{d}\mathbf{u}_K \\
&= \int p(\mathbf{f}_0|\mathbf{u}_0) q(\mathbf{u}_0) \mathbf{J}_{\mathbf{f}_K, \mathbf{u}_K} \mathrm{d}\mathbf{u}_K
\end{aligned}
\tag{5.22}
$$

Because $\mathbb{G}$ is a marginal flow the Jacobian matrix is diagonal and decomposes such that $\mathbf{J}_{\mathbf{f}_K, \mathbf{u}_K} = \mathbf{J}_{\mathbf{f}_K} \mathbf{J}_{\mathbf{u}_K}$, see Equation 5.18. Substituting this into the above expression and applying LOTUS rule[5] by recognizing this as an expectation w.r.t. $q(\mathbf{u}_K)$:

$$
\begin{aligned}
q(\mathbf{f}_K) &= \int p(\mathbf{f}_0|\mathbf{u}_0) q(\mathbf{u}_0) \mathbf{J}_{\mathbf{f}_K} \mathbf{J}_{\mathbf{u}_K} \mathrm{d}\mathbf{u}_K \\
&= \mathbf{J}_{\mathbf{f}_K} \int p(\mathbf{f}_0|\mathbf{u}_0) q(\mathbf{u}_0) \mathrm{d}\mathbf{u}_0 \\
&= \mathbf{J}_{\mathbf{f}_K} q(\mathbf{f}_0)
\end{aligned}
\tag{5.23}
$$

where $\mathbf{J}_{\mathbf{f}_K}$ does not depend on $\mathbf{u}_0$ and the marginal $q(\mathbf{f}_0)$ is given by Equation 5.8. With this, the final objective function is given by:

$$
\begin{aligned}
\text{ELBO} = &- \int q(\mathbf{u}_K) \log \frac{q(\mathbf{u}_0) \cancel{\mathbf{J}_{\mathbf{u}_K}}}{p(\mathbf{u}_0) \cancel{\mathbf{J}_{\mathbf{u}_K}}} \mathrm{d}\mathbf{u}_K + \int q(\mathbf{f}_K) \log p(\mathbf{Y}|\mathbf{f}_K) \mathrm{d}\mathbf{f}_K = \\
&- \int q(\mathbf{u}_0) \log \frac{q(\mathbf{u}_0)}{p(\mathbf{u}_0)} \mathrm{d}\mathbf{u}_0 + \int q(\mathbf{f}_0) \log p(\mathbf{Y}|\mathbb{G}(\mathbf{f}_0)) \mathrm{d}\mathbf{f}_0 = \\
&- \text{KLD}[q(\mathbf{u}_0)||p(\mathbf{u}_0)] + \sum_{n=1}^{N} \int q(\mathbf{f}_{0,n}) \log p(\mathbf{Y}_n|\mathbb{G}(\mathbf{f}_{0,n})) \mathrm{d}\mathbf{f}_{0,n}
\end{aligned}
\tag{5.24}
$$

where we have applied the LOTUS rule twice again from the first to the second row, and finally take the sum out of the integral and apply the marginalization property. Note that the KLD between the prior and the approximate posterior could also be derived by noting that the KLD is invariant under a reparametrization, although we derive it here through the LOTUS rule by noting that the KLD is an expectation of a log-ratio w.r.t. $q(\mathbf{u}_K)$.

The use of marginal flows and factorizing likelihoods results in the expected log-likelihood (ELL) term being decomposable across the latent variables $q(\mathbf{f}_{0,n})$ and observations $\mathbf{Y}_n$, making it particularly suitable for stochastic variational

[5] Described at the end of subsubsection 4.2.2.

inference and big N as in (Hensman et al. 2013). In other words, we can train using mini batches of data. The individual ELL components will generally be unavailable in closed form and computed using one-dimensional Gaussian quadrature. The whole process can be coded in parallel making use of modern GPUs.

Note that the presented algorithm scales to large dataset both at training and test times since we inherit the complexity from standard sparse GP. We will illustrate this in the experiment section.

Non marginal flows

The above derivation is particular for marginal flows. In this subsection we present the necessary steps for non diagonal flows. The difference is that we cannot integrate out the inducing points analytically, hence the ELL can be computed using Monte Carlo. Note that the marginal $q(\mathbf{f}_K)$ is given by:

$$\begin{aligned} q(\mathbf{f}_K) &= \int q(\mathbf{f}_K, \mathbf{u}_K) \mathrm{d}\mathbf{u}_K \\ &= \int p(\mathbf{f}_K|\mathbf{u}_K) q(\mathbf{u}_K) \mathrm{d}\mathbf{u}_K \\ &= \int [\mathbf{J}_{\mathbf{f}_K} - \mathbf{J}_{\mathbf{f}_K|\mathbf{u}_K} \mathbf{J}_{\mathbf{u}_K}^{-1} \mathbf{J}_{\mathbf{u}_K|\mathbf{f}_K}] p(\mathbf{f}_0|\mathbf{u}_0) q(\mathbf{u}_0) \mathrm{d}\mathbf{u}_0 \\ &= q(\mathbf{f}_0) \mathbf{J}_{\mathbf{f}_K} - \int [\mathbf{J}_{\mathbf{f}_K|\mathbf{u}_K} \mathbf{J}_{\mathbf{u}_K}^{-1} \mathbf{J}_{\mathbf{u}_K|\mathbf{f}_K}] p(\mathbf{f}_0|\mathbf{u}_0) q(\mathbf{u}_0) \mathrm{d}\mathbf{u}_0 \end{aligned} \tag{5.25}$$

Note that integrating the inducing points will be generally intractable due to the non-linearity of the flow $\mathbb{G}$ that appears in the conditional prior $p(\mathbf{f}_K|\mathbf{u}_K)$ through the elements $\mathbf{J}_{\mathbf{f}_K|\mathbf{u}_K}$,$\mathbf{J}_{\mathbf{u}_K|\mathbf{f}_K}$ and $\mathbf{J}_{\mathbf{u}_K}$. Resorting to a Monte-Carlo approximation the ELL is computed as follows:

$$\begin{aligned} \mathrm{ELL} &= \sum_{n=1}^{N} \int q(\mathbf{f}_K, \mathbf{u}_K) \left[\log p(\mathbf{Y}_n|\mathbf{f}_{K,n})\right] \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K \\ &= \sum_{n=1}^{N} \int q(\mathbf{f}_{K,n}, \mathbf{u}_K) \left[\log p(\mathbf{Y}_n|\mathbf{f}_{K,n})\right] \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K \\ &\approx \sum_{n=1}^{N} \frac{1}{S} \sum_{s=1}^{S} \left[\log p(\mathbf{Y}_n|\mathbf{f}_{K,n,s})\right] \end{aligned} \tag{5.26}$$

where we follow similar steps to marginal flows, i.e. we integrate out all the elements from $\mathbf{f}_K$ but $\mathbf{f}_{K,n}$. The last line is the Monte-Carlo approximation where samples are obtained by the generative process defined for flow based models, i.e *sample from the base distribution* and *warp samples with the flow*:

$$\begin{aligned} \mathbf{u}_{0,s} &\sim q(\mathbf{u}_0) \\ \mathbf{f}_{0,n,s} &\sim p(\mathbf{f}_{0,n}|\mathbf{u}_{0,s}) \\ \mathbf{f}_{K,n,s}, \mathbf{u}_{K,s} &= \mathbb{G}_\theta(\mathbf{f}_{0,n,s}, \mathbf{u}_{0,s}) \end{aligned} \tag{5.27}$$

where the samples $\mathbf{u}_{K,s}$ are then discarded.

5.4.4 Input Dependent Flows

In the case in which flows are input dependent, we have two options. If the parameters of the Neural Network are fixed, i.e. we do not perform inference over them, then for each datapoint $\mathbf{X}_n$ we need to compute the corresponding parameter when evaluating the ELL. Note that this can be easily done in parallel still making the bound efficient and computable in parallel hardware architectures.

When the flows are Bayesian, we need to perform inference over the Neural Network parameters. Since the posterior over the Neural Network weights is intractable, we approximate the posterior using a variational distribution $q_\phi(\mathbf{W})$, as in chapter 4. We assume independence between the latent processes and the latent parameters both in the prior and variational posterior. Thus, the complete prior and variational distributions are given by:

$$\begin{aligned} p(\mathbf{f}_K, \mathbf{u}_K, \mathbf{W}) &= p(\mathbf{f}_K|\mathbf{u}_K)p(\mathbf{u}_K)p_\lambda(\mathbf{W}) \\ q(\mathbf{f}_K, \mathbf{u}_K, \mathbf{W}) &= p(\mathbf{f}_K|\mathbf{u}_K)q(\mathbf{u}_K)q_\phi(\mathbf{W}) \end{aligned} \tag{5.28}$$

where ϕ are the variational parameters of the BNN. By plugging this into the ELBO we arrive at:

$$
\begin{aligned}
\text{ELBO} = & \int q(\mathbf{f}_K, \mathbf{u}_K) q(\mathbf{W}) \log \prod_{n=1}^{N} p(\mathbf{Y}_n | \mathbf{f}_{K,n}) \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K \mathrm{d}\mathbf{W} \\
& - \int q(\mathbf{f}_K, \mathbf{u}_K) q(\mathbf{W}) \log \frac{q(\mathbf{f}_K, \mathbf{u}_K) q_\phi(\mathbf{W})}{p(\mathbf{f}_K, \mathbf{u}_K) p(\mathbf{W})} \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{u}_K \mathrm{d}\mathbf{W} = \\
& = \int q(\mathbf{f}_0) q_\phi(\mathbf{W}) \log \left[\prod_{n=1}^{N} p(\mathbf{Y}_n | \mathbf{f}_{K,n}) \right] \mathrm{d}\mathbf{f}_0 \mathrm{d}\mathbf{W} \\
& - \mathrm{KLD}[q(\mathbf{u}_0) || p(\mathbf{u}_0) - \mathrm{KLD}[q_\phi(\mathbf{W}) || p(\mathbf{W})] = \\
& \sum_{n=1}^{N} \int q(\mathbf{f}_{0,n}) q_\phi(\mathbf{W}) \log p(\mathbf{Y}_n | \mathbb{G}_{\theta(\mathbf{X}, \mathbf{W})}(\mathbf{f}_{0,n})) \mathrm{d}\mathbf{f}_{0,n} \mathrm{d}\mathbf{W} \\
& - \mathrm{KLD}[q(\mathbf{u}_0) || p(\mathbf{u}_0)] - \mathrm{KLD}[q_\phi(\mathbf{W}) || p(\mathbf{W})] \approx \\
& \sum_{n=1}^{N} \frac{1}{S} \sum_{s=1}^{S} \int q(\mathbf{f}_{0,n}) \log p(\mathbf{Y}_n | \mathbb{G}_{\theta(\mathbf{X}, \mathbf{W}_s)}(\mathbf{f}_{0,n})) \mathrm{d}\mathbf{f}_{0,n} \\
& - \mathrm{KLD}[q(\mathbf{u}_0) || p(\mathbf{u}_0)] - \mathrm{KLD}[q_\phi(\mathbf{W}) || p(\mathbf{W})]; \mathbf{W}_s \sim q(\mathbf{W})
\end{aligned}
\tag{5.29}
$$

This bound has two interesting properties. First, one can allow for low variance and unbiased gradients w.r.t. ϕ by reparameterization (something satisfied for popular choices of $q(\mathbf{W})$ such as the mean-field Gaussian family) and use the local reparameterization trick, as in the previous chapter. Second, one can account for prior miss-specification by substituting the KLD for other divergences, which has been shown to improve the performance of this model (Knoblauch et al. 2019).

In our work however, we have implemented the BNN using Monte Carlo dropout (Gal et al. 2016). Although this technique has its own problems (e.g. tuning the dropout rate or poor performance in recent benchmarks) we found it worked well in the proposed model. Moreover it can be more efficiently trained and also allow us to avoid some well-known problems of mean-field VI such as variance under-estimation already discussed in the previous chapter. Nevertheless, our bound can be efficiently trained regardless of the specification of the variational family by using batched matrix computations by means of the BLAS or CUBLAS libraries.

Finally, note that computing the forward passes through the Neural Network are independent of the computation of $q(\mathbf{f}_0)$. This means that one can parallelize the computation of $q(\mathbf{f}_0)$ and $\theta(\mathbf{W}_s, \mathbf{X})$.

5.4.5 Computational benefits of the approximate posterior

This final subsection summarizes and extends the computational benefits of the choice of the approximate family. One of the most interesting properties of our proposed approach is that both training and predictions can be done without inverting $\mathbb{G}$, allowing us to use any expressive invertible transformation, in contrast to the Warped Gaussian Process, discussed in the next section.

On the other hand, other choices for the variational distribution would imply an undesirable increase in computational time. First, the definition of $q(\mathbf{u}_K)$ allows us to compute the KLD in closed form, avoiding the need to resort to estimation by sampling and to compute the Jacobian of the transformation. Note that computing this Jacobian can be done in linear time for marginal flows although it will have, in general, a cubic cost. Moreover, other choices of $q(\mathbf{u}_K)$ could require the computation of the inverse $\mathbb{G}_\theta^{-1}(\mathbf{u}_K)$ to evaluate the density of the posterior sample under $p(\mathbf{u}_0)\mathbf{J}_{\mathbf{u}_K}$. On the other hand, not canceling $p(\mathbf{f}_K|\mathbf{u}_K)$ would also require to approximate the determinant of the transformation $\mathbf{J}_{\mathbf{f}_K}$ and the inverse $\mathbb{G}_\theta$, with *the whole dataset* $\mathbf{X}$, something that cannot be done stochastically. So canceling this term is not only important to allow stochastic variational inference, but also to avoid costly Jacobian/inverse computations.

Finally, the use of marginal flows and definition of the variational posterior allow us to analytically integrate out the inducing points, a very desirable property both for training and when making predictions.

5.5 Warped Gaussian Processes

The use of invertible transformations in the GP literature is not new. In fact, the presented TGP model was already used by (Wauthier et al. 2010; Wilson et al. 2010). However, rather than as a general-purpose prior over functions, both works propose the TGP for specific applications. Clearly our work extends (Wauthier et al. 2010; Wilson et al. 2010) by providing a sparse inference algorithm, allowing Bayesian input dependent flows, and show that the TGP can be used as a general-purpose prior over functions. Note moreover that, beyond those applications, the warping function $\mathbb{G}$ can encode inductive biases. If our data is e.g. positive constrained then it does not make sense to use a GP, since the Gaussian distribution has support overall $\mathbb{R}$. In the experiment section, we will show that we can use a TGP prior which provides a positive constrained process to model positive constrained signals.

On the other hand, we can use invertible transformations $\mathbb{T}$ to warp the labels $\mathbf{Y}$ rather than the latent function $\mathbf{f}$. This model was presented in (Snelson et al. 2003) under the name of Warped Gaussian Process (WGP). The idea is that if the full GP model (likelihood and prior) is misspecified w.r.t. the data $\mathbf{Y}$ (for example because the data is positive), we can learn a transformation $\mathbb{T}(\mathbf{Y})$ that maps this non-Gaussian data to a data that can be well modeled by a GP, i.e. the data can be mapped to a GP with additive Gaussian noise. Note that whenever $\mathbb{T}$ is non linear[6], this implies that $\mathbf{Y}$ is non-Gaussian with non-additive noise[7]. This means that one can see this transformation as aiming to fix model misspecification, and thus is only justified if one has sufficient domain knowledge about the application at hand.

Also note that the WGP cannot be applied to discrete data $\mathbf{Y}$, in contrast to the TGP. Actually, both the TGP and the WGP are complementary approaches, that target different parts of the modeling process. Beyond that, and looking at the computational aspect, the WGP has the advantage that one can train the model by computing the marginal likelihood (Snelson et al. 2003), hence there is no need for an approximation. On the other hand, predictions require to compute the inverse of the warping function, which can be computational expensive needing approximate methods such as Newton-Raphson, or restricting the expressiveness of $\mathbb{T}$ so that the inverse can be analytically computed (Rios et al. 2019).

The original implementation of the WGP inherits the same limitations as standard GP. However, it turns out that extending this model to be sparse is straightforward. One just needs to warp $\mathbf{Y}$ and add the Jacobian correction to the ELL in the ELBO. Although straightforward this is the first work that sparsifies and evaluates the WGP. In the experiment section, we will show a comparison between the WGP and the TGP. The likelihood of the complete model (WGP + TGP) is given by:

$$p(\mathbf{Y}|\mathbb{T}, \mathbf{f}_K) = p(\mathbb{T}(\mathbf{Y})|\mathbf{f}_K) \prod_{k=0}^{K} \left| \det \frac{\mathbb{T}_k(\mathbf{Y}_k)}{\mathbf{Y}_k} \right|, \tag{5.30}$$

with the final objective function being:

[6] As desirable, otherwise the use of $\mathbb{T}$ is not justified since the overall model would be a GP.

[7] Note that if $\mathbf{Y} = \mathbb{T}(\mathbf{X} + \epsilon)$ with $\mathbb{T}$ being non-linear, then the output $\mathbf{Y}$ will not be an additive noise model since linearity implies $t(x + y) = t(x) + t(y)$

$$\begin{aligned}\text{ELBO} = \sum_{n=1}^{N} \left[\int q(\mathbf{f}_{0,n}) \log p(\mathbb{T}(\mathbf{Y}_n)|\mathbb{G}(\mathbf{f}_{0,n}))\mathrm{d}\mathbf{f}_{0,n} + \log \prod_{k=0}^{K} \left| \det \frac{\mathbb{T}_k(\mathbf{Y}_{k,n})}{\mathbf{Y}_{k,n}} \right| \right] \\ - \text{KLD}[q(\mathbf{u}_0)||p(\mathbf{u}_0)]\end{aligned} \tag{5.31}$$

5.6 Predictions

In order to compute the posterior predictive distribution at test time, we first replace the posterior distribution with the approximate posteriors which gives:

$$\begin{aligned}p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbb{T}(\mathbf{Y}^*)|\mathbf{f}_K) q(\mathbf{f}_K) q(\mathbf{W}) \mathrm{d}\mathbf{f}_K \mathrm{d}\mathbf{W} \prod_{k=0}^{K} \left| \det \frac{\mathbb{T}_k(\mathbf{Y}_k)}{\mathbf{Y}_k} \right| \approx \\ \frac{1}{S} \sum_{s=1}^{S} \int p(\mathbb{T}(\mathbf{Y}^*)|\mathbb{G}_{\theta(\mathbf{X},\mathbf{W}_s)}(\mathbf{f}_0)) q(\mathbf{f}_0) \mathrm{d}\mathbf{f}_0 \prod_{k=0}^{K} \left| \det \frac{\mathbb{T}_k(\mathbf{Y}_k)}{\mathbf{Y}_k} \right| ; \mathbf{W}_s \sim q(\mathbf{W})\end{aligned} \tag{5.32}$$

where we again apply LOTUS rule and integrate out w.r.t. the base distribution $q(\mathbf{f}_0)$. This integral can be computed with one-dimensional Quadrature, i.e. for each Monte Carlo sample $\mathbf{W}_s$ the marginalization over $q(\mathbf{f}_0)$ is performed using 100 quadrature points. We will see that this does not affect the computational performance of the proposed model.

To compute point-wise predictions such as the mean, then we can resort to approximate methods such as Gauss-Hermite quadrature and Monte Carlo, as explained above. Also, in the case of the WGP we need to compute the inverse since LOTUS rule applies again. Let $\mathbf{Z} = \mathbb{T}(\mathbf{Y}^*)$, we have for the mean:

$$\begin{aligned}\mathbb{E}[\mathbf{Y}^*] = \int \mathbf{Y}^* p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{X}, \mathbf{Y}) \mathrm{d}\mathbf{Y}^* = \\ \int \mathbb{T}^{-1}(\mathbf{Z}) p(\mathbf{Z}|\mathbf{X}^*, \mathbf{X}, \mathbf{Y}) \mathrm{d}\mathbf{Z}\end{aligned} \tag{5.33}$$

where one would need to plug in the necessary approximation steps in Equation 5.32 in order to integrate out the latent process and latent weights.

Quantiles from the posterior predictive can be computed by sampling in the case of the TGP or by warping the corresponding quantile points in $\mathbf{Z}$ using the inverse $\mathbb{T}^{-1}$ in the case of the WGP, see (Snelson et al. 2003).

5.7 Experimental Evaluation

In this final section, we evaluate the proposed TGP. Beyond its well-calibrated uncertainties, we illustrate other benefits from the TGP both in terms of computational performance and applications. Experimental details on the architectures and training hyperparameters are provided in Appendix B[8].

We will use the following acronyms to refer to the different models compared: Sparse Variational Gaussian Process (SVGP) described in section 5.2, Deep Gaussian Processes (DGP) with DSVI inference, TGP with non-input dependent flows (TGP), TGP with input dependent flows and a point estimation of the parameters of the Neural Network (PETGP), TGP with input dependent flows and Bayesian marginalization of the Neural Network parameters (BATGP) and the variational version of the WGP (V-WGP) described in the previous section. In the different figures the number in the subindex shows the number of inducing points used, e.g. BATGP_{100} refers to a Bayesian TGP fitted with 100 inducing points.

5.7.1 Bayesian Input Dependent TGP

In this first subsection, we illustrate the influence of making the TGP input dependent, and how accounting for uncertainty in the Neural Network's parameters affects the performance. Figure 5.6 compares the effect of parameterizing an input dependent TGP with a Neural Network with a point estimation via standard dropout (Srivastava et al. 2014) vs a Bayesian marginalization via Monte Carlo dropout (Gal et al. 2016). We can see how the BATGP prevents the Neural Network from overfitting yielding a performance boost compared with PETGP. We also illustrate how non-input-dependent flows are much less expressive than the input-dependent counterpart.

To provide a deep understanding of what is going on, Figure 5.7 shows the warping functions of an input dependent flow showing the effect of using a point estimation (top row) vs drawing samples from the approximate posterior (bottom row). We can see how the mean of both distributions is the same,

[8] In line with the rest of the chapters a Github with the implementation of the model is provided https://github.com/jmaronas/TGP.pytorch.git.

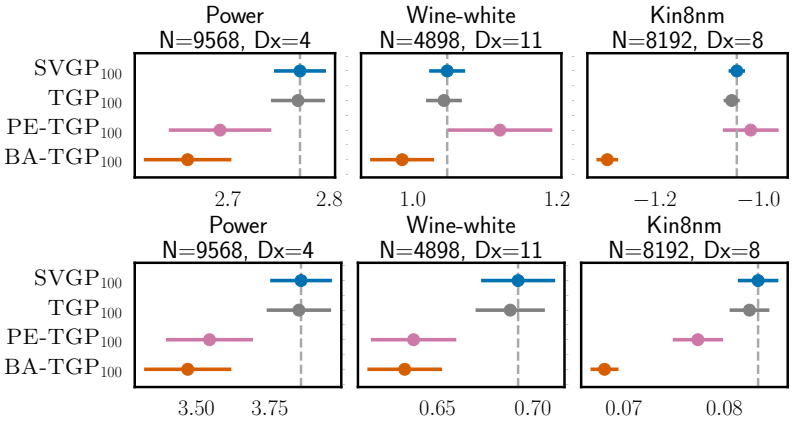


Figure 5.6: Comparison of NLL (top; left is better) and RMSE (bottom; left is better) for a standard SVGP with a non input-dependent flow (TGP), the input-dependent counterpart indexed by a Neural Network when the Neural Network is fitted using a point estimate (PETGP) or integrated out in a Bayesian fashion (BATGP).

however, the Bayesian flow provides uncertainty around the warping function. This uncertainty is responsible for the boost in performance we will see in the subsequent sections. This figure also shows how different inputs $\mathbf{X}_1$, $\mathbf{X}_2$ etc. parameterize different warping functions.

5.7.2 Calibration Properties of the TGP

In this subsection, we evaluate the performance of the TGP both in regression and classification datasets obtained from the UCI repository (Lichman 2013). To do so, we evaluate the model using the Root Mean Squared Error (RMSE) for regression and accuracy (ACC) for classification. Additionally, and in order to measure the calibration performance of the model, we report the Negative Log-Likelihood (NLL), which is a proper scoring rule since it is derived from a statistical divergence.

We compare the SVGP, DGP with a different number of layers, and the three variants of the TGP: BATGP, TGP and PETGP. Note that PETGP and BATGP share the same input-dependent architecture and learned parameters $\mathbf{W}$, and the only difference relies on whether we use standard Dropout (Srivastava et al. 2014) or Monte Carlo dropout (Gal et al. 2016) to make predictions.

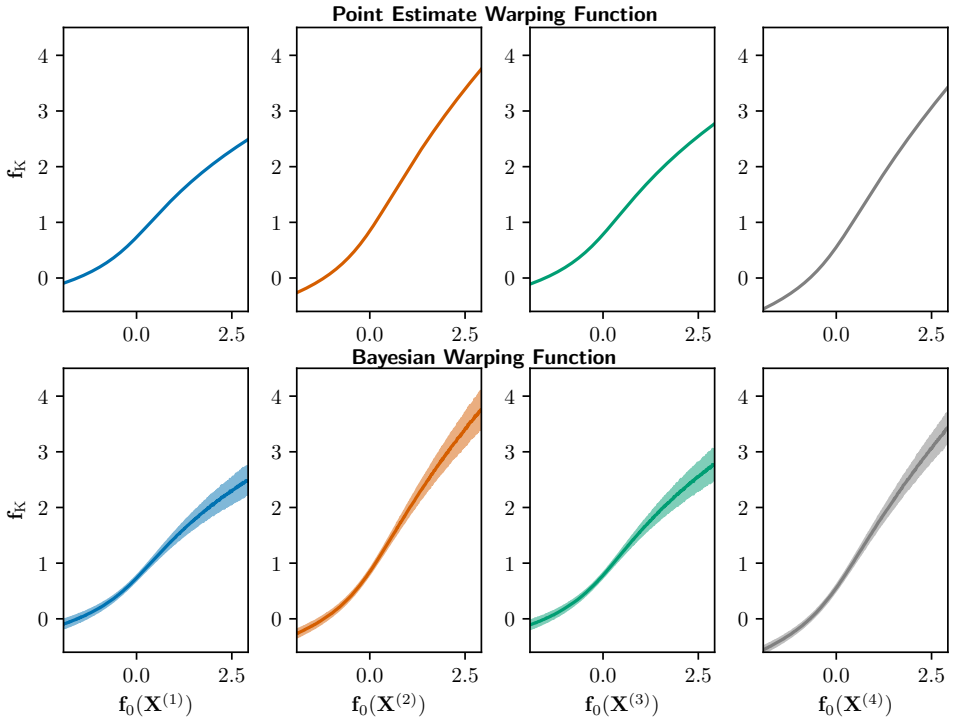
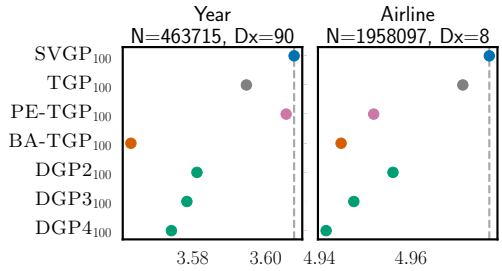


Figure 5.7: Example of warping functions obtained with input-dependent flows for the `power` dataset. The top row shows the point estimate warping function evaluated over a range $\mathbf{f}_0$, at different input locations using standard Dropout. The bottom row shows the mean and standard deviation of samples from the posterior of the Bayesian flow using Monte Carlo Dropout. We can see how the model learns a different function warping depending on the input locations and how the model accounts for parameter uncertainty.

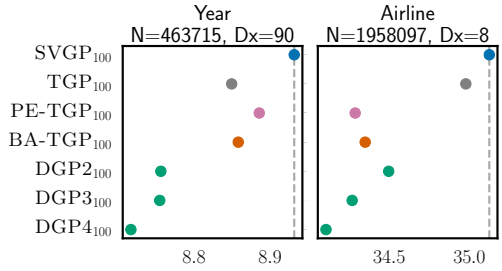
Large Scale Regression

Figure 5.8a and Figure 5.8b report the RMSE and NLL of two Large scale regression datasets. The year dataset has 0.5 Million and the airline has around 2 Million training points. This demonstrates the scalability of the proposed model to large datasets.

Across both datasets, our model outperforms the baseline SVGP both in RMSE and NLL. Furthermore, the TGP also performs well, although in general is outperformed by the BATGP. These plots also show the regularization effect of Bayesian marginalization, noted by comparing the pink and orange dotes. In



(a) NLL (left is better) for large scale regression datasets.



(b) RMSE (left is better) for large scale regression datasets.

Year the RMSE and NLL is highly improved when accounting for uncertainty in the parameters. In Airline both models provide similar RMSE, but the BATGP provides better uncertainty quantification, reflected by improved NLL scores. In these experiments only the 4-layers DGP clearly outperforms TGP, PETGP and BATGP except for the Year dataset where BATGP has a lower NLL, indicating better uncertainty quantification.

Medium-Small Regression

We now illustrate the results for the medium-small regression in Figure 5.9 (NLL) and Figure 5.10 (RMSE). We show results split across decreasing number of inducing points and different models. Across all levels of inducing points, the BATGP model ranks the best and consistently outperforms alternative models on both NLL and RMSE showing superior point-prediction, uncertainty quantification and calibration.

On the other hand, by looking at e.g `power` dataset, and comparing RMSE and NLL, we can see how in terms of RMSE both the PETGP and BATGP perform similarly. However, there is a big difference in terms of NLL, which is an indicator of good predictive uncertainty quantification provided by introducing

uncertainty in the flow parameters, as already noted in the previous subsection. This is a general tendency shown in these figures (check pink and orange dots).

Moreover, a particularly interesting outcome is the performance of the models when only using 5 inducing points. We can see that in `kin8nm`, `power` and `concrete` the 5 inducing points provides a similar performance to the 100 inducing points for the BATGP. We show in subsection 5.7.4 that even though the Neural network is highly expressive, the base SVGP is necessary and not 'ignored' by the model. Hence we can attribute the excellent performance of BATGP to both the combination of the BNN and the SVGP. We hypothesize that since the BATGP allow us to learn a non-stationary model, the underlying latent GP function can be super smooth, which is the reason for needing very few inducing points.

We can also see how the standard TGP is also able to improve upon the SVGP in some datasets, although the improvement is minimal, clearly highlighting the necessity of input-dependent flows. The fact that the standard TGP has been tested using more complex transformations than the input-dependent TGP (which uses just 1-3 length SAL flows) suggest that the boost in performance clearly comes from the input dependency and not the warping function. This means that these results can be improved by testing more complex input-dependent flow combinations, which is something we leave for future work. Also, we can see in `wine-red` that while the uncertainty quantification of our model and the DGP is quite similar, we clearly outperform it in terms of pointwise predictions. The DGP consistently outperforms the TGP and SVGP w.r.t. RMSE but in general the BATGP and PETGP achieve superior performance. These two observations might indicate that the proposed model is more expressive in terms of pointwise predictions than a DGP.

Finally, note how without doing specific model selection for the less inducing points models, the parameters extrapolated from the 100 inducing points one works very well, which means that our model is somewhat robust to the selection of hyperparameter. Also by noting that all the models are trained for 15000 epochs, we show how our model has not over-fit, although being much more complex than a 'simple' GP.

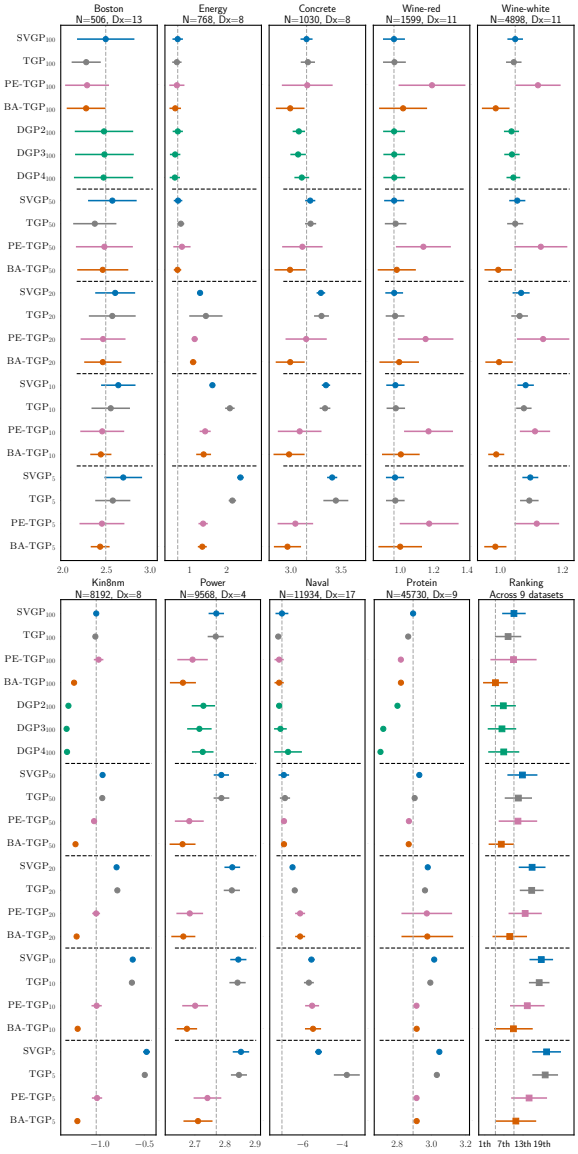


Figure 5.9: Comparing NLL (left is better) across 9 data sets for several number of inducing points. **Bottom right panel:** Ranking of the methods across all 9 data sets. TGP stands for non input-dependent flows, PETGP stands for point estimate input-dependent flows (Standard Dropout) and BATGP stands for the Bayesian input-dependent flows (MC Dropout)

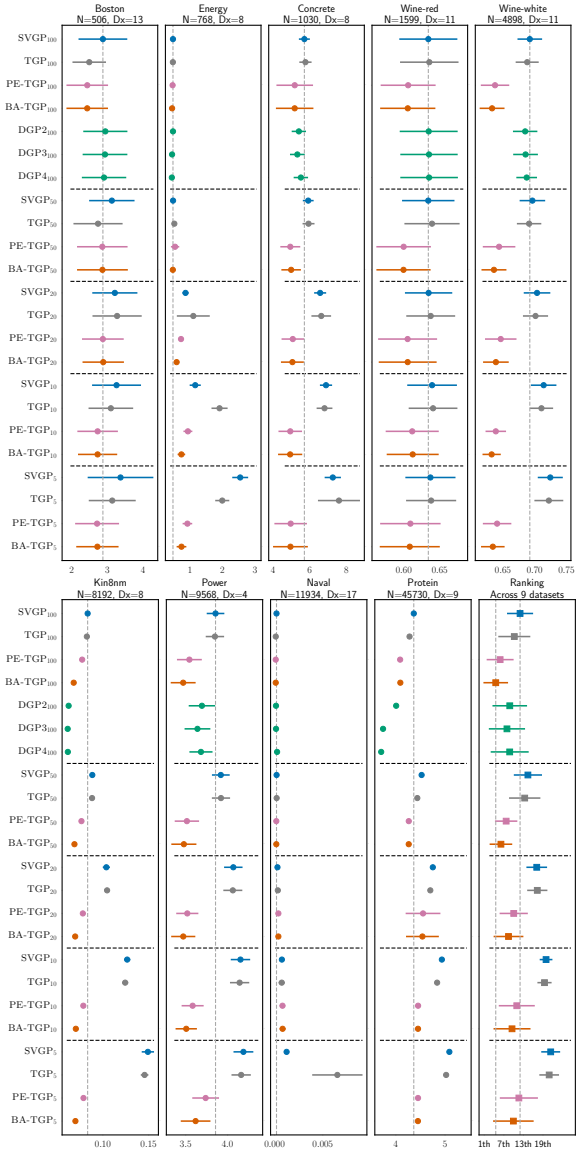
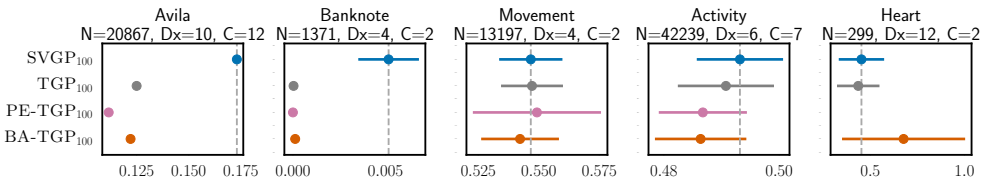


Figure 5.10: Comparing RMSE (left is better) across 9 data sets for several number of inducing points. **Bottom right panel:** Ranking of the methods across all 9 data sets. TGP stands for non input-dependent flows, PETGP stands for point estimate input-dependent flows (Standard Dropout) and BATGP stands for the Bayesian input-dependent flows (MC Dropout).

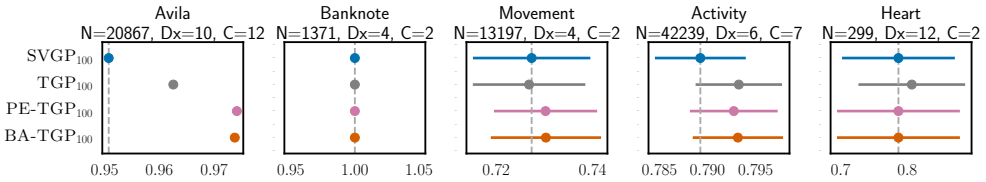
Classification

Finally, we show results on classification datasets in Figure 5.11a (NLL) and Figure 5.11b (ACC). Across all datasets our proposed models (either through input-dependent or non-input-dependent flows) outperform the SVGP, and only in `heart` does the TGP substantially outperform the input-dependent models BATGP and PETGP. We highlight the big boost in accuracy provided by our models (see e.g. `avila` dataset).

Surprisingly we observe that the PETGP performs well in classification and usually outperforms the BATGP. However, we have observed that sometimes the PETGP outputs extreme wrong values, giving a NLL of ∞. For this same model, we observe that the BATGP was able to remove those extreme predictions. We speculate that the model is correctly incorporating epistemic uncertainty relaxing extremely wrong assigned confidences. This observation corresponds to the `Heart` dataset where the pink point is not present since the NLL was ∞. Moreover, even though the BATGP solves this problem, it is unable to provide good predictions as compared with the non-input dependent TGP or the SVGP. We attribute this problem to prior misspecification. As explained by (Knoblauch et al. 2019), prior misspecification leads to a misleading quantification of uncertainty. Further, the number of training data points is small, meaning that the prior has a relatively strong influence relative to the



(a) NLL (left is better) of all the classification datasets



(b) Accuracy (right is better) of all the classification datasets

Figure 5.11: Results for classification datasets. TGP stands for non input-dependent flows, PETGP stands for point estimate input-dependent flows (Standard Dropout) and BATGP stands for the Bayesian input-dependent flows (MC Dropout).

likelihood terms. In situations like this, a badly specified prior dominates the likelihood terms and adversely affects the predictive likelihoods.

To build on this claim we note that the TGP outperforms the GP on this dataset indicating that having a more expressive prior is beneficial. This coupled with the fact that we did not tune the prior $p(\mathbf{W})$ for our BNN, and that this is the only dataset in which the BATGP does not give a clear boost in performance, are consistent observations with the hypothesis of prior misspecification.

Finally, on `banknote`, we see that all the models provide a 100% accuracy, which means that the improvement in the NLL is coming from reducing the calibration gap, as the NLL is a proper scoring rule. Our model is making the predictions extreme towards the correct class, which is a desirable property if your data distribution doesn't present overlap between classes as already discussed in the first chapters of this thesis. Note however how in this case the BATGP still provides uncertainty in the predictions, avoiding the extreme $\{0, 1\}$ probability assignments.

5.7.3 Computational Performance of the TGP

As we have seen in the previous section, the TGP can match the 3-layer DGP performance at a fraction of the computational complexity. Figure 5.12 compares the training and prediction time required by the SVGP the BATGP and DGP of different layers. Even on the `energy` dataset, where the DGP has only 8 GP per layer, computation times of the 3-layer DGP are 3× (training) and 10× (prediction) that of the BATGP.

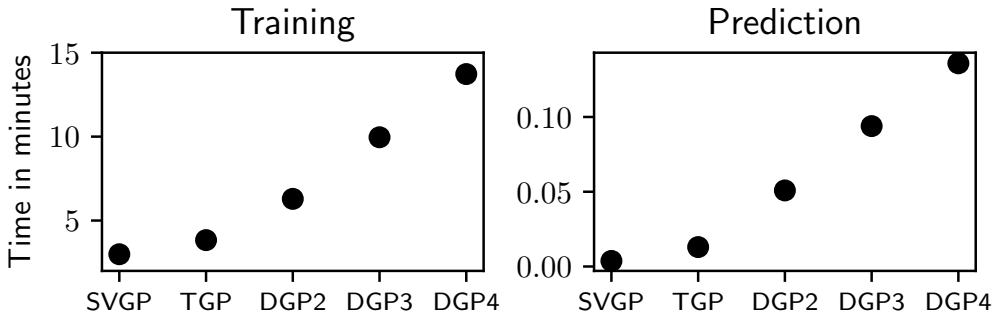
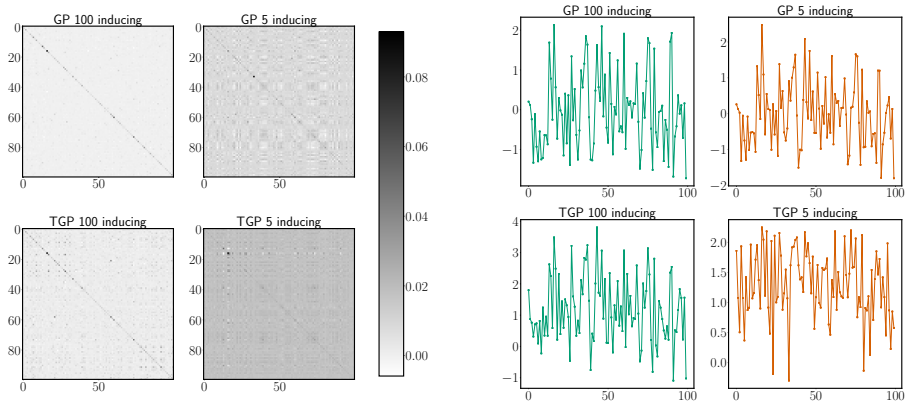


Figure 5.12: Average clock times for 100 runs with 1200 epochs on `energy`. Predictions use 100 samples from the posterior. The variance of training and prediction repetitions is negligible ($< 10^{-5}$).

This is true even though our implementation of the DGP fully exploits parallelization on a GPU cluster, while our current TGP implementation does not exploit potential parallelization across GP parameters. We further emphasize that our predictions use a 100 point quadrature integration rule per Monte Carlo sample, i.e. we use 100×100 integration points, in contrast to the DGP which only uses 100 Monte Carlo samples.

5.7.4 *Uncertainty handled by the* GP *and* TGP

In the UCI experiment section, we comment that sometimes the model with 5 inducing points provides similar results to the 100 inducing points models. Initially, we could think that the BNN is handling all the modeling power both in terms of uncertainty quantification and regressed values. In this subsection, we illustrate that this is not the case and that the modeling performance comes from a combination of the Neural Network and the GP. Note that the uncertainty provided by the GP is combined with the uncertainty provided by the BNN.



(a) Covariance from $q(\mathbf{f}_0)$ evaluated at 100 training points

(b) Mean from $q(\mathbf{f}_0)$ evaluated at 100 training points

Figure 5.13: This figure shows the mean and covariance from the GP variational distribution $q(\mathbf{f}_0)$ evaluated at 100 training points. As shown in the plot the covariance have not collapse to a point mass (i.e. the cells would have to be completely white) and the mean also change across training points. This means that the model has not learned to just output a constant value for $\mathbf{f}_0$ and model everything through the Neural Network.

To do so, we pick the `concrete` dataset, which is one of the datasets in which this effect is presented. For this dataset we plot the mean and covariance of $q(\mathbf{f}_0)$ at 100 random training locations in Figure 5.13. As we see in the plot, the mean and covariance from $q(\mathbf{f}_0)$ have not collapsed to a constant distribution. This means that the model has not learned to output a constant value for $\mathbf{f}_0$ and perform all the modeling through the input-dependent $\theta(\mathbf{W}, \mathbf{X}_n)$ model. Note however that understanding the specific role of the GP and the BNN in our model in terms of uncertainty quantification is something that we leave for future work.

5.7.5 Applications

Finally, we compare the proposed TGP with the variational V-WGP in two applications where we can encode inductive biases in the warping function. This will allow us to discuss the benefits of warping the likelihood (V-WGP) vs warping the prior (TGP), although in this chapter we do not investigate when is better to warp the likelihood vs when is better to warp the prior. In other words, we do not answer the question: is our data corrupted with non-Gaussian noise, or is our data modeled by a non-Gaussian process with a Gaussian observation model?. This is something we leave for future work.

Air Quality

Consider Particulate Matter of 2.5 μm size (PM25) in London (London 2020) as depicted in Figure 5.14. Measurements of PM25 are non-negative, exhibit periodic fluctuations due to vehicle traffic, and irregular peaks arising from weather conditions or traffic jams. Thus, we choose a SAL and Softplus (SP) composition (SAL+SP). This makes the TGP's latent function positive and guarantees $\mathbb{T}(\mathbf{Y}) \geq 0$ for the V-WGP.

The difference between methods is noticeable for low numbers of inducing points (see Figure 5.14 and Figure 5.15). As discussed before the V-WGP implicitly models $\mathbf{Y}$ with non-additive noise while the TGP transforms the prior, but models the noise additively. Hence, the TGP will attribute fluctuations to the underlying latent function, while the V-WGP is prone to absorb oscillations into the observation noise, as in Figure 5.14. Unsurprisingly, the TGP's fit is superior to that of the SVGP due to its additional flexibility. However, even though $\mathbb{G}_\theta$ is chosen so that $\mathbb{G}_\theta(\mathbf{f}_0) \geq 0$, the TGP assigns positive probability mass to PM25 being negative, since the observation model $p(\mathbf{Y}|\mathbf{f}_K)$ is Gaussian.

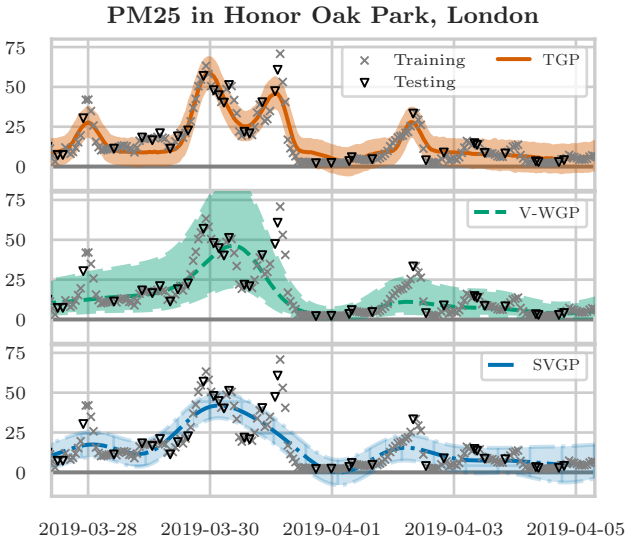


Figure 5.14: Model fits on PM25 with 5% of inducing points. The TGP's added flexibility provides the best fit. **Top**: the TGP with compositional SAL and SP flow. **Middle**: V-WGP with the same flow, but in reverse and applied to the likelihood. **Bottom**: SVGP.

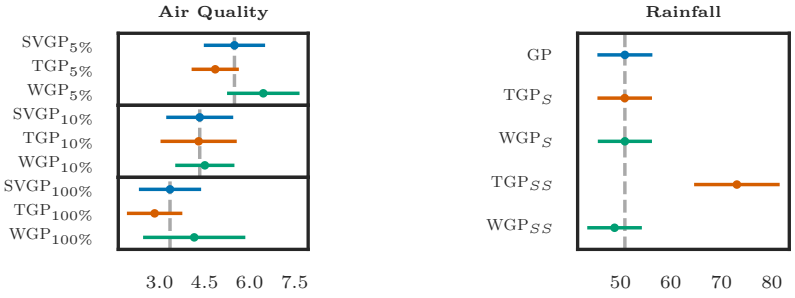


Figure 5.15: RMSE results (left is better) from the Air quality and Rainfall application. **Left**: Air quality experiments with 5%, 10% and 100% inducing points and a SAL plus SP flow. The TGP consistently outperforms the GP and V-WGP because it can better fit to irregular patterns in the data. **Right**: Rainfall experiments with 100% inducing points and SP flows (S) versus SAL plus SP flows (SS). When both the TGP and V-WGP use the more expressive SS flow, the V-WGP is superior, reflecting that the source of misspecification is the likelihood, not the prior.

Switzerland Rainfall

We also model daily rainfall in Switzerland (Dubois 2003), see Figure 5.16. As observations are non-negative, we again employ SAL+SP flows.

Unlike the TGP and SVGP, the V-WGP does not fit the latent function to peaks in the data and guarantees positive predictions. The resulting smoother fit is desirable and explains why the V-WGP's predictive performance in Figure 5.15 outperforms that of the TGP.

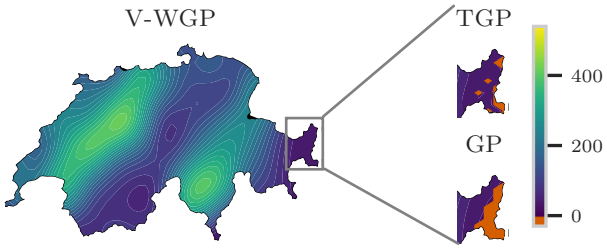


Figure 5.16: Spatial median predictions from a V-WGP, TGP and SVGP on the Switzerland daily rainfall dataset (units in 10 μm). The V-WGP (left) is guaranteed to have non-negative predictions. The TGP and SVGP (right) do not and predict negative rainfall in Graubünden.

5.8 Conclusions and Future work

This chapter introduces the Transformed Gaussian Process, a new flexible family of function priors that are constructed by warping samples from a GP using an invertible, input-dependent, and Bayesian transformation.

Many desirable properties have been outlined, including better performance than SVGP and DGP at a much lower computational cost. We have also derived a sparse variational inference algorithm which allows us to deploy this model in large scale applications.

This model can be used within inter-domain inducing point approximations (Lázaro-Gredilla et al. 2009), to improve multitask GPs (Bonilla et al. 2007), density estimation (Dutordoir et al. 2018), model calibration (Maroñas et al. 2020), and probabilistic dimensionality reduction (Titsias et al. 2010). We also think many applications can benefit from the proposed model, being the combination of V-WGP and TGP a clear line to explore.

Chapter 6

Conclusions and Future Work

This thesis has focused on how to make modern state-of-the-art machine learning models more reliable. We have done this by targeting the final objective in different ways.

We started by introducing Bayes Decision Rule, a framework to make decisions under uncertainty. We then motivated the need of recovering this uncertainty for a machine learning model to be calibrated. We saw how this idea could be used to justify why any Data Augmentation technique is not necessarily calibrated, in contrast to many recent works showing the opposite direction; and to show how a super simple loss function can fix this miscalibration. From this work (and this is a personal opinion) the most interesting thing is not the loss function proposed, but how do we arrive at this loss function. To my knowledge, this line of research has not been really exploited, at least in this way, and I am planning to pursue some research in this direction. This could let us not only provide better-calibrated models but also improve the error probability of the final application, guaranteed by targeting the data distribution more reliably.

This thesis has also explored another line of research which has attracted the attention of the community in the last years. This corresponds to decoupled techniques that aim at fixing the miscalibration of a Deep Neural Network. To this end, this thesis proposes the use of Bayesian Neural Networks. We

evaluated the proposed approach, showing that is a promising line of research, but with a lot of work to do. We have started to work in this direction already, analyzing some of the problems outlined in this thesis such as the source of accuracy degradation the BNN suffers from. We are also studying if the robustification against prior misspecification can improve upon the limitations shown by this work. The work from my colleague Jeremias was not just a reading that I recommend to anyone starting with Bayesian Inference, but also provide me with the light on the path I need to follow to pursue future directions on this line of research. Beyond this, I would like to explore the design of hierarchical priors for a BNN applied to this particular task.

Finally, we showed how can we design non-stationary non-Gaussian priors directly over the space of functions, that can solve many problems that Bayesian Neural Networks suffer from when specifying a prior distribution over its parameters. The proposed approach provides a principled way of constructing more expressive priors than the typical Gaussian Process, but at a fraction of the computational cost of other proposed models. On this site I would like to explore applications of the TGP or new models that can be build on top of the tools developed for this work.

Beyond the future work outlined in this chapter, the other thing I have started to learn early this 2021 year is the mathematical field of topology. These years have served me to discover one of the (for me) most unknown but most useful fields in mathematics for my particular interests. My objective is to learn about differential geometry, topology and information geometry during these years although I am not planning to perform research on this direction. Other lines I would like to explore in the future are stochastic differential equations and sum-product networks.

Appendix A

Additional Calibration Results in Chapter 3

A.1 Additional Loss Analysis

In this appendix, we first provide additional insights on the loss proposed which might explain some of the behaviour presented in the main text.

A.1.1 Accuracy Improvement

First, it should be noted that this cost presents other desirable properties that aim at improving regularization. First, consider a set of samples lying in the confidence range $[0.6, 0.7]$. If the accuracy of these samples is located in this range then our loss function will encourage the model to adjust these confidences to be as close as possible to the accuracy. Second, if the accuracy provided by the model has a value over this range, e.g 80%, then the model will raise these confidences to recover a calibrated model. It should be noted that in this case, our loss function will not change the accuracy as we are just pushing upwards the confidence of the samples which are originally correctly/incorrectly assigned, and thus the decision of which class should be assigned to each sample remains intact. Third, consider the same set of sam-

ples but with a provided accuracy of 40%. Our loss function will encourage this set of samples to reduce its confidences. It is clear that reducing this confidence has to be done at the cost of raising the confidence towards other classes. By doing this, we have a chance of changing the decision made by the model towards another class, thus helping to improve the discrimination of the model and consequently raising the accuracy. This serves as a possible explanation of the boost in accuracy that the CARS dataset present (over $1-3\%$ w.r.t. the baseline model see Table 3.1).

A.1.2 Further discussion about hyperparameter search

In this subsection, we provide a further discussion about the hyperparameter search we have performed using a ResNet-18 on each dataset, which is then used by the rest of the models.

Note that this way of searching for hyperparameters is clearly not optimal. We found, however, that in general, the extrapolated hyperparameter performed well (as shown in the results and detailed in Github). This means that there is a great chance to improve the results by performing this search on each model individually (which is what one does in a real application).

However, sometimes, we experimented accuracy degradation in the training set. This is because a pathological solution of optimizing the ARC loss is by setting the parameters to output the data prior probability,. This solution evaluates the ARC loss to 0, but at the cost of parameterizing a useless prior classifier. How do we solve these particular cases?. As an example consider, for instance, that on the ResNet-18 we found that the optimal hyperparameter was $\beta = 42$, but when training a DenseNet-121 this hyperparameter degraded the accuracy over the training set at the cost of providing perfect calibration. When this effect was observed we just picked the next hyperparameter that provided the next top performance over the ResNet-18 ($\beta = 40$ in this case); until the training accuracy was not degraded[1].

On the other hand, in CIFAR100 with Mixup we found this way of searching for the hyperparameter not to be as effective for the models without dropout. As provided in the specific results for each model trained on CIFAR100 (see Github), we can see that all the models except ResNet-18 improve calibration when using Mixup. Thus, we cannot expect the hyperparameter of the ResNet-18 to extrapolate as it happens with other datasets. This was observed by training any of the deeper models with a validation set. To solve this, we

[1]Note that we are not cheating since we always look at the training set.

Table A.1: This table shows different calibration metrics for average results. ACC in (%), MCE in (%), BS ×100 and LS

	CIFAR10				CIFAR100				SVHN				Birds				Cars			
Model	ACC	MCE	BS	LS	ACC	MCE	BS	LS	ACC	MCE	BS	LS	ACC	MCE	BS	LS	ACC	MCE	BS	LS
B	94.76	2.16	0.86	0.23	77.21	5.20	0.35	1.08	96.32	1.86	0.62	0.17	78.51	0.58	0.17	1.04	86.74	0.56	0.10	0.52
B+M	96.01	2.88	0.65	0.18	80.04	0.67	0.29	0.79	96.41	2.76	0.63	0.18	79.63	1.49	0.18	1.11	86.67	1.81	0.13	0.71
M	94.24	1.06	0.89	0.21	72.68	0.61	0.38	0.98	96.28	1.12	0.61	0.17	78.78	0.46	0.17	1.02	86.83	0.59	0.10	0.52
M+M	91.90	2.73	1.33	0.32	78.52	1.18	0.31	0.86	96.59	1.46	0.59	0.16	79.99	1.23	0.17	1.07	86.03	1.28	0.12	0.67
A	94.85	2.13	0.85	0.23	77.04	4.78	0.35	1.05	96.26	1.16	0.62	0.17	78.52	0.65	0.17	1.04	87.78	1.32	0.10	0.51
A+M	95.90	0.78	0.67	0.17	79.84	0.54	0.29	0.80	96.02	1.11	0.64	0.16	79.74	0.65	0.15	0.82	89.63	1.70	0.08	0.44

Table A.2: This table shows different calibration metrics for the best model per task and technique. ACC in (%), MCE in (%), BS ×100 and LS

	CIFAR10				CIFAR100				SVHN				Birds				Cars			
Model	ACC	MCE	BS	LS	ACC	MCE	BS	LS	ACC	MCE	BS	LS	ACC	MCE	BS	LS	ACC	MCE	BS	LS
B	95.35	1.23	0.65	0.15	79.79	2.36	0.29	0.81	97.07	0.18	0.48	0.12	80.31	1.01	0.16	0.98	89.13	0.42	0.089	0.45
B+M	97.19	3.11	0.47	0.14	82.34	0.46	0.26	0.70	96.97	2.55	0.53	0.16	82.09	1.12	0.15	0.97	89.45	1.84	0.110	0.61
M	95.58	0.46	0.67	0.15	74.98	1.18	0.36	0.92	96.90	0.34	0.49	0.13	80.64	0.99	0.16	0.97	89.40	0.35	0.087	0.44
M+M	97.02	0.73	0.45	0.11	81.31	0.70	0.28	0.74	97.17	1.36	0.50	0.14	82.41	1.05	0.15	0.96	88.47	1.14	0.105	0.59
A	95.99	1.02	0.62	0.14	80.77	2.25	0.28	0.79	97.08	0.17	0.47	0.12	80.32	1.17	0.16	0.99	90.09	1.21	0.080	0.42
A+M	97.09	0.39	0.48	0.12	82.02	0.31	0.26	0.72	96.82	1.75	0.51	0.14	82.45	0.34	0.13	0.69	91.13	0.91	0.078	0.42

simply perform a hyper-parameter search over one of the deeper models in which Mixup showed great calibration performance, and use this parameter with the rest of the models. Due to computational limitations, we did not perform such an exhaustive search as we did with the ResNet-18, and just select a subset of the hyperparameters based on the previous wider analysis performed over the ResNet-18.

A.2 Additional Results

In addition, we present the rest of calibration metrics of the corresponding experiments in Table 3.1 and Table 3.2. These are given in Table A.1 and Table A.2.

Appendix B

Experimental Details of the Transformed Gaussian Process

In this appendix, we provide additional details on the experiments. We mainly provide training hyperparameters and experiment configurations. Details about the kind of warping functions that can be used are provided in e.g. (Rios et al. 2019; Maroñas et al. 2021b). Details about the type of warping functions used in each experiment are provided in Github.

B.1 Flow Parameters Initialization

One of the critical points in the performance of the TGP is the initialization of the model. We propose two different ways to initialize the flow hyperparameters.

B.1.1 Initializing Flows from Data

In this section, we describe an initialization scheme that attempts to learn flow parameters that Gaussianize the prior. In the derivation, we approximate the data as being noise-free and so in practice, this may also be used for the likelihood transformations of the WGP. Ideally, we would want to learn a normalizing flow $\mathbb{G}(\cdot)$ that transforms a standard Gaussian φ to the true prior $p(\mathbf{f})$:

$$\varphi(\mathbb{G}^{-1}(\mathbf{f}_K))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{f}_K} = p(\mathbf{f}) \tag{B.1}$$

In practice we do not have access to the true prior but instead observations $\mathbf{Y}$. By using $\mathbf{Y}$ as approximate samples from $p(\mathbf{f})$ we can then optimize $\mathbb{G}$ to approximately satisfy Equation B.1. To optimize we directly minimize the KLD divergence between $p(\mathbf{f})$ and $\varphi(\mathbb{G}^{-1}(\mathbf{f}_K))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{f}_K}$:

$$\begin{aligned} &\text{KLD}\left[p(\mathbf{f}) \;||\; \varphi(\mathbb{G}^{-1}(\mathbf{f}_K))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{f}_K}\right] = \\ &\mathbb{E}_{p(\mathbf{f})}\left[\log \varphi(\mathbb{G}^{-1}(\mathbf{f}_K))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{f}_K}\right] - \mathbb{E}_{p(\mathbf{f})}\left[p(\mathbf{f})\right] \end{aligned} \tag{B.2}$$

The second term is constant w.r.t. the flow parameters and hence we only need to consider and optimize the first term. Because we have assumed that the $\mathbf{Y}$ are approximate samples from the true prior we write:

$$\begin{aligned} &\mathbb{E}_{p(\mathbf{f})}\left[\log \varphi(\mathbb{G}^{-1}(\mathbf{f}_K))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{f}_K}\right] \approx \\ &\sum_{n=1}^{N} \log \varphi(\mathbb{G}^{-1}(\mathbf{Y}_n))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{Y}_n} = \sum_{n=1}^{N} \log \varphi(\mathbb{G}^{-1}(\mathbf{Y}_n))\left(\frac{\partial \mathbb{G}}{\partial \mathbf{Y}_n}\right)^{-1} \end{aligned} \tag{B.3}$$

and the final initialization optimization procedure is:

$$\underset{\theta}{\text{argmin}} \sum_{n=1}^{N} \log \varphi(\mathbb{G}^{-1}(\mathbf{Y}_n))\frac{\partial \mathbb{G}^{-1}}{\partial \mathbf{Y}_n} \tag{B.4}$$

A similar derivation is used by (Papamakarios et al. 2017) but in a different context. Note that the procedure described here is basically the formalization in terms of divergences of the maximum likelihood training criteria derived from the KLD, already done in chapter 3. In this subsection, we have followed the same derivation as in the original publication (Maroñas et al. 2021b).

B.1.2 Initializing Flows approximately to Identity

In this section, we provide details on how we initialize flows close to the identity function. Many transformations can already recover identity but for those that cannot this method provides an effective and simple way to initialize them. To find these parameters we simply generate observations from the line $\mathbf{Y} = \mathbf{X} : \{\mathbf{X}_n, \mathbf{Y}_n\}_{n=1}^N$ and minimize the mean squared loss of the flow mapping from $\mathbf{X}$ to $\mathbf{Y}$:

$$\text{argmin}_\theta \frac{1}{N} \sum_{n=1}^{N} (\mathbf{Y}_n - \mathbb{G}(\mathbf{X}_n))^2 \tag{B.5}$$

B.1.3 Initialization of Input-dependent flows

To initialize input-dependent flows we first initialize standard (non-input-dependent) flow parameters $\hat{\theta}$ by any of the procedures described above. Then, we turn the parameters into input-dependent and initialize the Neural network parameters to match the values learned in the first step. This is done through stochastic gradient optimization, i.e. by first sampling a minibatch from the data distribution, and then minimize the empirical MSE loss between $\text{DNN}(\mathbf{X})$ and $\hat{\theta}$.

B.2 Experiment Details

B.2.1 Regression and Classification

In the black-box experiments, we explore the performance of TGP across many UCI datasets (Lichman 2013). The performance measures are evaluated by employing random 10 fold train-test partitions and reporting average results plus standard error. This is done for all the datasets except Year and Avila (because the test partition is already provided) and Airline, where we just use a random 1 fold partition following previous works e.g. (Salimbeni et al.

2017). We perform flow selection by running each of the candidate models using random validation splits. We use one validation split on the first and second random fold partitions, except for Year and Airline where we only use one. This selection is done for 100 inducing points. The selected model is then used across all the experiments reported, including the experiments with fewer inducing points. We use 100 quadrature points and 1 Monte Carlo samples to evaluate the ELBO during training and 100 Monte Carlo samples to evaluate the posterior predictive.

To initialize our models we follow (Salimbeni et al. 2017). We use RBF kernels with parameters initialized to 2.0. The inducing points are initialized using the best of 10 KMeans runs except for Year and Airline where we just use 1 run. We use a whitened representation of inducing points and initialize the variational parameters to $\mathbf{m} = 0$ and $\mathbf{S} = 1^{-5}I$. The DGP have an additional white noise kernel added to the RBF in each hidden layer, with the noise parameter initialized to 1^{-6}. The noise parameter of the Gaussian likelihood is initialized to 0.05 for the regression experiments. For classification, we use a noise-free latent function and Bernoulli/Categorical likelihoods for Binary/-Multiclass problems. We use probit and softmax link functions respectively. We use Adam optimizer with a learning rate of 0.01. The flow initializers are run over 2000 epochs for the identity initializer and over 2000 (rest-of-datasets) or 20 (Year-Airline) epochs for input-dependent flows, to match the learned parameters in the previous initialization step. In our experiments however, we observed that the flow could be initialized with fewer epochs. We have tried a set of different combinations of flows as described in the appendix of the original work (Maroñas et al. 2021b). This includes different flow lengths and different number of flows in the linear combinations. For input-dependent flows we just tried the SAL flow with lengths 1 and 3; where input dependency is encoded just in the non-linear flow (i.e the sinh-arcsinh). For these experiments, we focus on exploring different architectures of the Neural Networks. We search over $\{1, 2\}$ hidden layers, $\{25, 50\}$ neurons per layer, $\{0.25, 0.5, 0.75\}$ dropout probabilities, batch normalization and ReLu and Tanh activations. We found that any of these possible combinations could work except the use of Batch Normalization, and that most of these combinations could be successfully optimized using the default optimizer, although some combinations suggested that a lower learning rate was needed to make optimization stable (these combinations were discarded as we wanted to show robustness against optimizer hyperparameter search). The prior over the neural network parameters is kept fixed and is introduced in the model by fixing a 1^{-5} weight decay in the optimizer ($\lambda = 1^{-5}$). In our Github, we provide additional information about the model selection process and the final selected models.

All of our models were optimized for 15000 epochs for all datasets except Year and Airline where we use 200 epochs. Each epoch corresponds to a full pass over the dataset. For classification, we follow (Hensman et al. 2015) and freeze the covariance parameters before learning everything end to end. This is done for the first 2000 epochs.

For experiments using less than 100 inducing points, we use the same flow architecture selected from the validation sets and optimizer hyperparameters as the corresponding 100 inducing point experiment. The performance obtained highlights that our approach is somewhat robust to hyperparameter selection. On just one dataset we observe that this extrapolation was suboptimal and that the algorithm diverge. Those results are not reported in this appendix and correspond to 5 inducing points, input-dependent flows and `naval` dataset. We attribute this fact to not having performed model selection and optimizer hyperparameter search for each set of inducing points specifically. On the other hand, if any of the experiments carried out failed due to numerical errors (e.g Cholesky decomposition) we increase the standard amount of jitter added by Gpytorch from 1^{-8} to 1^{-6}. This is just needed on some train-test folds and some datasets only when using less inducing points. In general, we found that our experiments were quite stable.

B.2.2 Real World Experiments

For both real world experiments, we consider 2 different seeds, shuffle the observations and run 5-fold cross-validation across 2 different optimization schemes. The first optimization scheme optimizes both the variational and hyperparameters jointly. The second holds the likelihood noise fixed for 60% of iterations. This is to help avoid early local minimum that causes the models to underfit and explain the observations as noise.

For all models, we use RBF kernels with length scales initialized to 0.1, and Gaussian likelihoods with noise initialized to 1.0. We optimize the whitened variational objective using Adam optimizer with a learning rate of 0.01.

Air Quality

We used data from the London Air Quality Network London 2020 and we focus on site HP5 (Honor Oak Park, London) using 1 month of PM25 data (731 observations, date range 03/15/2019 - 04/15/2019). Because the observations are non-negative bounded we only consider the following positive enforcing flow: SAL+SP initialized from data.

We shuffle observations and run 5-fold cross-validation across 5%, 10% and 100% of inducing points and optimize each for a total of 10000 epochs. We compute means and standard deviations across all folds and seeds.

Rainfall

The Switzerland rainfall uses data collected on the 8th of May 1986. Because all the observations are positively bounded we again use positive enforcing flows. We consider: SP, SAL+SP (from data), SAL+SP (from identity), SAL+SAL+SP (from identity), SAL+SAL+SP (from data).

We optimize for a total of 20000 epochs and compute means and standard deviations across all flows, folds and seeds.

Bibliography

Agakov, Felix V. et al. (2004). "An Auxiliary Variational Method". In: *Neural Information Processing, 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25, 2004, Proceedings*, pp. 561–566 (cit. on p. 81).

Alvarez-Balanya, Sergio (2020). "Técnicas expresivas de calibración para clasificadores multiclase". In: *Master Thesis*. Universidad Autónoma de Madrid (cit. on pp. 67, 68).

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc. ISBN: 0198538642 (cit. on p. 25).

— (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387310738 (cit. on pp. 28, 68, 86).

Bissiri, P. G., C. C. Holmes, and S. G. Walker (2016). "A general framework for updating belief distributions". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 78.5, pp. 1103–1130 (cit. on pp. 34, 61).

Blundell, Charles et al. (2015). "Weight Uncertainty in Neural Network". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1613–1622 (cit. on p. 66).

Bojarski, Mariusz et al. (2016). "End to End Learning for Self-Driving Cars". In: *In Nips Depp Learning Synopsium Workshop* (cit. on p. 2).

Bonilla, Edwin V., Kian Ming A. Chai, and Christopher K. I. Williams (2007). "Multi-Task Gaussian Process Prediction". In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. Vancouver, British Columbia, Canada: Curran Associates Inc., 153–160. ISBN: 9781605603520 (cit. on p. 120).

Bröcker, Jochen (2009). "Reliability, sufficiency, and the decomposition of proper scores". In: *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography* 135.643, pp. 1512–1519 (cit. on pp. 15, 16).

Brooks, S. et al. (2011). *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. CRC Press. ISBN: 9781420079425 (cit. on pp. 2, 58, 59).

Cao, Q. et al. (2018). "VGGFace2: A dataset for recognising faces across pose and age". In: *International Conference on Automatic Face and Gesture Recognition* (cit. on p. 71).

Carpenter, Bob et al. (2017). "Stan: A probabilistic programming language". In: *Journal of statistical software* 76.1 (cit. on p. 68).

Caruana, Rich et al. (2015). "Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 1721–1730. ISBN: 9781450336642 (cit. on p. 2).

Chapelle, Olivier et al. (2001). "Vicinal Risk Minimization". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press (cit. on pp. 29, 36).

Chen, Tianqi, Emily Fox, and Carlos Guestrin (2014). "Stochastic Gradient Hamiltonian Monte Carlo". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, pp. 1683–1691 (cit. on p. 64).

Chen, Yunpeng et al. (2017). "Dual Path Networks". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 4467–4475 (cit. on p. 71).

Cohen, Ira and Moises Goldszmidt (2004). "Properties and Benefits of Calibrated Classifiers". In: *Knowledge Discovery in Databases: PKDD 2004*. Ed. by Jean-François Boulicaut et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 125–136 (cit. on p. 21).

Damianou, Andreas and Neil D. Lawrence (2013). "Deep Gaussian Processes". In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Carlos M. Carvalho and Pradeep Ravikumar. Vol. 31. Proceedings of Machine Learning Research. Scottsdale, Arizona, USA: PMLR, pp. 207–215 (cit. on pp. 5, 92).

Degroot, M. and S. Fienberg (1983). "The Comparison and Evaluation of Forecasters." In: *The Statistician* 32, pp. 12–22 (cit. on pp. 2, 15).

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255 (cit. on p. 9).

Dietterich, Thomas G. (2000). "Ensemble Methods in Machine Learning". In: *Proceedings of the First International Workshop on Multiple Classifier Systems*. MCS '00. Berlin, Heidelberg: Springer-Verlag, 1–15. ISBN: 3540677046 (cit. on p. 2).

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). "Density estimation using Real NVP". In: *International Conference on Learning Representations, ICLR* (cit. on p. 97).

Dubois, G (2003). *Mapping radioactivity in the environment : Spatial Interpolation Comparison 97*. Luxembourg: Office for Official Publications of the European Communities. ISBN: 92-894-5371-0 (cit. on p. 120).

Duda, Richard O., Peter E. Hart, and David G. Stork (2000). *Pattern Classification (2nd Edition)*. USA: Wiley-Interscience. ISBN: 0471056693 (cit. on p. 9).

Dusenberry, Michael et al. (2020). "Efficient and Scalable Bayesian Neural Nets with Rank-1 Factors". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 2782–2792 (cit. on p. 57).

Dutordoir, Vincent et al. (2018). "Gaussian Process Conditional Density Estimation". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 2385–2395 (cit. on p. 120).

Duvenaud, David (2014). "Automatic Model Construction with Gaussian Processes". PhD thesis. Computational and Biological Learning Laboratory, University of Cambridge (cit. on p. 85).

Eidinger, Eran, Roee Enbar, and Tal Hassner (Dec. 2014). "Age and Gender Estimation of Unfiltered Faces". In: *Trans. Info. For. Sec.* 9.12, pp. 2170–2179. ISSN: 1556-6013. DOI: 10.1109/TIFS.2014.2359646 (cit. on p. 71).

Gal, Yarin and Zoubin Ghahramani (2016). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1050–1059 (cit. on pp. 46, 56, 71, 79, 104, 108, 109).

Gal, Yarin, Jiri Hron, and Alex Kendall (2017). "Concrete Dropout". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 3581–3590 (cit. on p. 79).

Gneiting, Tilmann and Adrian E. Raftery (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation". In: *Journal of the American Statistical Association* 102, pp. 359–378 (cit. on pp. 12, 14).

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton (2013). "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947 (cit. on p. 1).

Guo, Chuan et al. (2017). "On Calibration of Modern Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*.

Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1321–1330 (cit. on pp. 22, 36, 42, 45, 49, 55, 56, 71–73, 75, 77, 79).

Havasi, Marton et al. (2021). "Training independent subnetworks for robust prediction". In: *International Conference on Learning Representations* (cit. on p. 56).

He, Kaiming et al. (2016a). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on pp. 1, 46, 71).

— (2016b). "Identity Mappings in Deep Residual Networks". In: *ECCV* (cit. on p. 71).

Heinonen, Markus et al. (2016). "Non-Stationary Gaussian Process Regression with Hamiltonian Monte Carlo". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, pp. 732–740 (cit. on p. 92).

Hendrycks, Dan and Thomas G. Dietterich (2019a). "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *International Conference on Learning Representations* (cit. on p. 38).

Hendrycks, Dan, Kimin Lee, and Mantas Mazeika (2019b). "Using Pre-Training Can Improve Model Robustness and Uncertainty". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2712–2721 (cit. on pp. 39, 48).

Hendrycks, Dan et al. (2019c). "Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. (cit. on p. 39).

Hendrycks, Dan et al. (2020). "AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty". In: *International Conference on Learning Representations* (cit. on pp. 37, 38).

Hensman, James, Nicolò Fusi, and Neil D. Lawrence (2013). "Gaussian Processes for Big Data". In: *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*. UAI'13. Bellevue, WA: AUAI Press, 282–290 (cit. on pp. 89, 102).

Hensman, James, Alexander G. de G. Matthews, and Zoubin Ghahramani (2015). "Scalable Variational Gaussian Process Classification." In: *AISTATS*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. JMLR Workshop and Conference Proceedings. JMLR.org (cit. on p. 131).

Hoffman, Matthew, Alexey Radul, and Pavel Sountsov (2021). "An Adaptive-MCMC Scheme for Setting Trajectory Lengths in Hamiltonian Monte Carlo". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 3907–3915 (cit. on p. 64).

Hoffman, Matthew D. and Andrew Gelman (Jan. 2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *J. Mach. Learn. Res.* 15.1, 1593–1623. ISSN: 1532-4435 (cit. on p. 64).

Holmes, C. C. and S. G. Walker (Mar. 2017). "Assigning a value to a power likelihood in a general Bayesian model". In: *Biometrika* 104.2, pp. 497–503 (cit. on p. 68).

Hu, Jie et al. (2018). "Squeeze-and-Excitation Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141 (cit. on p. 71).

Huang, Gao et al. (2017). "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269 (cit. on pp. 46, 71).

Izmailov, Pavel et al. (2021). *What Are Bayesian Neural Network Posteriors Really Like?* arXiv: `2104.14421 [cs.LG]` (cit. on pp. 57, 64).

Jewson, Jack, Jim Q. Smith, and Chris Holmes (2018). "Principles of Bayesian Inference Using General Divergence Criteria". In: *Entropy* 20.6. ISSN: 1099-4300 (cit. on pp. 33, 61).

Kendall, Alex and Yarin Gal (2017). "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. (cit. on p. 57).

Kingma, Diederik P. and Jimmy Ba (2015a). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 71).

Kingma, Diederik P and Max Welling (2014). "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations* (cit. on p. 66).

Kingma, Durk P, Tim Salimans, and Max Welling (2015b). "Variational Dropout and the Local Reparameterization Trick". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc. (cit. on pp. 46, 67).

Knoblauch, Jeremias, Jack Jewson, and Theodoros Damoulas (2019). *Generalized Variational Inference: Three arguments for deriving new Posteriors* (cit. on pp. 33, 34, 60–64, 67, 68, 70, 81, 104, 115).

Kober, Jens, J. Andrew Bagnell, and Jan Peters (2013). "Reinforcement Learning in Robotics: A Survey". In: *International Journal of Robotics Research* 32.11, pp. 1238–1274. DOI: 10.1177/0278364913495721 (cit. on p. 2).

Krause, Jonathan et al. (2013). "3D Object Representations for Fine-Grained Categorization". In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia (cit. on pp. 46, 70).

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton (2009a). "CIFAR-10 (Canadian Institute for Advanced Research)". In: (cit. on pp. 46, 70).

— (2009b). "CIFAR-100 (Canadian Institute for Advanced Research)". In: (cit. on pp. 46, 70).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in*

Neural Information Processing Systems. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. (cit. on p. 1).

Kuleshov, Volodymyr, Nathan Fenner, and Stefano Ermon (2018). "Accurate Uncertainties for Deep Learning Using Calibrated Regression". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 2796–2804 (cit. on pp. 56, 57, 79, 80).

Kumar, Aviral, Sunita Sarawagi, and Ujjwal Jain (2018). "Trainable Calibration Measures for Neural Networks from Kernel Mean Embeddings". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 2805–2814 (cit. on pp. 45, 49, 56, 57, 71, 79, 80).

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. (cit. on pp. 56, 57, 71, 79).

Lázaro-Gredilla, Miguel (2012). "Bayesian Warped Gaussian Processes". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. (cit. on p. 93).

Lázaro-Gredilla, Miguel and Aníbal Figueiras-Vidal (2009). "Inter-domain Gaussian Processes for Sparse Inference using Inducing Features". In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio et al. Curran Associates, Inc., pp. 1087–1095 (cit. on p. 120).

Lee, Kimin et al. (2018). *Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples* (cit. on p. 79).

Lichman, Moshe (2013). *UCI machine learning repository* (cit. on pp. 109, 129).

London, Imperial College (2020). *Londonair - London air quality network (LAQN)*. `https://www.londonair.org.uk` (cit. on pp. 118, 132).

Louizos, Christos, Karen Ullrich, and Max Welling (2017a). "Bayesian Compression for Deep Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. (cit. on p. 29).

Louizos, Christos and Max Welling (2017b). "Multiplicative Normalizing Flows for Variational Bayesian Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 2218–2227 (cit. on p. 81).

MacKay, David (1992). "Bayesian Methods for Adaptative Models". PhD thesis. California Institute of Technology (cit. on pp. 33, 87).

MacKay, David J. C. (2002). *Information Theory, Inference and Learning Algorithms*. USA: Cambridge University Press. ISBN: 0521642981 (cit. on p. 87).

Mariet, Zelda E et al. (2021). "Distilling Ensembles Improves Uncertainty Estimates". In: *Third Symposium on Advances in Approximate Bayesian Inference* (cit. on p. 56).

Maroñas, Juan, Roberto Paredes, and Daniel Ramos (2019). "Generative Models for Deep Learning with Very Scarce Data". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Ed. by Ruben Vera-Rodriguez, Julian Fierrez, and Aythami Morales. Cham: Springer International Publishing, pp. 20–28 (cit. on p. 6).

Maroñas, Juan, Roberto Paredes, and Daniel Ramos (2020). "Calibration of deep probabilistic models with decoupled bayesian neural networks". In: *Neurocomputing* 407, pp. 194–205. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2020.04.103` (cit. on pp. 4, 5, 49, 54, 70, 120).

Maroñas, Juan, Daniel Ramos, and Roberto Paredes (2021a). "On Calibration of Mixup Training for Deep Neural Networks". In: *Structural, Syntactic, and Statistical Pattern Recognition*. Ed. by Andrea Torsello et al. Cham: Springer International Publishing, pp. 67–76 (cit. on pp. 4, 5, 35).

Maroñas, Juan et al. (2021b). "Transforming Gaussian Processes With Normalizing Flows". In: *Proceedings of The 24th International Conference on*

Artificial Intelligence and Statistics. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 1081–1089 (cit. on pp. 4, 5, 83, 127, 129, 130).

Murphy, Kevin P (2021). *Machine learning: a probabilistic perspective*. Cambridge, MA (cit. on p. 10).

Naeini, Mahdi Pakdaman, Gregory F. Cooper, and Milos Hauskrecht (2015). "Obtaining Well Calibrated Probabilities Using Bayesian Binning". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2901–2907. ISBN: 0262511290 (cit. on pp. 23, 55).

Netzer, Yuval et al. (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011* (cit. on pp. 46, 71).

Niculescu-Mizil, Alexandru and Rich Caruana (2005). "Predicting Good Probabilities with Supervised Learning". In: *Proceedings of the 22nd International Conference on Machine Learning*. ICML '05. Bonn, Germany: Association for Computing Machinery, 625–632. ISBN: 1595931805 (cit. on p. 55).

Nixon, Jeremy et al. (2019). "Measuring Calibration in Deep Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (cit. on p. 24).

Paciorek, Christopher and Mark Schervish (2004). "Nonstationary Covariance Functions for Gaussian Process Regression". In: *Advances in Neural Information Processing Systems*. Ed. by S. Thrun, L. Saul, and B. Schölkopf. Vol. 16. MIT Press (cit. on p. 92).

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 2338–2347 (cit. on p. 129).

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on p. 40).

Platt, John C. (1999). "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods". In: *ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, pp. 61–74 (cit. on p. 54).

Pollastri, Federico et al. (2021a). "A deep analysis on high-resolution dermoscopic image classification". In: *IET Computer Vision* (cit. on p. 6).

Pollastri, Federico et al. (2021b). "Confidence Calibration for Deep Renal Biopsy Immunofluorescence Image Classification". In: *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 1298–1305 (cit. on pp. 2, 3, 5).

Quiñonero-Candela, Joaquin and Carl Edward Rasmussen (2005). "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *Journal of Machine Learning Research* 6.65, pp. 1939–1959 (cit. on p. 89).

Ramos, Daniel, Juan Maroñas, and Jose Almirall (2021). "Improving Calibration of Forensic Glass Comparisons by Considering Uncertainty in Feature-Based Elemental Data". In: *Chemometrics and Intelligent Laboratory Systems*, p. 104399. ISSN: 0169-7439. DOI: `https://doi.org/10.1016/j.chemolab.2021.104399` (cit. on pp. 2, 6).

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press. ISBN: 026218253X (cit. on pp. 2, 85).

Rezende, Danilo and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1530–1538 (cit. on pp. 81, 93).

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, pp. 1278–1286 (cit. on p. 66).

Rios, Gonzalo (2020). *Transport Gaussian Processes for Regression.* arXiv: 2001.11473 [stat.ML] (cit. on p. 97).

Rios, Gonzalo and Felipe Tobar (2019). "Compositionally-warped Gaussian processes". In: *Neural Networks* 118, 235–246. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.06.012 (cit. on pp. 94, 106, 127).

Salimbeni, Hugh and Marc Deisenroth (2017). "Doubly Stochastic Variational Inference for Deep Gaussian Processes". In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. (cit. on pp. 93, 129, 130).

Sandler, Mark et al. (2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 71).

Seo, Seonguk, Paul Hongsuck Seo, and Bohyung Han (2019). "Learning for Single-Shot Confidence Calibration in Deep Neural Networks Through Stochastic Inferences". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 56, 79).

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference On Learning Representations* (cit. on p. 71).

Sklar, Abe (1959). "Fonctions de répartition à n dimensions et leurs marges". In: *Publications de l'Institut de Statistique de l'Université de Paris* 8, pp. 229–231 (cit. on pp. 95, 97).

Snelson, Edward and Zoubin Ghahramani (2006). "Sparse Gaussian Processes using Pseudo-inputs". In: *Advances in Neural Information Processing Systems.* Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press (cit. on p. 89).

Snelson, Edward, Carl Edward Rasmussen, and Zoubin Ghahramani (2003). "Warped Gaussian Processes". In: *Proceedings of the 16th International Conference on Neural Information Processing Systems.* NIPS'03. Whistler, British Columbia, Canada: MIT Press, 337–344 (cit. on pp. 93, 106, 108).

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958 (cit. on pp. 46, 71, 108, 109).

Szegedy, Christian et al. (2014). "Intriguing properties of neural networks". In: *International Conference on Learning Representations* (cit. on pp. 29, 36, 42, 71).

Szegedy, Christian et al. (2016). "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826 (cit. on p. 49).

Tan, Mingxing and Quoc Le (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6105–6114 (cit. on p. 49).

Thulasidasan, Sunil et al. (2019). "On Mixup Training: Improved Calibration and Predictive Uncertainty for Deep Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. (cit. on pp. 37, 39, 48, 49).

Titsias, Michalis (2009). "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by David van Dyk and Max Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, pp. 567–574 (cit. on pp. 89, 90).

Titsias, Michalis and Neil D. Lawrence (2010). "Bayesian Gaussian Process Latent Variable Model". In: ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, pp. 844–851 (cit. on p. 120).

Vapnik, V. N. and A. Y. Chervonenkis (1971). "On the uniform convergence of relative frequencies of events to their probabilities". In: *Theory of Probab. and its Applications* 16.2, pp. 264–280 (cit. on p. 28).

Vapnik, Vladimir Naumovich (1998). *Statistical Learning Theory*. New York, NY, USA: Wiley (cit. on p. 42).

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. (cit. on p. 1).

Vehtari, Aki et al. (2021). "Rank-Normalization, Folding, and Localization: An Improved $\widehat{R}$ for Assessing Convergence of MCMC". In: *Bayesian Analysis*, pp. 1 –38. DOI: 10.1214/20-BA1221 (cit. on p. 64).

Verma, Vikas et al. (2019). "Manifold Mixup: Better Representations by Interpolating Hidden States". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6438–6447 (cit. on p. 37).

Walker, Stephen G. (2013). "Bayesian inference with misspecified models". In: *Journal of Statistical Planning and Inference* 143.10, pp. 1621–1633. ISSN: 0378-3758. DOI: https://doi.org/10.1016/j.jspi.2013.05.013 (cit. on p. 33).

Wauthier, Fabian L and Michael I. Jordan (2010). "Heavy-Tailed Process Priors for Selective Shrinkage". In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty et al. Curran Associates, Inc., pp. 2406–2414 (cit. on pp. 97, 98, 105).

Welinder, P. et al. (2010). *Caltech-UCSD Birds 200*. Tech. rep. CNS-TR-2010-001. California Institute of Technology (cit. on pp. 46, 70).

Welling, Max and Yee Whye Teh (2011). "Bayesian Learning via Stochastic Gradient Langevin Dynamics." In: *ICML*. Ed. by Lise Getoor and Tobias Scheffer, pp. 681–688 (cit. on p. 64).

Wenger, Jonathan, Hedvig Kjellström, and Rudolph Triebel) (2020). "Non-Parametric Calibration for Classification". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, pp. 178–190 (cit. on p. 62).

Wenzel, Florian et al. (2020). "Hyperparameter Ensembles for Robustness and Uncertainty Quantification". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 6514–6527 (cit. on p. 56).

Wilk, Mark van der et al. (2018). "Learning Invariances using the Marginal Likelihood". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. (cit. on p. 39).

Wilson, Andrew G and Zoubin Ghahramani (2010). "Copula Processes". In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty et al. Curran Associates, Inc., pp. 2460–2468 (cit. on pp. 93, 95, 97, 98, 105).

Wilson, Andrew G and Pavel Izmailov (2020). "Bayesian Deep Learning and a Probabilistic Perspective of Generalization". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 4697–4708 (cit. on pp. 2, 57).

Xie, Saining et al. (2017). "Aggregated Residual Transformations for Deep Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995 (cit. on p. 71).

Yun, Sangdoo et al. (2019). "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features". In: *International Conference on Computer Vision (ICCV)* (cit. on p. 37).

Zadrozny, Bianca and Charles Elkan (2001). "Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 609–616. ISBN: 1558607781 (cit. on p. 55).

— (2002). "Transforming Classifier Scores into Accurate Multiclass Probability Estimates". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, 694–699. ISBN: 158113567X (cit. on p. 55).

Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide Residual Networks". In: *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. Ed. by Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith. BMVA Press (cit. on pp. 46, 71).

Zellner, Arnold (1988). "Optimal Information Processing and Bayes's Theorem". In: *The American Statistician* 42.4, pp. 278–280 (cit. on p. 60).

Zhang, Chiyuan et al. (2017). "Understanding deep learning requires rethinking generalization". In: *International Conference on Learning Representations* (cit. on pp. 22, 36, 42).

Zhang, Hongyi et al. (2018). "mixup: Beyond Empirical Risk Minimization". In: *International Conference on Learning Representations* (cit. on pp. 37, 46).