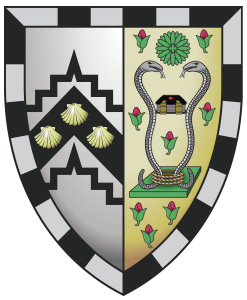
Uncertainty in Deep Learning



Yarin Gal

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Gonville and Caius College

September 2016

Declaration

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

Yarin Gal
September 2016

Acknowledgements

I would like to thank the many people that helped through comments and discussions during the writing of the various papers composing this thesis. I would like to thank (in alphabetical order) Christof Angermueller, Yoshua Bengio, Phil Blunsom, Yutian Chen, Roger Frigola, Zoubin Ghahramani, Shane Gu, Alex Kendall, Yingzhen Li, Rowan McAllister, Carl Rasmussen, Ilya Sutskever, Gabriel Synnaeve, Nilesh Tripuraneni, Richard Turner, Oriol Vinyals, Adrian Weller, Mark van der Wilk, Yan Wu, and many other reviewers for their helpful comments and discussions. I would further like to thank my collaborators Rowan McAllister, Carl Rasmussen, Richard Turner, Mark van der Wilk, and special thanks to my supervisor Zoubin Ghahramani.

Lastly, I would like to thank Google for supporting three years of my PhD with the Google European Doctoral Fellowship in Machine Learning, and Qualcomm for supporting my fourth year with the Qualcomm Innovation Fellowship.

Abstract

Deep learning has attracted tremendous attention from researchers in various fields of information engineering such as AI, computer vision, and language processing [Kalchbrenner and Blunsom, 2013; Krizhevsky et al., 2012; Mnih et al., 2013], but also from more traditional sciences such as physics, biology, and manufacturing [Anjos et al., 2015; Baldi et al., 2014; Bergmann et al., 2014]. Neural networks, image processing tools such as convolutional neural networks, sequence processing models such as recurrent neural networks, and regularisation tools such as dropout, are used extensively. However, fields such as physics, biology, and manufacturing are ones in which representing model uncertainty is of crucial importance [Ghahramani, 2015; Krzywinski and Altman, 2013]. With the recent shift in many of these fields towards the use of Bayesian uncertainty [Herzog and Ostwald, 2013; Nuzzo, 2014; Trafimow and Marks, 2015], new needs arise from deep learning.

In this work we develop tools to obtain practical uncertainty estimates in deep learning, casting recent deep learning tools as Bayesian models without changing either the models or the optimisation. In the first part of this thesis we develop the theory for such tools, providing applications and illustrative examples. We tie approximate inference in Bayesian models to dropout and other stochastic regularisation techniques, and assess the approximations empirically. We give example applications arising from this connection between modern deep learning and Bayesian modelling such as active learning of image data and data-efficient deep reinforcement learning. We further demonstrate the tools' practicality through a survey of recent applications making use of the suggested techniques in language applications, medical diagnostics, bioinformatics, image processing, and autonomous driving. In the second part of the thesis we explore the insights stemming from the link between Bayesian modelling and deep learning, and its theoretical implications. We discuss what determines model uncertainty properties, analyse the approximate inference analytically in the linear case, and theoretically examine various priors such as spike and slab priors.

Table of contents

Nomenclature

Roman Symbols

$\mathbf{A}$	matrix
$\mathbf{a}$	vector
a	scalar
$\mathbf{W}$	Weight matrix
$\mathcal{D}$	Dataset
$\mathbf{X}$	Dataset inputs (matrix with N rows, one for each data point)
$\mathbf{Y}$	Dataset outputs (matrix with N rows, one for each data point)
$\mathbf{x}_n$	Input data point for model
$\mathbf{y}_n$	Output data point for model
$\widehat{\mathbf{y}}_n$	Model output on input $\mathbf{x}_n$
Q	Input dimensionality
D	Output dimensionality
L	Number of network layers
K_i	Number of network units in layer i

Greek Symbols

ϵ	a random variable
$\widehat{\epsilon}$	a random variable realisation

$\boldsymbol{\epsilon}$ a vector of random variables

$\boldsymbol{\omega}$ a set of random variables (for example random weight matrices)

Superscripts

$\boldsymbol{\omega}$ parameters of a function (e.g. $f^{\boldsymbol{\omega}}$)

Subscripts

1 variable (e.g. $\mathbf{W}_1 : Q \times K, \mathbf{W}_2 : K \times D$)

$1, ij$ specific element of variable $\mathbf{W}_1$ (e.g. $w_{1,ij}$ denotes the element at row i column j of the variable $\mathbf{W}_1$)

i row / column of a matrix (with bold letter, e.g. $\mathbf{x}_i$)

ij specific element of a matrix (e.g. x_{ij})

q with reference to a variable $\mathbf{W} : Q \times K$, $\mathbf{w}_q$ denotes a row, and $\mathbf{w}_k$ denotes a column of $\mathbf{W}$

Other Symbols

$\mathcal{N}$ The Gaussian distribution

$\mathcal{MN}$ The matrix Gaussian distribution

$\mathbb{R}$ The real numbers

Acronyms / Abbreviations

BNN Bayesian neural network

ELBO Evidence lower bound

KL Kullback–Leibler

CDF Cumulative distribution function

CNN Convolutional neural network

GP Gaussian process

MC Monte Carlo

MCMC Markov chain Monte Carlo

MDL Minimum description length

NN Neural network

pdf Probability density function

RL Reinforcement learning

RMSE Root mean square error

RNN Recurrent neural network

SRT Stochastic regularisation technique (such as dropout, multiplicative Gaussian noise, etc.)

VI Variational inference

a.e. Almost everywhere

e.g. Exempli gratia ("for the sake of an example")

i.e. Id est ("it is")

i.i.d. Independent and identically distributed

s.t. Such that

w.r.t. With respect to

Notation

We use the following notation throughout the work. Bold lower case letters ($\mathbf{x}$) denote vectors, bold upper case letters ($\mathbf{X}$) denote matrices, and standard weight letters (x) denote scalar quantities. We use subscripts to denote either entire rows / columns (with bold letters, $\mathbf{x}_i$), or specific elements (x_{ij}). We use subscripts to denote variables as well (such as $\mathbf{W}_1 : Q \times K, \mathbf{W}_2 : K \times D$), with corresponding lower case indices to refer to specific rows / columns ($\mathbf{w}_q, \mathbf{w}_k$ for the first variable and $\mathbf{w}_k, \mathbf{w}_d$ for the second). We use a second subscript to denote the element index of a specific variable: $w_{1,qk}$ denotes the element at row q column k of the variable $\mathbf{W}_1$. Lastly, we use $\boldsymbol{\omega}$ to denote a set of variables (e.g. $\boldsymbol{\omega} = \{\mathbf{W}_1, ..., \mathbf{W}_L\}$) and superscript $f^{\boldsymbol{\omega}}$ to denote a function parametrised by the variables $\boldsymbol{\omega}$.

Chapter 1

Introduction: The Importance of Knowing What We Don't Know

In the *Bayesian machine learning* community we work with probabilistic models and uncertainty. Models such as Gaussian processes, which define probability distributions over functions, are used to learn the more likely and less likely ways to generalise from observed data. This probabilistic view of machine learning offers confidence bounds for data analysis and decision making, information that a biologist for example would rely on to analyse her data, or an autonomous car would use to decide whether to brake or not. In analysing data or making decisions, it is often necessary to be able to tell whether a model is certain about its output, being able to ask "maybe I need to use more diverse data? or change the model? or perhaps be careful when making a decision?". Such questions are of fundamental concern in Bayesian machine learning, and have been studied extensively in the field [Ghahramani, 2015]. When using *deep learning models* on the other hand [Goodfellow et al., 2016], we generally only have point estimates of parameters and predictions at hand. The use of such models forces us to sacrifice our tools for answering the questions above, potentially leading to situations where we can't tell whether a model is making sensible predictions or just guessing at random.

Most deep learning models are often viewed as *deterministic functions*, and as a result viewed as operating in a very different setting to the probabilistic models which possess uncertainty information. Perhaps for this reason it is quite surprising to see how close modern deep learning is to probabilistic modelling. In fact, we shall see that we can get uncertainty information from existing deep learning models for free—without changing

a thing. The main goal of this thesis is to develop such practical tools to reason about uncertainty in deep learning.

1.1 Deep learning

To introduce deep learning, I shall start from the simplest of the statistical tools: *linear regression* [Gauss, 1809; Legendre, 1805; Seal, 1967]. In linear regression we are given a set of N input-output pairs $\{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, for example CO_2-temperature observations, or the average number of accidents for different driving speeds. We assume that there exists a *linear* function mapping each $\mathbf{x}_i \in \mathbb{R}^Q$ to $\mathbf{y}_i \in \mathbb{R}^D$ (with $\mathbf{y}_i$ potentially corrupted with observation noise). Our *model* in this case is a linear transformation of the inputs: $\mathbf{f}(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$, with $\mathbf{W}$ some Q by D matrix over the reals and $\mathbf{b}$ a real vector with D elements. Different parameters $\mathbf{W}, \mathbf{b}$ define different linear transformations, and our aim is to find parameters that, for example, would minimise the average squared error over our observed data: $\frac{1}{N}\sum_i ||\mathbf{y}_i - (\mathbf{x}_i\mathbf{W} + \mathbf{b})||^2$.

In more general cases where the relation between $\mathbf{x}$ and $\mathbf{y}$ need not be linear, we may wish to define a *non-linear* function $\mathbf{f}(\mathbf{x})$ mapping the inputs to the outputs. For this we can resort to *linear basis function regression* [Bishop, 2006; Gergonne, 1815; Smith, 1918], where the input $\mathbf{x}$ is fed through K fixed scalar-valued non-linear transformations $\phi_k(\mathbf{x})$ to compose a *feature* vector $\Phi(\mathbf{x}) = [\phi_1(\mathbf{x}), ..., \phi_K(\mathbf{x})]$. We then perform linear regression with this vector instead of $\mathbf{x}$ itself. The transformations ϕ_k are our basis functions, and with scalar input x, these can be wavelets parametrised by k, polynomials of different degrees x^k, or sinusoidals with various frequencies. When $\phi_k(\mathbf{x}) := x_k$ and $K = Q$, basis function regression reduces to linear regression. The basis functions are often assumed to be fixed and orthogonal to each other, and an optimal combination of these functions is sought.

Relaxing the constraint for the basis functions to be fixed and mutually orthogonal, we can use *parametrised* basis functions instead [Bishop, 2006]. For example, we may define the basis functions to be $\phi_k^{\mathbf{w}_k, b_k}$ where the scalar-valued function ϕ_k is applied to the inner-product $\langle \mathbf{w}_k, \mathbf{x} \rangle + b_k$. In this case ϕ_k are often defined to be identical for all k, for example $\phi_k(\cdot) = \sin(\cdot)$ giving $\phi_k^{\mathbf{w}_k, b_k}(\mathbf{x}) = \sin(\langle \mathbf{w}_k, \mathbf{x} \rangle + b_k)$. The feature vector composed of the basis functions' output is again fed as the input to a linear transformation. The model output can then be written more compactly as $\mathbf{f}(\mathbf{x}) = \Phi^{\mathbf{W}_1, \mathbf{b}_1}(\mathbf{x})\mathbf{W}_2 + \mathbf{b}_2$ with $\Phi^{\mathbf{W}_1, \mathbf{b}_1}(\mathbf{x}) = \phi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$, $\mathbf{W}_1$ a matrix of dimensions Q by K, $\mathbf{b}_1$ a vector with K elements, $\mathbf{W}_2$ a matrix of dimensions K by D, and $\mathbf{b}_2$ a vector with D elements. To

perform regression now we can find $\mathbf{W}_1, \mathbf{b}_1$ as well as $\mathbf{W}_2, \mathbf{b}_2$ that minimise the average squared error over our observed data, $||\mathbf{y} - \mathbf{f}(\mathbf{x})||^2$.

The most basic model in *deep learning* can be described as a hierarchy of these parametrised basis functions (such a hierarchy is referred to as a *neural network* for historical reasons, with each feature vector in the hierarchy referred to as a *layer*). In the simplest setting of regression we would simply compose multiple basis function regression models, and for classification we would further compose a logistic function at the end (which "squashes" the linear model's output to obtain a probability vector). Each layer in the hierarchy can be seen as a "building block", and the modularity in the composition of such blocks embodies the versatility of deep learning models. The simplicity of each block, together with the many possibilities of model combinations, might be what led many engineers to work in the field. This in turn has led to the development of tools that *work well* and *scale well*.

We will continue with a review of simple neural network models, relating the notation in the field of deep learning to the mathematical formalism of the above linear basis function regression models. We then extend these to specialised models designed to process image data and sequence data. In the process we will introduce some of the terminology and mathematical notation used throughout the work. We will describe the models formally but succinctly, which will allow us to continue our discussion in the introduction using a more precise language.

Feed-forward neural networks (NNs). We will first review a neural network model [Rumelhart et al., 1985] for a *single hidden layer*. This is done for ease of notation, and the generalisation to multiple layers is straightforward. We denote by $\mathbf{x}$ the model input (referred to as *input layer*, a row vector with Q elements), and transform it with an affine transformation to a row vector with K elements. We denote by $\mathbf{W}_1$ the linear map (referred to as a *weight matrix*) and by $\mathbf{b}$ the translation (referred to as a *bias*) used to transform the input $\mathbf{x}$ to obtain $\mathbf{x}\mathbf{W}_1 + \mathbf{b}$. An element-wise non-linearity $\sigma(\cdot)$ (such as the rectified linear[1] (ReLU) or TanH) is then applied to the transformation output, resulting in a *hidden layer* with each element referred to as a *network unit*. This is followed by a second linear transformation with weight matrix $\mathbf{W}_2$ mapping the hidden layer to the model output (referred to as *output layer*, a row vector with D elements). These two layers are also referred to as *inner-product layers*. We thus have $\mathbf{W}_1$ is a $Q \times K$ matrix, $\mathbf{W}_2$ is a $K \times D$ matrix, and $\mathbf{b}$ is a K dimensional vector. A standard

[1] $\text{relu}(x) = \max(x, 0)$.

network would output

$$\widehat{\mathbf{y}} = \sigma(\mathbf{x}\mathbf{W}_1 + \mathbf{b})\mathbf{W}_2$$

given some input[2] $\mathbf{x}$.

To use the network for regression we might use the Euclidean loss,

$$E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{X},\mathbf{Y}) = \frac{1}{2N}\sum_{i=1}^{N}||\mathbf{y}_i - \widehat{\mathbf{y}}_i||^2 \tag{1.1}$$

where $\{\mathbf{y}_1,...,\mathbf{y}_N\}$ are N observed outputs, and $\{\widehat{\mathbf{y}}_1,...,\widehat{\mathbf{y}}_N\}$ are the outputs of the model with corresponding observed inputs $\{\mathbf{x}_1,...,\mathbf{x}_N\}$. Minimising this loss w.r.t. $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}$ would hopefully result in a model that can generalise well to unseen test data $\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}}$.

To use the model for classification, predicting the probability of $\mathbf{x}$ being classified with a label in the set $\{1,...,D\}$, we pass the output of the model $\widehat{\mathbf{y}}$ through an element-wise softmax function to obtain normalised scores: $\widehat{p}_d = \exp(\widehat{y}_d)/\left(\sum_{d'}\exp(\widehat{y}_{d'})\right)$. Taking the log of $\widehat{p}_d$ (with d being the observed label) results in a *softmax* loss,

$$E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{X},\mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\log(\widehat{p}_{i,d_i}) \tag{1.2}$$

where $d_i \in \{1,2,...,D\}$ is the observed class for input i.

A big difficulty with the models above is their tendency to overfit—decrease their loss on the training set $\mathbf{X},\mathbf{Y}$ while increasing their loss on the test set $\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}}$. For this reason a regularisation term is often added during optimisation. We often use L_2 regularisation for each parameter weighted by some weight decays λ_i, resulting in a minimisation objective (often referred to as a *cost*),

$$\mathcal{L}(\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}) := E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{X},\mathbf{Y}) + \lambda_1||\mathbf{W}_1||^2 + \lambda_2||\mathbf{W}_2||^2 + \lambda_3||\mathbf{b}||^2. \tag{1.3}$$

The above single hidden layer NN with the Euclidean loss is identical to a basis function regression model. Extending this simple NN model to multiple layers results in a more expressive model.

Remark (Model expressiveness)**.** An intuitive definition for model expressiveness might be the complexity of functions a model can capture (defining what complex functions are is not trivial by itself, although when dealing with polynomials one might define polynomials of high degree to be more complex than polynomials of

[2]Note that the output bias was omitted here; this is equivalent to centring the observed outputs.

low degree). In this sense hierarchical basis function models are more expressive than their "flat" counter-parts. It is interesting to note that even though "flat" basis function regression can model any function up to any given precision with a large enough number of basis functions [Cybenko, 1989; Hornik, 1991], with a hierarchy we can use much smaller models. Consider the example of basis function regression with polynomial basis functions $\phi_k \in \{1, x, x^2, ..., x^{K-1}\}$: the set of functions expressible with these K basis functions is {all polynomials up to degree $K-1$}. Composing the basis function regression model L times results in a model that can capture (a subset of) functions from the set {all polynomials up to degree $(K-1)^L$}. A "flat" model capturing polynomials up to degree $(K-1)^L$ would require K^L basis functions, whereas a hierarchical model with similar (but not identical) expressiveness would only require $K \times L$ basis functions. Model expressiveness is further discussed in [Bengio and LeCun, 2007] for example, where binary circuits are used as an illustrative example.

The simple model structure presented above can be extended to specialised models, aimed at treating image inputs or sequence inputs. We will next review these models quickly.

Convolutional neural networks (CNNs). CNNs [LeCun et al., 1989; Rumelhart et al., 1985] are popular deep learning tools for image processing, which can solve tasks that until recently were considered to lie beyond our reach [Krizhevsky et al., 2012; Szegedy et al., 2014]. The model is made of a recursive application of convolution and pooling layers, followed by inner product layers at the end of the network (simple NNs as described above). A convolution layer is a linear transformation that preserves spatial information in the input image (depicted in figure 1.1). Pooling layers simply take the output of a convolution layer and reduce its dimensionality (by taking the maximum of each $(2, 2)$ block of pixels for example). The convolution layer will be explained in more detail in section §3.4.1.

Similarly to CNNs, recurrent neural networks are specialised models designed to handle sequence data.

Recurrent neural networks (RNNs). RNNs [Rumelhart et al., 1985; Werbos, 1988] are sequence-based models of key importance for natural language understanding, language generation, video processing, and many other tasks [Kalchbrenner and Blunsom, 2013; Mikolov et al., 2010; Sundermeyer et al., 2012; Sutskever et al., 2014]. The model's

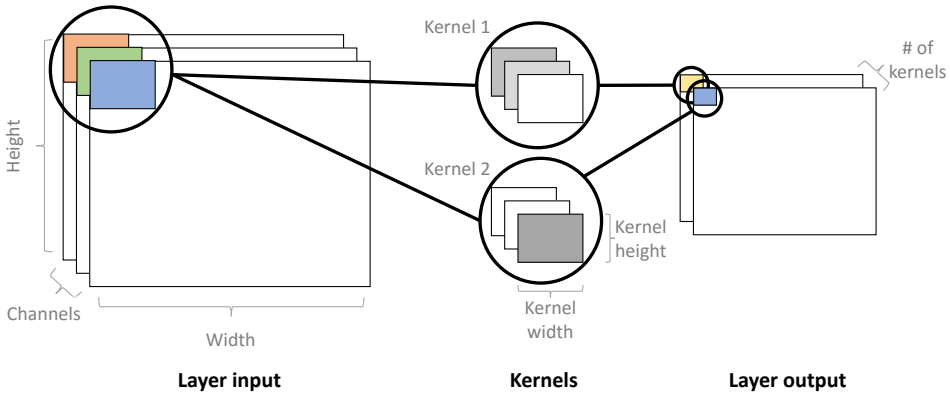


Fig. 1.1 A convolution layer in a CNN. The input image (Layer input) has a given height, width, and channels (RGB for example). Each kernel is convolved with each image patch (a single image patch is depicted under the left-most circle highlight). For example, kernel 2 preserves the blue channel only, resulting in a blue pixel in Layer output. Kernel 1, on the other hand, ignores the blue channel resulting in a yellow pixel in the output layer. This is a simplified view of convolutions: kernels are often not composed of the same value in each spatial location, but rather act as edge detectors or feature detectors.

input is a sequence of symbols, where at each time step a simple neural network (*RNN unit*) is applied to a single symbol, as well as to the network's output from the previous time step. RNNs are powerful models, showing superb performance on many tasks.

We will concentrate on simple RNN models for brevity of notation. Given input sequence $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ of length[3] T, a simple RNN is formed by a repeated application of a function $\mathbf{f_h}$. This generates a hidden state $\mathbf{h}_t$ for time step t:

$$\mathbf{h}_t = \mathbf{f_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) = \sigma(\mathbf{x}_t\mathbf{W_h} + \mathbf{h}_{t-1}\mathbf{U_h} + \mathbf{b_h}).$$

for some non-linearity σ. The model output can be defined, for example, as

$$\widehat{\mathbf{y}} = \mathbf{f_y}(\mathbf{h}_T) = \mathbf{h}_T\mathbf{W_y} + \mathbf{b_y}.$$

The definition of LSTM and GRU—more complicated RNN models—is given later in section §3.4.2.

[3]Note the overloading of notation used here: $\mathbf{x}_i$ is a vector in $\mathbb{R}^Q$, and $\mathbf{x}$ is a sequence of vectors of length T.

1.2 Model uncertainty

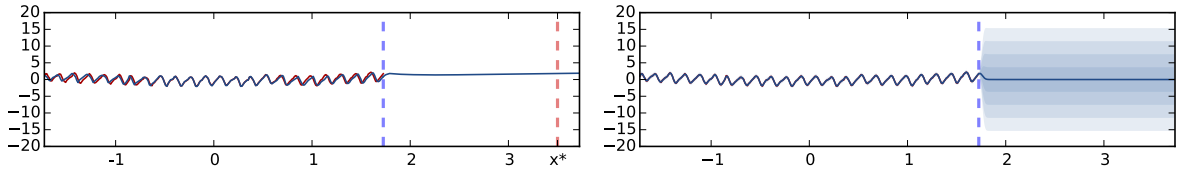
The models above can be used for applications as diverse as skin cancer diagnosis from lesion images, steering in autonomous vehicles, and dog breed classification in a website where users upload pictures of their pets. For example, given several pictures of dog breeds as training data—when a user uploads a photo of his dog—the hypothetical website should return a prediction with rather high confidence. But what should happen if a user uploads a photo of a cat and asks the website to decide on a dog breed?

The above is an example of *out of distribution* test data. The model has been trained on photos of dogs of different breeds, and has (hopefully) learnt to distinguish between them well. But the model has never seen a cat before, and a photo of a cat would lie outside of the data distribution the model was trained on. This illustrative example can be extended to more serious settings, such as MRI scans with structures a diagnostics system has never observed before, or scenes an autonomous car steering system has never been trained on. A possible desired behaviour of a model in such cases would be to return a prediction (attempting to extrapolate far away from our observed data), but return an answer with the added information that the point lies outside of the data distribution (see a simple depiction for the case of regression in figure 1.2). I.e. we want our model to possess some quantity conveying a high level of *uncertainty* with such inputs (alternatively, conveying low *confidence*).

Other situations that can lead to uncertainty include

- noisy data (our observed labels might be noisy, for example as a result of measurement imprecision, leading to *aleatoric uncertainty*),
- *uncertainty in model parameters* that best explain the observed data (a large number of possible models might be able to explain a given dataset, in which case we might be uncertain which model parameters to choose to predict with),
- and *structure uncertainty* (what model structure should we use? how do we specify our model to extrapolate / interpolate well?).

The latter two uncertainties can be grouped under *model uncertainty* (also referred to as *epistemic uncertainty*). Aleatoric uncertainty and epistemic uncertainty can then be used to induce *predictive uncertainty*, the confidence we have in a prediction.



(a) Standard deep learning model (b) Probabilistic model

Fig. 1.2 **Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset, for various models, with out of distribution test point x^*.** In red is the observed function (left of the dashed blue line); in blue is the predictive mean plus/minus two standard deviations. Different shades of blue represent half a standard deviation. Marked with a dashed red line is a point far away from the data: standard deep learning models confidently predict an unreasonable value for the point; the probabilistic model predicts an unreasonable value as well but with the additional information that the model is uncertain about its prediction.

Remark (A note on terminology)**.** The word *epistemic* comes from "episteme", Greek for "knowledge", i.e. epistemic uncertainty is "knowledge uncertainty". *Aleatoric* comes from the Latin "aleator", or "dice player", i.e. aleatoric uncertainty is the "dice player's" uncertainty. Epistemic and aleatoric uncertainties are sometimes referred to as *reducible* and *irreducible* uncertainties respectively, since epistemic uncertainty can be reduced with more data (knowledge), while aleatoric uncertainty cannot (the stochasticity of a dice roll cannot be reduced by observing more rolls). We will avoid this terminology though, since aleatoric uncertainty can also be seen as "reducible" through an increase in measurement precision, i.e. by changing the underlying system with which we perform the experiment.

Uncertainty information is often used in the life sciences, as discussed in the Nature papers by Herzog and Ostwald [2013]; Krzywinski and Altman [2013]; Nuzzo [2014], as well as the entertaining case in [Trafimow and Marks, 2015]. In such fields it is quite important to quantify our confidence about the models' predictions. Uncertainty information is also important for the practitioner. Understanding if a model is underconfident or falsely over-confident (i.e. its uncertainty estimates are too small) can help get better performance out of it. Recognising that test data is far from the training data we could augment the training data for example.

But perhaps much more important, model uncertainty information can be used in systems that make decisions that affect human life—either directly or indirectly—as discussed next.

1.3 Model uncertainty and AI safety

With recent engineering advances in the field of machine learning, systems that until recently were only applied to toy data are now being deployed in real-life settings. Among these settings are scenarios in which control is handed-over to automated systems in situations which have the possibility to become life-threatening to humans. These include automated decision making or recommendation systems in the medical domain, autonomous control of drones and self driving cars, the ability to affect our economy on a global scale through high frequency trading, as well as control of critical systems. These can all be considered under the umbrella field of *AI safety*.

This interpretation of AI safety is rather different to other interpretations given in the field, which mostly concentrate on *reinforcement learning* (RL) settings. For example, some look at the design of environments for self-learning agents that do not allow the agent to exploit deficiencies in the learning process itself (such as "reward hacking"; see for example [Amodei et al., 2016]). In contrast, I will discuss scenarios in which certain decisions made by a machine learning model trained in a *supervised* setting might endanger human life (i.e. settings in which a model incorrectly mapping inputs to outputs can lead to undesirable consequences). In some of these scenarios, relying on model uncertainty to adapt the decision making process might be key to preventing unintended behaviour.

1.3.1 Physician diagnosing a patient

When a physician advises a patient to use certain drugs based on a medical record analysis, the physician would often rely on the confidence of the expert analysing the medical record. The introduction of systems such as automated cancer detection based on MRI scans though could make this process much more complicated. Even at the hands of an expert, such systems could introduce biases affecting the judgement of the expert. A system encountering test examples which lie outside of its data distribution could easily make unreasonable suggestions, and as a result unjustifiably bias the expert. However, given model confidence an expert could be informed at times when the system is essentially guessing at random.

1.3.2 Autonomous vehicles

Autonomous systems can range from a simple robotic vacuum scuttering around the floor, to self-driving cars transporting people and goods from one place to another. These

systems can largely be divided into two groups: those relying on rule-based systems to control their behaviour, and those which can learn to adapt their behaviour to their environment. Both can make use of machine learning tools. The former group through low-level use of machine learning algorithms for feature extraction, and the latter one through reinforcement learning.

With self-driving cars, low level feature extraction such as image segmentation and image localisation are used to process raw sensory input [Bojarski et al., 2016]. The output of such models is then fed into higher-level decision making procedures. The higher-level decision making can be done through expert systems for example relying on a fixed set of rules ("if there is a cyclist to your left, do not steer left"). However, mistakes done by lower-level machine learning components can propagate up the decision making process and lead to devastating results. One concrete example demonstrating the risk of such approaches, in an assisted driving system, is the failure of a low-level component to distinguish the white side of a turning trailer from a bright sky, which led to the first fatality of an assisted driving system [NHTSA, 2017]. In such modular systems, one could use the model's confidence in low-level components and make high-level decisions given this uncertainty information. For example, a segmentation system which can identify its uncertainty in distinguishing between the sky and another vehicle could alert the user to take control over the steering.

1.3.3 Critical systems and high frequency trading

As a final example, it is interesting to notice that control over critical systems is slowly being handed over to machine learning systems. This can happen in a post office, sorting letters according to their zip code [LeCun and Cortes, 1998; LeCun et al., 1998], or in a nuclear power plant with a system responsible for critical infrastructure [Linda et al., 2009]. Even in high frequency trading—where computers are given control over systems that could potentially destabilise entire economic markets—issuing an unusual trading command could cause a calamity. A possible solution, when using rule-based decision making, is to rely on formal program verification systems. Such systems allow the developer to verify the program is working as intended before deployment. But machine learning-based decision making systems do not allow this. What should a system do in the case of outputs with high uncertainty?

With model confidence at hand one possible solution would be to treat uncertain outputs as special cases explicitly. In the case of a critical system we might decide to pass the input to a human to make a decision. Alternatively, one could use a simple and

fast model to perform predictions, and use a more elaborate but slower model only for inputs on which the weak model is uncertain.

In this work we will develop the tools required to reason about model confidence in deep learning, which could be applied in the scenarios discussed above. This will be by developing a general framework that can be applied to existing tools. This in turn allows the continued use of existing systems which have proven themselves useful, and for which large amounts of research have already been dedicated. Concrete examples of the use of these developments in the settings mentioned above will be given in section §5.1.

1.4 Applications of model uncertainty

Beside AI safety, there exist many applications which rely on model uncertainty. These applications include choosing what data to learn from, or exploring an agent's environment efficiently. Common to both these tasks is the use of model uncertainty to learn from *small amounts of data*. This is often a necessity in settings in which data collection is expensive (such as the annotation of individual examples by an expert), or time consuming (such as the repetition of an experiment multiple times).

1.4.1 Active learning

How could we use machine learning to aid an expert working in a laborious field? One approach is to automate small parts of the expert's work, such as mundane cell counting, or cancer diagnosis based on MRI scans. This can be a difficult problem in machine learning though. Many machine learning algorithms, including deep learning, often require large amounts of labelled data to generalise well. The amount of labelled data required increases with the complexity of the problem or the complexity of the input data: image inputs for example often require large models to be processed, and in turn these require large amounts of data (Krizhevsky et al. [2012] for example use hundreds of gigabytes of labelled images). To automate MRI scan analysis for example, this would require an expert to annotate a large number of MRI scans, labelling them to indicate a patient having cancer or not. But expert time is expensive, and often obtaining the amount of required labelled data is not feasible. How can we learn in settings where labelled data is scarce and expert knowledge is expensive?

One possible approach to this task could rely on active learning [Settles, 2010]. In this learning framework the model itself would choose what unlabelled data would be most

informative for it, and ask an external "oracle" (for example a human annotator) for a label only for these new data points. The choice of data points to be labelled is done through an acquisition function, which ranks points based on their potential informativeness. Different acquisition functions exist, and many make use of model uncertainty about the unlabelled data points in order to decide on their potential informativeness [Houlsby et al., 2011]. Following this learning framework we can decrease the amount of required data by orders of magnitude, while still maintaining good model performance (as we will see below in §5.2).

Returning to the example above of cancer diagnosis from MRI scans, we would seek a model that can produce *good uncertainty estimates* for image data, and rely on these to design an appropriate acquisition function. Deep learning provides superb tools for image processing that generalise well, but these rely on huge amounts of labelled data and do not provide model uncertainty. In this work we will develop extensions of such tools that can be deployed in small data regimes, and provide good model confidence. With these tools we will demonstrate the feasibility of the ideas above in active learning (section §5.2, joint work with Riashat Islam as part of his Master's project).

1.4.2 Efficient exploration in deep reinforcement learning

Reinforcement learning (RL) algorithms learn control tasks via trial and error, much like a child learning to ride a bicycle [Sutton and Barto, 1998]. But trials of real world control tasks often involve time and resources we wish not to waste. Alternatively, the number of trials might be limited due to wear and tear of the system, making data-efficiency critical.

As a simple introduction to reinforcement learning, consider an agent (a Roomba vacuum for example) that needs to learn about its environment (a living room) based on its actions (rolling around in different directions). It can decide to go forward and might bump into a wall. Encouraging the Roomba not to crash into walls with positive rewards, over time it will learn to avoid them in order to maximise its rewards. The Roomba has to explore its environment looking for these rewards, and trade-off between this exploration, and exploitation of what it already knows.

Recent advances in deep learning approaches to RL (referred to as *deep RL*) have demonstrated impressive results in game playing [Mnih et al., 2013]. Such approaches make use of NNs for Q-value function approximation. These are functions that estimate the quality of different actions an agent can take. Epsilon greedy search is often used where the agent selects its best action with some probability and explores otherwise. But with uncertainty information an agent can decide when to exploit and when to explore

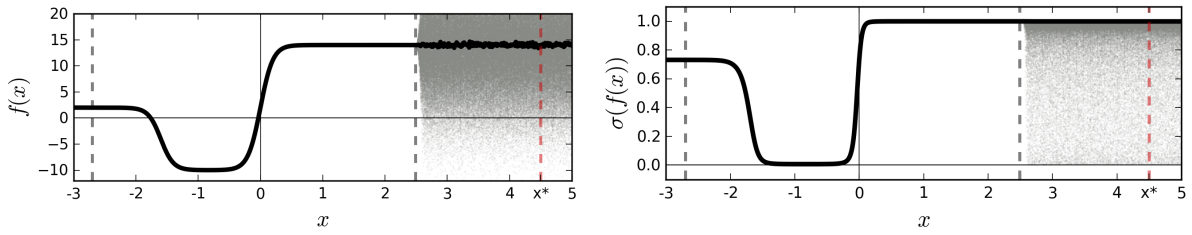
its environment. And with uncertainty estimates over the agent's Q-value function, techniques such as Thompson sampling [Thompson, 1933] can be used to learn much faster. This will be demonstrated below (section §5.3).

Even though exploration techniques such as Thompson sampling can help learn better policies faster, a much more drastic improvement in data efficiency can be achieved by modelling the system dynamics [Atkeson and Santamaria, 1997]. A dynamics model allows the agent to generalise its knowledge about the system dynamics to other, unobserved, states. *Probabilistic* dynamics models allow an agent to consider transition uncertainty throughout planning and prediction, improving data efficiency even further. PILCO [Deisenroth and Rasmussen, 2011], for example, is a data-efficient probabilistic model-based policy search algorithm. PILCO analytically propagates uncertain state distributions through a Gaussian process dynamics model. This is done by recursively feeding the output state distribution (*output uncertainty*) of one time step as the input state distribution (*input uncertainty*) of the next time step, until a fixed time horizon T. This allows the agent to consider the long-term consequences (expected cumulative cost) of a particular controller parametrisation w.r.t. all plausible dynamics models. PILCO relies on Gaussian processes (GPs), which work extremely well with small amounts of low dimensional data, but scale cubically with the number of trials. Further, PILCO's distribution propagation adds a squared term in the observation space dimensionality, making it hard to scale the framework to high dimensional observation spaces. This makes it difficult to use PILCO with tasks that require a larger number of trials. Even more so, PILCO does not consider temporal correlation in model uncertainty between successive state transitions. This means that PILCO underestimates state uncertainty at future time steps [Deisenroth et al., 2015], which can lead to diminished performance.

In section §5.4 we attempt to answer these shortcomings by replacing PILCO's Gaussian process with a Bayesian deep dynamics model, while maintaining the framework's probabilistic nature and its data-efficiency benefits (joint work with Rowan McAllister and Carl Rasmussen [Gal et al., 2016]).

1.5 Model uncertainty in deep learning

Having established that model confidence is a good quantity to possess, it is important to note that most deep learning models do not offer such information. Regression models output a single vector regressing to the mean of the data (as can be seen in figure 1.2). In classification models, the probability vector obtained at the end of the pipeline (the softmax output) is often erroneously interpreted as model confidence. A model can be



(a) Arbitrary function $f(\mathbf{x})$ as a function of data $\mathbf{x}$ (softmax *input*)

(b) $\sigma(f(\mathbf{x}))$ as a function of data $\mathbf{x}$ (softmax *output*)

Fig. 1.3 **A sketch of softmax input and output for an idealised binary classification problem.** Training data is given between the dashed grey lines. Function point estimate is shown with a solid line. Function uncertainty is shown with a shaded area. Marked with a dashed red line is a point x^* far from the training data. Ignoring function uncertainty, point x^* is classified as class 1 with probability 1.

uncertain in its predictions even with a high softmax output (figure 1.3). Passing a point estimate of a function (solid line 1.3a) through a softmax (solid line 1.3b) results in extrapolations with unjustified high confidence for points far from the training data. x^* for example would be classified as class 1 with probability 1. However, passing the distribution (shaded area 1.3a) through a softmax (shaded area 1.3b) better reflects classification uncertainty far from the training data.

Even though modern deep learning models used in practice do not capture model confidence, they are closely related to a family of probabilistic models which induce probability distributions over functions: the Gaussian process. Given a neural network, by placing a probability distribution over each weight (a standard normal distribution for example), a Gaussian process can be recovered in the limit of infinitely many weights (see Neal [1995] or Williams [1997]). For a finite number of weights, model uncertainty can still be obtained by placing distributions over the weights—these models are called *Bayesian neural networks*. These were extensively studied by [MacKay, 1992b], work which was further extended by [Neal, 1995]. More recently these ideas have been resurrected under different names with variational techniques by [Blundell et al., 2015; Graves, 2011; Kingma and Welling, 2013] (although such techniques used with Bayesian neural networks can be traced back as far as Hinton and Van Camp [1993] and Barber and Bishop [1998]). But some of these models are quite difficult to work with—often requiring many more parameters to be optimised—and haven't really caught-on within the deep learning community, perhaps because of their limited practicality.

What would make a tool for obtaining model uncertainty *practical* then? One requirement of such a tool would be to scale well to large data, and scale well to complex models (such as CNNs and RNNs). Much more important perhaps, it would be

impractical to change existing model architectures that have been well studied, and it is often impractical to work with complex and cumbersome techniques which are difficult to explain to non-experts. Existing approaches to obtain model confidence often do not scale to complex models or large amounts of data, and require us to develop new models for existing tasks for which we already have well performing tools.

We will thus concentrate on the development of practical techniques to obtain model confidence in deep learning, techniques which are also well rooted within the theoretical foundations of probability theory and Bayesian modelling. Specifically, we will make use of stochastic regularisation techniques (SRTs). SRTs are recently developed techniques for model regularisation that have been tremendously successful within deep learning, and are used in almost *all* modern deep learning models. These techniques adapt the model output stochastically as a way of model regularisation (hence the name *stochastic regularisation*). This results in the loss becoming a random quantity, which is optimised using tools from the stochastic non-convex optimisation literature. Popular SRTs include dropout [Hinton et al., 2012], multiplicative Gaussian noise [Srivastava et al., 2014], dropConnect [Wan et al., 2013], and countless other recent techniques[4,5].

As we will see below, we can take almost any network trained with an SRT, and given some input $\mathbf{x}^*$ obtain a predictive mean $\mathbb{E}[\mathbf{y}^*]$ (the expected model output given our input), and predictive variance $\text{Var}[\mathbf{y}^*]$ (how much the model is confident in its prediction). To obtain these, we simulate a network output with input $\mathbf{x}^*$, treating the SRT as if we were using the model during training (i.e. obtain a random output through a *stochastic forward pass*). We repeat this process several times (for T repetitions), sampling i.i.d. outputs $\{\widehat{\mathbf{y}}_1^*(\mathbf{x}^*), ..., \widehat{\mathbf{y}}_T^*(\mathbf{x}^*)\}$. As will be explained below, these are empirical samples from an *approximate predictive distribution*. We can get an empirical estimator for the predictive mean of our approximate predictive distribution as well as the predictive variance (our uncertainty) from these samples:

$$\begin{aligned}\mathbb{E}[\mathbf{y}^*] &\approx \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}_t^*(\mathbf{x}^*)\\ \text{Var}[\mathbf{y}^*] &\approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}_t^*(\mathbf{x}^*)^T\widehat{\mathbf{y}}_t^*(\mathbf{x}^*) - \mathbb{E}[\mathbf{y}^*]^T\mathbb{E}[\mathbf{y}^*].\end{aligned} \tag{1.4}$$

Theoretical justification for these two simple equations will be given in chapter 3.

[4] The idea of adding noise to a model to avoid overfitting is quite old, and for input noise has been studied for example in [Bishop, 1995].

[5] In the dropout case, initial research studying the technique from a Bayesian perspective includes [Ba and Frey, 2013; Maeda, 2014; Wang and Manning, 2013].

Equation (1.4) results in uncertainty estimates which are practical with large models and big data, and that can be applied in image based models, sequence based models, and many different settings such as reinforcement learning and active learning. Further, the combination of these techniques allows us to perform tasks that were not possible until recently. For example, we demonstrate below how to perform active learning with *image data*, a task which is extremely challenging due to the lack of tools offering good uncertainty estimates from image data.

1.6 Thesis structure

The first part of this thesis (chapters 3–5) will be concerned with providing tools to obtain practical uncertainty estimates, and demonstrating how these tools could be used in many example applications. This part should be easily accessible for experts as well as non-experts in the field. The second part of this thesis (chapter 6) goes in depth into the theoretical implications of the work above.

Some of the work in this thesis was previously presented in [Gal, 2015; Gal and Ghahramani, 2015a,b,c,d, 2016a,b,c; Gal et al., 2016], but this thesis contains many new pieces of work as well. The most notable of these are a theoretical analysis of Monte Carlo estimator variance used in variational inference (§3.1.1–§3.1.2), a survey of measures of uncertainty in classification tasks (§3.3.1), an empirical analysis of different Bayesian neural network priors (§4.1) and posteriors with various approximating distributions (§4.2), new quantitative results comparing dropout to existing techniques (§4.3), tools for heteroscedastic predictive uncertainty in Bayesian neural networks (§4.6), applications in active learning (§5.2), a discussion of what determines what our model uncertainty looks like (§6.1–§6.2), an analytical analysis of the dropout approximating distribution in Bayesian linear regression (§6.3), an analysis of ELBO-test log likelihood correlation (§6.4), discrete prior models (§6.5), an interpretation of dropout as a proxy posterior in spike and slab prior models (§6.6), as well as a procedure to optimise the dropout probabilities based on the variational interpretation to separate the different sources of uncertainty (§6.7).

The code for the experiments presented in this work is available at `https://github.com/yaringal`.

Chapter 2

The Language of Uncertainty

To formalise our discussion of model uncertainty we will rely on probabilistic modelling, and more specifically on Bayesian modelling. Bayesian probability theory offers us the machinery we need to develop our tools. Together with techniques for approximate inference in Bayesian models, in the next chapter we will present the main results of this work. But prior to that, let us review the main ideas underlying Bayesian modelling, approximate inference, and a model of key importance to us: the Bayesian neural network.

2.1 Bayesian modelling

Given training inputs $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and their corresponding outputs $\mathbf{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$, in Bayesian (parametric) regression we would like to find the parameters $\boldsymbol{\omega}$ of a function $\mathbf{y} = \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x})$ that are *likely to have generated* our outputs. What parameters are likely to have generated our data? Following the Bayesian approach we would put some *prior* distribution over the space of parameters, $p(\boldsymbol{\omega})$. This distribution represents our prior belief as to which parameters are likely to have generated our data before we observe any data points. Once some data is observed, this distribution will be transformed to capture the more likely and less likely parameters given the observed data points. For this we further need to define a likelihood distribution $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})$—the probabilistic model by which the inputs generate the outputs given some parameter setting $\boldsymbol{\omega}$.

For classification tasks we may assume a softmax likelihood,

$$p\big(y = d|\mathbf{x}, \boldsymbol{\omega}\big) = \frac{\exp(f_d^{\boldsymbol{\omega}}(\mathbf{x}))}{\sum_{d'} \exp(f_{d'}^{\boldsymbol{\omega}}(\mathbf{x}))}$$

or a Gaussian likelihood for regression:

$$p\Big(\mathbf{y}|\mathbf{x},\boldsymbol{\omega}\Big) = \mathcal{N}(\mathbf{y};\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}),\tau^{-1}I) \tag{2.1}$$

with model precision τ. This can be seen as corrupting the model output with observation noise with variance τ^{-1}.

Given a dataset $\mathbf{X},\mathbf{Y}$, we then look for the *posterior* distribution over the space of parameters by invoking Bayes' theorem:

$$p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})}.$$

This distribution captures the most probable function parameters given our observed data. With it we can predict an output for a new input point $\mathbf{x}^*$ by integrating

$$p(\mathbf{y}^*|\mathbf{x}^*,\mathbf{X},\mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\mathrm{d}\boldsymbol{\omega}.$$

This process is known as *inference*[1].

A key component in posterior evaluation is the normaliser, also called *model evidence*:

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})p(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}. \tag{2.2}$$

Performing this integration is also referred to as *marginalising* the likelihood over $\boldsymbol{\omega}$, which is the origin of the alternative name for the model evidence: *marginal likelihood*. Marginalisation can be done analytically for simple models such as Bayesian linear regression. In such models the likelihood is *conjugate* to the prior, and the integral can be solved with known tools in calculus. Marginalisation is the bread and butter of Bayesian modelling, and ideally we would want to marginalise over all uncertain quantities—i.e. average w.r.t. all possible model parameter values $\boldsymbol{\omega}$, each weighted by its plausibility $p(\boldsymbol{\omega})$.

But with more interesting models (even basis function regression when the basis functions are not fixed) this marginalisation cannot be done analytically. In such cases an approximation is needed.

[1]Note that "inference" in Bayesian modelling has a different meaning to that in deep learning. In Bayesian modelling "inference" is the process of integration over model parameters. This means that "approximate inference" can involve optimisation at training time (approximating this integral). This is in contrast to deep learning literature where "inference" often means model evaluation at test time alone.

2.1.1 Variational inference

The true posterior $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ cannot usually be evaluated analytically. Instead we define an approximating *variational* distribution $q_\theta(\boldsymbol{\omega})$, parametrised by θ, whose structure is easy to evaluate. We would like our approximating distribution to be as close as possible to the posterior distribution obtained from the original model. We thus minimise the Kullback–Leibler (KL) divergence [Kullback, 1959; Kullback and Leibler, 1951] w.r.t. θ, intuitively a measure of similarity between two distributions:

$$\mathrm{KL}(q_\theta(\boldsymbol{\omega}) \,||\, p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})) = \int q_\theta(\boldsymbol{\omega}) \log \frac{q_\theta(\boldsymbol{\omega})}{p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})} \mathrm{d}\boldsymbol{\omega}. \tag{2.3}$$

Note that this integral is only defined when $q_\theta(\boldsymbol{\omega})$ is absolutely continuous w.r.t. $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ (i.e. for every measurable set A, $p(A|\mathbf{X}, \mathbf{Y}) = 0$ implies $q_\theta(A) = 0$). We denote by $q^*_\theta(\boldsymbol{\omega})$ the minimum of this optimisation objective (often a local minimum).

Minimising the KL divergence allows us to approximate the predictive distribution as

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \approx \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} =: q^*_\theta(\mathbf{y}^*|\mathbf{x}^*). \tag{2.4}$$

KL divergence minimisation is also equivalent to maximising the *evidence lower bound* (ELBO) w.r.t. the variational parameters defining $q_\theta(\boldsymbol{\omega})$,

$$\mathcal{L}_{\mathrm{VI}}(\theta) := \int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} - \mathrm{KL}(q_\theta(\boldsymbol{\omega}) \,||\, p(\boldsymbol{\omega})) \leq \log p(\mathbf{Y}|\mathbf{X}) = \log \text{evidence}, \tag{2.5}$$

which defines the objective we will refer to henceforth. Maximising the first term in this last equation (referred to as the *expected log likelihood*) encourages $q_\theta(\boldsymbol{\omega})$ to explain the data well, while minimising the second term (referred to as the *prior KL*) encourages $q_\theta(\boldsymbol{\omega})$ to be as close as possible to the prior. This acts as an "Occam razor" term and penalises complex distributions $q_\theta(\boldsymbol{\omega})$.

This procedure is known as *variational inference* (VI), a standard technique in Bayesian modelling [Jordan et al., 1999]. Variational inference replaces the Bayesian modelling marginalisation with optimisation, i.e. we replace the calculation of integrals with that of derivatives. But compared to the optimisation approaches often used in deep learning, in this setting we optimise over distributions instead of point estimates[2]. This approach preserves many of the advantages of Bayesian modelling (such as the

[2]Note that optimisation in the deep learning sense can be recovered by setting the approximating distribution as a delta $q_\theta(\boldsymbol{\omega}) := \delta(\boldsymbol{\omega} - \theta)$.

balance between complex models and models that explain the data well), and results in probabilistic models that capture model uncertainty.

The calculation of derivatives is often much easier than that of integrals, which makes many approximations tractable. But even though this procedure makes inference analytical for a large class of models, it still lacks in many ways. This technique does not scale to large data (evaluating $\int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}$ requires calculations over the entire dataset), and the approach does not adapt to complex models (models in which this last integral cannot be evaluated analytically). Recent advances in VI allow us to circumvent these difficulties, and we will get back to this topic later in §3.1. But first we review a model of key importance to us: the Bayesian neural network.

2.2 Bayesian neural networks

First suggested in the '90s and studied extensively since then [MacKay, 1992b; Neal, 1995], Bayesian neural networks (BNNs, Bayesian NNs) offer a probabilistic interpretation of deep learning models by inferring distributions over the models' weights. The model offers robustness to over-fitting, uncertainty estimates, and can easily learn from small datasets.

Bayesian NNs place a prior distribution over a neural network's weights, which induces a distribution over a parametric set of functions. Given weight matrices $\mathbf{W}_i$ and bias vectors $\mathbf{b}_i$ for layer i, we often place standard matrix Gaussian prior distributions over the weight matrices, $p(\mathbf{W}_i) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and often assume a point estimate for the bias vectors for simplicity. Likelihood specification often follows the standard Bayesian literature (such as the softmax likelihood or Gaussian likelihood, §2.1).

Bayesian neural networks are easy to formulate, but difficult to perform inference in. Many have tried over the years, to varying degrees of success. This is surveyed next.

2.2.1 Brief history

In their work at AT&T Bell Laboratories in 1987, Denker, Schwartz, Wittner, Solla, Howard, Jackel, and Hopfield [1987] looked at the general problem of learning from examples. They extended on an already vast existing literature in neural networks research, and proposed a new way to train these. In essence, Denker et al. [1987, page 904] proposed placing a prior distribution over the space of weights (and used a uniform distribution on a compact space). Denoting all possible (binary) inputs as $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$, they then mapped each weight configuration to a set of corresponding network outputs

$\{\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_N\}$. This allowed them to integrate over the weights and obtain a marginal probability for each set $\{\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_N\}$. Given observed training data, the probabilities of inconsistent network weights were set to zero, leading to an updated marginal probability for each set $\{\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_N\}$. These marginal probabilities were then used to calculate the efficiency of a network (by calculating its entropy).

Working from AT&T Bell Laboratories as well, Tishby, Levin, and Solla [1989] extended on the ideas of Denker et al. [1987] and developed a statistical framework to reason about network generalisation error. As far as I could find, this is the earliest citation for what one would consider today to be a "Bayesian neural network". Tishby et al. [1989] showed that the only statistical interpretation of a NN Euclidean loss is as maximum likelihood w.r.t. a Gaussian likelihood over the network outputs. They proceeded to define a prior distribution over the network weights, and showed that inference could theoretically be performed through the invocation of Bayes' theorem. This was used to propose a quantity dependent on the training set alone that would correlate to the network's generalisation error on a test set (although the problem of inference practicality was overlooked).

In more work coming out from AT&T Bell Laboratories, Denker and LeCun [1991] extended on Tishby et al. [1989]'s ideas and suggested the use of Laplace's method to approximate the posterior of the Bayesian NN. Denker and LeCun [1991] used the (back then) newly suggested "backpropagation" technique, and optimised the neural network weights to find a mode. They then fitted a Gaussian to the discovered mode, with the width of the Gaussian determined by the Hessian of the likelihood at that mode.

Working from California Institute of Technology, MacKay [1992b] performed an extensive study of Bayesian NNs. MacKay [1992b] advocated the use of model evidence for model comparison, and obtained this quantity following the approximation of Denker and LeCun [1991]. Using an array of experiments with different model sizes and model configurations, MacKay [1992b] showed that model evidence correlates to generalisation error, and thus can be used to select model size. MacKay [1992b] further showed that model misspecification can lead to Bayes failure, where model evidence does not indicate model generalisation. As he showed, this could happen for example when the priors of multiple weight layers are tied together, and thus low probability could be given to models with large input weight magnitude together with small output weight magnitude unjustifiably.

As a way of model regularisation, Hinton and Van Camp [1993] (working from Toronto) suggested the use of minimum description length (MDL) to penalise high amounts of information contained in a network's weights. They showed that for a single hidden

layer NN it is possible to compute their suggested (and somewhat ad-hoc) objective analytically. This technique can be seen as the first variational inference approximation to Bayesian NNs—even though the suggested technique is motivated through information theoretic foundations, the resulting optimisation objective is identical to VI's ELBO (as will be explained below).

In his thesis, Neal [1995] developed alternative inference approximations for Bayesian NNs based on Monte Carlo (MC) techniques. Hamiltonian Monte Carlo (HMC, also *Hybrid Monte Carlo*) was suggested for posterior inference, a technique based on dynamical simulation that does not rely on any prior assumptions about the form of the posterior distribution. Neal [1995] attempted to reproduce MacKay [1992b]'s experiments which relied on Laplace's method [Denker and LeCun, 1991]. Neal [1995, page 122] validated some of MacKay [1992b]'s experiments, but could not reproduce some others, presumably because of the approximation error of Laplace's method. In addition to this work, Neal [1995] further studied different prior distributions in Bayesian NNs, and showed that in the limit of the number of units the model would converge to various stable processes, depending on the prior used (for example, the model would converge to a Gaussian process when a Gaussian prior is used).

One last key development relevant to our setting stems from work by Barber and Bishop [1998]. In the work, Barber and Bishop [1998] developed Hinton and Van Camp [1993]'s MDL approximation under a VI interpretation, and replaced Hinton and Van Camp [1993]'s diagonal covariance matrices with full covariance matrices. Barber and Bishop [1998] further highlighted the fact that the obtained objective forms a lower bound to the model evidence. Lastly, Barber and Bishop [1998] placed gamma priors over the network hyper-parameters. They then performed VI with free-form variational distributions over the hyper-parameters, and derived their optimal form. This allowed the model evidence (the quantity that the ELBO bounds) to remain constant w.r.t. the changing hyper-parameter values, since the hyper-parameters are always averaged w.r.t. their approximating distribution.

Remark (Multimodal or unimodal approximating distributions?)**.** MacKay [1992b] used a unimodal Gaussian distribution to fit the posterior and reasoned that this already gives us much more than a maximum likelihood estimate (MLE). That's because the width of the fitted Gaussian acts as an "Occam razor" and penalises complex models (in essence capturing the ratio between the posterior parameters' distribution volume to prior parameters' distribution volume). Neal [1995] criticised the simplistic unimodal approximation though, and argued that the approximation

only works in low dimensional problems (and therefore we should use HMC that doesn't place any assumptions over the posterior structure, and able to capture complicated multimodal posterior and predictive distributions).
But fitting the posterior over the weights of a Bayesian NN with a unimodal approximating distribution does not mean the predictive distribution would be unimodal! imagine for simplicity that the intermediate feature output from the first layer is a unimodal distribution (a uniform for example) and let's say, for the sake of argument, that the layers following that are modelled with delta distributions (or Gaussians with very small variances). Given enough follow-up layers we can capture any function to arbitrary precision—including the inverse cumulative distribution function (CDF) of any multimodal distribution. Passing our uniform output from the first layer through the rest of the layers—in effect transforming the uniform with this inverse CDF—would give a multimodal predictive distribution.

These approaches represent important first steps towards practical inference in Bayesian NNs. But these are difficult to adapt to modern needs found in the field. Next we review more modern approaches to approximate inference in Bayesian NNs.

2.2.2 Modern approximate inference

Modern research in Bayesian NNs often relies on either different flavours of variational inference, or sampling based techniques. Each approach has its merits, but has its limitations as well. Next we will survey some of the recently suggested approaches.

For variational inference, modern approaches follow the work of Hinton and Van Camp [1993] closely. This work relied on a fully factorised approximation—the approximating distribution assumes independence of each weight scalar in each layer from all other weights. We will go over this approach in more detail from the VI perspective. This will be followed by a survey of extensions to this approach.

Recall that we are interested in finding the distribution of weight matrices (parametrising our functions) that have generated our data. This is the posterior over the weights given our observables $\mathbf{X}, \mathbf{Y}$: $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$. This posterior is not tractable in general, and we use variational inference to approximate it. For this we need to define an approximating variational distribution $q_\theta(\boldsymbol{\omega})$, and then minimise the KL divergence between the

approximating distribution and the full posterior[3]:

$$\begin{aligned}\mathrm{KL}\Big(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\Big) &\propto -\int q_\theta(\boldsymbol{\omega})\log p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \\ &= -\sum_{i=1}^{N}\int q_\theta(\boldsymbol{\omega})\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))\end{aligned} \tag{2.6}$$

with $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$ the model output on input $\mathbf{x}_i$ and with the terms summed-over in the last equation referred to as expected log likelihoods. Hinton and Van Camp [1993] defined $q_\theta(\boldsymbol{\omega})$ to factorise over the weights:

$$q_\theta(\boldsymbol{\omega}) = \prod_{i=1}^{L} q_\theta(\mathbf{W}_i) = \prod_{i=1}^{L}\prod_{j=1}^{K_i}\prod_{k=1}^{K_{i+1}} q_{m_{ijk},\sigma_{ijk}}(w_{ijk}) = \prod_{i,j,k}\mathcal{N}(w_{ijk}; m_{ijk}, \sigma_{ijk}^2).$$

Optimising the objective is challenging, since the expected log likelihood is intractable for most BNN model structures. Therefore Hinton and Van Camp [1993] only demonstrated the technique with a single hidden layer in which case the optimisation objective is analytical.

Even when the optimisation objective is analytical, this approximation can work quite badly in practice. This could possibly be attributed to the method losing important information about weight correlations. Barber and Bishop [1998], by modelling correlations between the weights, managed to improve on [Hinton and Van Camp, 1993]. But this came with the price of increased computational complexity. The method now required the representation of covariance matrices with a number of entries quadratic in the number of weights in the model. This is impractical for most modern models, since the number of model parameters in modern deep learning tends to be as large as modern hardware would allow. With the additional constraint of having to handle large amounts of data, plain VI could not scale to modern needs and was thus mostly forgotten.

In recent work, Graves [2011] has attempted to answer the difficulties above. Graves [2011] used *data sub-sampling* techniques in a fully factorised VI objective (although this was developed in the context of MDL). This allowed the technique to scale to large amounts of data. Graves [2011] approximated the intractable expected log likelihood with Monte Carlo estimates [Opper and Archambeau, 2009], which allowed the technique to scale to more complex models, going beyond the single layer restriction[4]. Ironically, even though Bayesian NNs were not mentioned in the paper even once, Graves [2011]'s

[3]We slightly abuse standard notation here, and use $A \propto B$ to mean that A is identical to B up to some constant c: $A = B + c$ rather than $A = cB$.

[4]These are two techniques that started gaining in popularity within VI at the time, and will be reviewed in more detail in the next chapter.

work can be seen as a big step forward from previous research on approximate inference in BNNs. For the first time a *practical* technique was proposed, which scaled to complex models and big data. Sadly, similarly to Hinton and Van Camp [1993], the technique still performed badly in practice [Hernandez-Lobato and Adams, 2015], perhaps because of the lack of correlations over the weights. As a result it was not picked-up and actively extended-on by the community for a long period of time.

In more recent work, done in parallel to the one presented in this thesis, Blundell et al. [2015] has built on Graves [2011]'s approach, re-parametrising the expected log likelihood MC estimates following Kingma and Welling [2013]. Blundell et al. [2015] further changed the BNN model itself by putting a mixture of Gaussians prior over each weight and optimised the mixture components. This allowed them to improve model performance compared to Graves [2011], and match that of existing approaches in the field. But even this variational approach can still be computationally expensive—the use of Gaussian approximating distributions increases the number of model parameters considerably, without increasing model capacity by much. Blundell et al. [2015] for example used Gaussian distributions for Bayesian NN posterior approximation, doubling the number of model parameters. This makes the approach difficult for use with large complex models since the increase in the number of parameters can be too costly, especially on systems with limited memory.

An alternative approach to variational inference makes use of expectation propagation [Hernandez-Lobato and Adams, 2015]. Hernandez-Lobato and Adams [2015]'s probabilistic back propagation (PBP) has improved considerably on VI approaches such as [Graves, 2011] both in root mean square error (RMSE) and in uncertainty estimation. Closely related to PBP are the recently suggested approximate inference techniques based on α-divergence minimisation [Hernández-Lobato et al., 2016; Li and Turner, 2016; Minka, 2005]. Most of these are still under active development, and not used in practice yet (with the exception of [Hernández-Lobato et al., 2016], which was used in reinforcement learning [Depeweg et al., 2016]). These approximate inference techniques offer alternative minimisation objectives to VI's ELBO, relying on various forms of Rényi's α-divergence [Rényi et al., 1961]. The approximating distributions used in these papers are still rather simple, with [Depeweg et al., 2016; Hernández-Lobato et al., 2016] for example making use of fully factorised Gaussian distributions. The main motivation behind the techniques is to avoid VI's mode-seeking behaviour, and indeed in fitting a more dispersed distribution (a distribution spreading its mass on larger parts of the support), these techniques seem to sacrifice their estimation of the dominant modes of the posterior. But in performing

predictions we often care about finding modes and exploring around them, rather than interpolating between different modes.

An alternative line of research has extended on Neal [1995]'s HMC rather than the work of Hinton and Van Camp [1993]. HMC makes use of Hamiltonian dynamics in MCMC [Duane et al., 1987], following Newton's laws of motion [Newton, 1687]. Neal [1995]'s use of HMC in statistics was to generate samples from a model's posterior which would be difficult to sample from directly. But HMC can be difficult to work with in practice, as setting the leapfrog step sizes can be somewhat of an art. Further, the method does not scale to large data. This is because the method requires us to calculate gradients for the likelihood evaluated on the entire dataset [Neal, 2011]. The Langevin method is a simplification of Hamiltonian dynamics where only a single leapfrog step is used. The use of a single step simplifies the inference method itself considerably, and allows it to be scaled to large data. The latter is achieved through the use of stochastic estimates of the gradient of the likelihood, replacing the gradients over the entire dataset [Welling and Teh, 2011]. These stochastic estimates of the likelihood are similar to the ones used to scale-up VI used by Graves [2011]. Welling and Teh [2011]'s scalable Langevin method was named Stochastic Gradient Langevin Dynamics (SGLD). The SGLD technique generates a set of samples $\{\widehat{\boldsymbol{\omega}}_t\}$ from the model's posterior over the random variable $\boldsymbol{\omega}$ by adding stochastic gradient steps to the previously generated samples:

$$\begin{aligned}\Delta\boldsymbol{\omega} &= \frac{\epsilon}{2}\left(\nabla \log p(\boldsymbol{\omega}) + \frac{N}{M}\sum_{i\in S}\nabla \log p(\mathbf{y}_i|\mathbf{x}_i,\boldsymbol{\omega})\right) + \eta \\ \eta &\sim \mathcal{N}(0,\epsilon).\end{aligned}$$

Here S is a randomly sampled set of M indices from $\{1,...,N\}$ and ϵ is decreased in magnitude following the Robbins–Monro equations [Robbins and Monro, 1951]. Unlike fully-factorised VI, this approach can capture weight correlations since the approximate posterior structure need not be specified. However, the main difficulty with SGLD is that it often collapses to a single mode and explores that mode alone. This is because ϵ decreases so rapidly that the probability of the method to jump outside of the region of a mode and converge to another mode is extremely small. Even though that in the limit of time the entire support of the distribution will be explored, in practice the method explores a single mode, similar to VI. Further, the method generates correlated samples—which means that many samples need to be generated, since many intermediate samples have to be discarded. These factors somewhat limit SGLD's practicality.

One last approach I shall review here cannot technically be considered as approximate inference in BNNs, but nonetheless can be used to estimate model uncertainty. The technique uses an ensemble of deterministic models, meaning that each model in the ensemble produces a point estimate rather than a distribution. It works by independently training many randomly initialised instances of a model on the same dataset (or different random subsets in the case of bootstrapping), and given an input test point, evaluating the sample variance of the outputs from all deterministic models [Osband et al., 2016]. Even though this approach is more computationally efficient than many Bayesian approaches to model uncertainty (apart from the need to represent the parameters of multiple models), its produced uncertainty estimates lack in many ways as explained in the next illustrative example. To see this, let's see what would happen if each deterministic model were to be given by an RBF network (whose predictions coincide with the predictive mean of a Gaussian process with a squared-exponential (SE) covariance function). An RBF network predicts zero for test points far from the training data. This means that in an ensemble of RBF networks, each and every network will predict zero for a given test point far from the training data. As a result, the sample variance of this technique will be zero at the given test point. The ensemble of models will have very high confidence in its prediction of zero even though the test point lies far from the data! This limitation can be alleviated by using an ensemble of *probabilistic* models instead of *deterministic* models. Even though the RBF network's predictions coincide with the predictive mean of the SE Gaussian process, by using a Gaussian process we could also make use of its *predictive variance*. The Gaussian process predictions far from the training data will have large model uncertainty. In the ensemble, we would thus wish to take into account each model's confidence as well as its mean (by sampling an output from each model's predictive distribution before calculating our sample variance). These ideas are assessed below in the context of Bayesian recurrent neural networks (§4.5).

2.2.3 Challenges

In the introduction (§1.5) we discussed what makes an approximate inference technique practical. We concluded that a technique should:

1. scale well to large data,
2. easily adapt to complex models (that can be as big as modern architecture would allow, or complex pipelines combining many building blocks),
3. not necessitate the change of existing model architectures (or objectives),

4. and should be easy for non-experts to use and understand.

The practicality of the techniques above is mixed, as discussed per technique. HMC for example, even though shown to obtain good results, does not scale to large data [Neal, 1995], and it is difficult to explain the technique to non-experts. α-divergence based techniques share this latter difficulty as well, limiting their use by non-experts.

In the next chapter we will develop an approximate inference technique for Bayesian NNs which will satisfy the requirements above. This will be demonstrated in the following chapters, where a large number of real-world use cases will be presented. Even more interesting, it will be shown that most modern deep learning models have been performing approximate Bayesian inference all along.

Chapter 3

Bayesian Deep Learning

In previous chapters we reviewed Bayesian neural networks (BNNs) and historical techniques for approximate inference in these, as well as more recent approaches. We discussed the advantages and disadvantages of different techniques, examining their practicality. This, perhaps, is the most important aspect of modern techniques for approximate inference in BNNs. The field of deep learning is pushed forward by practitioners, working on real-world problems. Techniques which cannot scale to complex models with potentially millions of parameters, scale well with large amounts of data, need well studied models to be radically changed, or are not accessible to engineers, will simply perish.

In this chapter we will develop on the strand of work of [Graves, 2011; Hinton and Van Camp, 1993], but will do so from the Bayesian perspective rather than the information theory one. Developing Bayesian approaches to deep learning, we will tie approximate BNN inference together with deep learning stochastic regularisation techniques (SRTs) such as dropout. These regularisation techniques are used in many modern deep learning tools, allowing us to offer a practical inference technique.

We will start by reviewing in detail the tools used by Graves [2011], and extend on these with recent research. In the process we will comment and analyse the variance of several stochastic estimators used in variational inference (VI). Following that we will tie these derivations to SRTs, and propose practical techniques to obtain model uncertainty, even from existing models. We finish the chapter by developing specific examples for image based models (CNNs) and sequence based models (RNNs). These will be demonstrated in chapter 5, where we will survey recent research making use of the suggested tools in real-world problems.

3.1 Advanced techniques in variational inference

We start by reviewing recent advances in VI. We are interested in the posterior over the weights given our observables $\mathbf{X}, \mathbf{Y}$: $p\big(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}\big)$. This posterior is not tractable for a Bayesian NN, and we use variational inference to approximate it. Recall our minimisation objective (eq. (2.6)),

$$\mathcal{L}_{\text{VI}}(\theta) := -\sum_{i=1}^{N} \int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) \mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})). \tag{3.1}$$

Evaluating this objective poses several difficulties. First, the summed-over terms $\int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) \mathrm{d}\boldsymbol{\omega}$ are not tractable for BNNs with more than a single hidden layer. Second, this objective requires us to perform computations over the entire dataset, which can be too costly for large N.

To solve the latter problem, we may use data sub-sampling (also referred to as *mini-batch optimisation*). We approximate eq. (3.1) with

$$\widehat{\mathcal{L}}_{\text{VI}}(\theta) := -\frac{N}{M} \sum_{i \in S} \int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) \mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \tag{3.2}$$

with a random index set S of size M.

The data sub-sampling approximation forms an unbiased stochastic estimator to eq. (3.1), meaning that $\mathbb{E}_S[\widehat{\mathcal{L}}_{\text{VI}}(\theta)] = \mathcal{L}_{\text{VI}}(\theta)$. We can thus use a stochastic optimiser to optimise this stochastic objective, and obtain a (local) optimum θ^* which would be an optimum to $\mathcal{L}_{\text{VI}}(\theta)$ as well [Robbins and Monro, 1951]. This approximation is often used in the deep learning literature, and was suggested by Hoffman et al. [2013] in the VI context (surprisingly) only recently[1].

The remaining difficulty with the objective in eq. (3.1) is the evaluation of the expected log likelihood $\int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) \mathrm{d}\boldsymbol{\omega}$. Monte Carlo integration of the integral has been attempted by some, to varying degrees of success. We will review three approaches used in the VI literature and analyse their variance.

3.1.1 Monte Carlo estimators in variational inference

We often use Monte Carlo (MC) estimation in variational inference to estimate the expected log likelihood (the integral in eq. (3.2)). But more importantly we wish to estimate the expected log likelihood's derivatives w.r.t. the approximating distribution

[1] Although online VI methods processing one point at a time have a long history [Ghahramani and Attias, 2000; Sato, 2001]

parameters θ. This allows us to optimise the objective and find the optimal parameters θ^*. There exist three main techniques for MC estimation in the VI literature (a brief survey of the literature was collected by [Schulman et al., 2015]). These have very different characteristics and variances for the estimation of the expected log likelihood and its derivative. Here we will contrast all three techniques in the context of VI and analyse them both empirically and theoretically.

To present the various techniques we will consider the general case of estimating the *integral derivative*:

$$I(\theta) = \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x \tag{3.3}$$

which arises when we optimise eq. (3.2) (we will also refer to a stochastic estimator of this quantity as a *stochastic derivative estimator*). Here $f(x)$ is a function defined on the reals, differentiable almost everywhere (a.e., differentiable on $\mathbb{R}$ apart from a zero measure set), and $p_\theta(x)$ is a probability density function (pdf) parametrised by θ from which we can easily generate samples. We assume that the integral exists and is finite, and that $f(x)$ does not depend on θ. Note that we shall write $f'(x)$ when we differentiate $f(x)$ w.r.t. its input (i.e. $\frac{\partial}{\partial x}f(x)$, in contrast to the differentiation of $f(x)$ w.r.t. other variables).

We further use $p_\theta(x) = \mathcal{N}(x;\mu,\sigma^2)$ as a concrete example, with $\theta = \{\mu,\sigma\}$. In this case, we will refer to an estimator of (3.3) as the *mean derivative estimator* (for some function $f(x)$) when we differentiate w.r.t. $\theta = \mu$, and the *standard deviation derivative estimator* when we differentiate w.r.t. $\theta = \sigma$.

Three MC estimators for eq. (3.3) are used in the VI literature:

1. The *score function* estimator (also known as a *likelihood ratio estimator* and *Reinforce*, [Fu, 2006; Glynn, 1990; Paisley et al., 2012; Williams, 1992]) relies on the identity $\frac{\partial}{\partial\theta}p_\theta(x) = p_\theta(x)\frac{\partial}{\partial\theta}\log p_\theta(x)$ and follows the parametrisation:

$$\begin{aligned}\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x &= \int f(x)\frac{\partial}{\partial\theta}p_\theta(x)\mathrm{d}x \\ &= \int f(x)\frac{\partial\log p_\theta(x)}{\partial\theta}p_\theta(x)\mathrm{d}x\end{aligned} \tag{3.4}$$

   leading to the unbiased stochastic estimator $\widehat{I}_1(\theta) = f(x)\frac{\partial\log p_\theta(x)}{\partial\theta}$ with $x\sim p_\theta(x)$, hence $\mathbb{E}_{p_\theta(x)}[\widehat{I}_1(\theta)] = I(\theta)$. Note that the first transition is possible since x and $f(x)$ do not depend on θ, and only $p_\theta(x)$ depends on it. This estimator is simple and applicable with discrete distributions, but as Paisley et al. [2012] identify, it

has rather high variance. When used in practice it is often coupled with a variance reduction technique.

2. Eq. (3.3) can be re-parametrised to obtain an alternative MC estimator, which we refer to as a *pathwise derivative* estimator (this estimator is also referred to in the literature as the *re-parametrisation trick*, *infinitesimal perturbation analysis*, and *stochastic backpropagation* [Glasserman, 2013; Kingma and Welling, 2013, 2014; Rezende et al., 2014; Titsias and Lázaro-Gredilla, 2014]). Assume that $p_\theta(x)$ can be re-parametrised as $p(\epsilon)$, a parameter-free distribution, s.t. $x = g(\theta, \epsilon)$ with a deterministic *differentiable* bivariate transformation $g(\cdot, \cdot)$. For example, with our $p_\theta(x) = \mathcal{N}(x; \mu, \sigma^2)$ we have $g(\theta, \epsilon) = \mu + \sigma\epsilon$ together with $p(\epsilon) = \mathcal{N}(\epsilon; 0, I)$. Then the following estimator arises:

$$\widehat{I}_2(\theta) = f'(g(\theta, \epsilon))\frac{\partial}{\partial\theta}g(\theta, \epsilon). \tag{3.5}$$

Here $\mathbb{E}_{p(\epsilon)}[\widehat{I}_2(\theta)] = I(\theta)$.

Note that compared to the estimator above where $f(x)$ is used, in this case we use its derivative $f'(x)$. In the Gaussian case, both the derivative w.r.t. μ as well as the derivative w.r.t. σ use $f(x)$'s first derivative (note the substitution of ϵ with $x = \mu + \sigma\epsilon$):

$$\frac{\partial}{\partial\mu}\int f(x)p_\theta(x)\mathrm{d}x = \int f'(x)p_\theta(x)\mathrm{d}x,$$
$$\frac{\partial}{\partial\sigma}\int f(x)p_\theta(x)\mathrm{d}x = \int f'(x)\frac{(x-\mu)}{\sigma}p_\theta(x)\mathrm{d}x,$$

i.e. $\widehat{I}_2(\mu) = f'(x)$ and $\widehat{I}_2(\sigma) = f'(x)\frac{(x-\mu)}{\sigma}$ with $\mathbb{E}_{p_\theta(x)}[\widehat{I}_2(\mu)] = I(\mu)$ and $\mathbb{E}_{p_\theta(x)}[\widehat{I}_2(\sigma)] = I(\sigma)$. An interesting question arises when we compare this estimator ($\widehat{I}_2(\theta)$) to the one above ($\widehat{I}_1(\theta)$): why is it that in one case we use the function $f(x)$, and in the other use its derivative? This is answered below, together with an alternative derivation to that of Kingma and Welling [2013, section 2.4].

The pathwise derivative estimator seems to (empirically) result in a lower variance estimator than the one above, although to the best of my knowledge no proof to this has been given in the literature. An analysis of this parametrisation, together with examples where the estimator has *higher* variance than that of the score function estimator is given below.

3. Lastly, it is important to mention the estimator given in [Opper and Archambeau, 2009, eq. (6-7)] (studied in [Rezende et al., 2014] as well). Compared to both estimators above, Opper and Archambeau [2009] relied on the characteristic function of the Gaussian distribution, restricting the estimator to Gaussian $p_\theta(x)$ alone. The resulting mean derivative estimator, $\frac{\partial}{\partial \mu} \int f(x) p_\theta(x) \mathrm{d}x$, is identical to the one resulting from the pathwise derivative estimator.

   But when differentiating w.r.t. σ, the estimator depends on $f(x)$'s second derivative [Opper and Archambeau, 2009, eq. (19)]:

$$\frac{\partial}{\partial \sigma} \int f(x) p_\theta(x) \mathrm{d}x = 2\sigma \cdot \frac{1}{2} \int f''(x) p_\theta(x) \mathrm{d}x \tag{3.6}$$

   i.e. $\widehat{I}_3(\sigma) = \sigma f''(x)$ with $\mathbb{E}_{p_\theta(x)}[\widehat{I}_3(\sigma)] = I(\sigma)$. This is in comparison to the estimators above that use $f(x)$ or its first derivative. We refer to this estimator as a *characteristic function* estimator.

These three MC estimators are believed to have decreasing estimator variances (1) > (2) > (3). Before we analyse this variance, we offer an alternative derivation to Kingma and Welling [2013]'s derivation of the pathwise derivative estimator.

Remark (Auxiliary variable view of the pathwise derivative estimator)**.** Kingma and Welling [2013]'s derivation of the pathwise derivative estimator was obtained through a change of variables. Instead we justify the estimator through a construction relying on an auxiliary variable augmenting the distribution $p_\theta(x)$.
Assume that the distribution $p_\theta(x)$ can be written as $\int p_\theta(x, \epsilon) \mathrm{d}\epsilon = \int p_\theta(x|\epsilon) p(\epsilon) \mathrm{d}\epsilon$, with $p_\theta(x|\epsilon) = \delta(x - g(\theta, \epsilon))$ a dirac delta function. Then

$$\begin{aligned}
\frac{\partial}{\partial \theta} \int f(x) p_\theta(x) \mathrm{d}x &= \frac{\partial}{\partial \theta} \int f(x) \Big(\int p_\theta(x, \epsilon) \mathrm{d}\epsilon \Big) \mathrm{d}x \\
&= \frac{\partial}{\partial \theta} \int f(x) p_\theta(x|\epsilon) p(\epsilon) \mathrm{d}\epsilon \mathrm{d}x \\
&= \frac{\partial}{\partial \theta} \int \Big(\int f(x) \delta\big(x - g(\theta, \epsilon)\big) \mathrm{d}x \Big) p(\epsilon) \mathrm{d}\epsilon.
\end{aligned}$$

Now, since $\delta(x - g(\theta, \epsilon))$ is zero for all x apart from $x = g(\theta, \epsilon)$,

$$\frac{\partial}{\partial \theta} \int \Big(\int f(x) \delta\big(x - g(\theta, \epsilon)\big) \mathrm{d}x \Big) p(\epsilon) \mathrm{d}\epsilon = \frac{\partial}{\partial \theta} \int f(g(\theta, \epsilon)) p(\epsilon) \mathrm{d}\epsilon$$

$$= \int \frac{\partial}{\partial\theta} f(g(\theta,\epsilon))p(\epsilon)\mathrm{d}\epsilon$$
$$= \int f'(g(\theta,\epsilon))\frac{\partial}{\partial\theta} g(\theta,\epsilon)p(\epsilon)\mathrm{d}\epsilon.$$

This derivation raises an interesting question: why is it that here the function $f(x)$ depends on θ, whereas in the above (the score function estimator) it does not? A clue into explaining this can be found through a measure theoretic view of the score function estimator:

$$\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x = \frac{\partial}{\partial\theta}\int \Big(f(x)p_\theta(x)\Big)\mathrm{d}\lambda(x)$$

where $\lambda(x)$ is the Lebesgue measure. Here the measure does not depend on θ, hence x does not depend on θ. Only the integrand $f(x)p_\theta(x)$ depends on θ, therefore the estimator depends on $\frac{\partial}{\partial\theta}p_\theta(x)$ and $f(x)$. This is in comparison to the pathwise derivative estimator:

$$\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x = \frac{\partial}{\partial\theta}\int f(x)\mathrm{d}p_\theta(x).$$

Here the integration is w.r.t. the measure $p_\theta(x)$, which depends on θ. As a result the random variable x as a measurable function depends on θ, leading to $f(x)$ being a measurable function depending on θ. Intuitively, the above can be seen as "stretching" and "contracting" the space following the density $p_\theta(x)$, leading to a function defined on this space to depend on θ.

3.1.2 Variance analysis of Monte Carlo estimators in variational inference

Next we analyse the estimator variance for the three estimators above using a Gaussian distribution $p_\theta(x) = \mathcal{N}(x;\mu,\sigma^2)$. We will refer to the estimators as the score function estimator (1), the pathwise derivative estimator (2), and the characteristic function estimator (3). We will alternate between the estimator names and numbers indistinguishably.

We begin with the observation that none of the estimators has the lowest variance for *all* functions $f(x)$. For each estimator there exists a function $f(x)$ such that the estimator would achieve lowest variance when differentiating w.r.t. μ, $\frac{\partial}{\partial\boldsymbol{\mu}}\int f(x)p_\theta(x)\mathrm{d}x$. Further, for each estimator there exists a function $f(x)$ (not necessarily the same) such that the estimator would achieve lowest variance when differentiating w.r.t. σ, $\frac{\partial}{\partial\boldsymbol{\sigma}}\int f(x)p_\theta(x)\mathrm{d}x$.

$f(x)$	Score function	Pathwise derivative	Character. function
$x + x^2$	$1.7 \cdot 10^1$	$\mathbf{4.0 \cdot 10^0}$	$\mathbf{4.0 \cdot 10^0}$
$\sin(x)$	$3.3 \cdot 10^{-1}$	$\mathbf{2.0 \cdot 10^{-1}}$	$\mathbf{2.0 \cdot 10^{-1}}$
$\sin(10x)$	$\mathbf{5.0 \cdot 10^{-1}}$	$5.0 \cdot 10^1$	$5.0 \cdot 10^1$

Table 3.1 $\frac{\partial}{\partial \boldsymbol{\mu}} \int f(x) p_\theta(x) \mathrm{d}x$ variance

$f(x)$	Score function	Pathwise derivative	Character. function
$x + x^2$	$8.6 \cdot 10^1$	$9.1 \cdot 10^0$	$\mathbf{0.0 \cdot 10^0}$
$\sin(x)$	$8.7 \cdot 10^{-1}$	$\mathbf{3.0 \cdot 10^{-1}}$	$4.3 \cdot 10^{-1}$
$\sin(10x)$	$\mathbf{1.0 \cdot 10^0}$	$5.0 \cdot 10^1$	$5.0 \cdot 10^3$

Table 3.2 $\frac{\partial}{\partial \boldsymbol{\sigma}} \int f(x) p_\theta(x) \mathrm{d}x$ variance

Table 3.3 Estimator variance for various functions $f(x)$ for the score function estimator, the pathwise derivative estimator, and the characteristic function estimator ((1), (2), and (3) above). On the left is mean derivative estimator variance, and on the right is standard deviation derivative estimator variance, both w.r.t. $p_\theta = \mathcal{N}(\mu, \sigma^2)$ and evaluated at $\mu = 0, \sigma = 1$. In bold is lowest estimator variance.

We assess empirical sample variance of $T = 10^6$ samples for the mean and standard deviation derivative estimators. Tables 3.1 and 3.2 show estimator sample variance for the integral derivative w.r.t. μ and σ respectively, for three different functions $f(x)$. Even though all estimators result in roughly the same means, their variances differ considerably.

For functions with slowly varying derivatives in a neighbourhood of zero (such as the smooth function $f(x) = x + x^2$) we have that the estimator variance obeys (1) > (2) > (3). Also note that mean variance for (2) and (3) are identical, and that σ derivative variance under the characteristic function estimator in this case is zero (since the second derivative for this function is zero). Lastly, note that even though $f(x) = \sin(x)$ is smooth with bounded derivatives, the similar function $f(x) = \sin(10x)$ has variance (3) > (2) > (1). This is because the derivative of the function has high magnitude and varies much near zero.

I will provide a simple property a function has to satisfy for it to have lower variance under the pathwise derivative estimator and the characteristic function estimator than the score function estimator. This will be for the mean derivative estimator.

Proposition 1. *Let $f(x)$, $f'(x)$, $f''(x)$ be real-valued functions s.t. $f(x)$ is an indefinite integral of $f'(x)$, and $f'(x)$ is an indefinite integral of $f''(x)$. Assume that $Var_{p_\theta(x)}((x-\mu)f(x)) < \infty$, and $Var_{p_\theta(x)}(f'(x)) < \infty$, as well as $\mathbb{E}_{p_\theta(x)}(|(x-\mu)f'(x) + f(x)|) < \infty$ and $\mathbb{E}_{p_\theta(x)}(|f''(x)|) < \infty$, with $p_\theta(x) = \mathcal{N}(\mu, \sigma^2)$.*

If it holds that

$$\mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x) + f(x)\Big)^2 - \sigma^4 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big) \geq 0,$$

then the pathwise derivative and the characteristic function mean derivative estimators w.r.t. the function $f(x)$ will have lower variance than the score function estimator.

Before proving the proposition, I will give some intuitive insights into what the condition means. For this, assume for simplicity that $\mu = 0$ and $\sigma = 1$. This gives the simplified condition

$$\mathbb{E}_{p_\theta(x)}\Big(xf'(x) + f(x)\Big)^2 \geq \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big).$$

First, observe that all expectations are taken over $p_\theta(x)$, meaning that we only care about the functions' average behaviour near zero. Second, a function $f(x)$ with a large derivative absolute value will change considerably, hence will have high variance. Since the pathwise derivative estimator boils down to $f'(x)$ in the Gaussian case, and the score function estimator boils down to $xf(x)$ (shown in the proof), we wish the expected change in $\frac{\partial}{\partial x}xf(x) = xf'(x) + f(x)$ to be higher than the expected change in $\frac{\partial}{\partial x}f'(x) = f''(x)$, hence the condition.

Proof. We start with the observation that the score function mean estimator w.r.t. a Gaussian $p_\theta(x)$ is given by

$$\frac{\partial}{\partial \mu}\int f(x)p_\theta(x)\mathrm{d}x = \int f(x)\frac{x-\mu}{\sigma^2}p_\theta(x)\mathrm{d}x$$

resulting in the estimator $\widehat{I}_1 = f(x)\frac{x-\mu}{\sigma^2}$. The pathwise derivative and the characteristic function estimators are identical for the mean derivative, and given by $\widehat{I}_2 = f'(x)$. We thus wish to show that $\mathrm{Var}_{p_\theta(x)}(\widehat{I}_2) \leq \mathrm{Var}_{p_\theta(x)}(\widehat{I}_1)$ under the proposition's assumptions. Since $\mathrm{Var}_{p_\theta(x)}(\widehat{I}_1) = \frac{\mathrm{Var}_{p_\theta(x)}(xf(x)-\mu f(x))}{\sigma^4}$, we wish to show that

$$\sigma^4\mathrm{Var}_{p_\theta(x)}\Big(f'(x)\Big) \leq \mathrm{Var}_{p_\theta(x)}\Big((x-\mu)f(x)\Big).$$

Proposition 3.2 in [Cacoullos, 1982] states that for $g(x)$, $g'(x)$ real-valued functions s.t. $g(x)$ is an indefinite integral of $g'(x)$, and $\mathrm{Var}_{p_\theta(x)}(g(x)) < \infty$ and $\mathbb{E}_{p_\theta(x)}(|g'(x)|) < \infty$, there exists that

$$\sigma^2\mathbb{E}_{p_\theta(x)}\Big(g'(x)\Big)^2 \leq \mathrm{Var}_{p_\theta(x)}\Big(g(x)\Big) \leq \sigma^2\mathbb{E}_{p_\theta(x)}\Big(g'(x)^2\Big).$$

Substituting $g(x)$ with $(x-\mu)f(x)$ we have

$$\sigma^2\mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x) + f(x)\Big)^2 \leq \mathrm{Var}_{p_\theta(x)}\Big((x-\mu)f(x)\Big)$$

and substituting $g(x)$ with $f'(x)$ we have

$$\text{Var}_{p_\theta(x)}\Big(f'(x)\Big) \leq \sigma^2 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big).$$

Under the proposition's assumption that

$$\mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x)+f(x)\Big)^2 - \sigma^4 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big) \geq 0$$

we conclude

$$\begin{aligned}\sigma^4 \text{Var}_{p_\theta(x)}\Big(f'(x)\Big) \leq \sigma^6 \mathbb{E}_{p_\theta(x)}\Big(f''(x)^2\Big) &\leq \sigma^2 \mathbb{E}_{p_\theta(x)}\Big((x-\mu)f'(x)+f(x)\Big)^2 \\ &\leq \text{Var}_{p_\theta(x)}\Big((x-\mu)f(x)\Big)\end{aligned}$$

as we wanted to show. □

For example, with the simple polynomial function $f(x) = x + x^2$ and $\mu = 0, \sigma = 1$ from table 3.3 we have

$$\begin{aligned}\mathbb{E}_{\mathcal{N}(0,1)}\Big(xf'(x)+f(x)\Big)^2 - \mathbb{E}_{\mathcal{N}(0,1)}\Big(f''(x)^2\Big) &= \mathbb{E}_{\mathcal{N}(0,1)}\Big(2x+3x^2\Big)^2 - 2^2 \\ &= 3^2 - 2^2 > 0,\end{aligned}$$

and indeed the score function estimator variance is higher than that of the pathwise derivative estimator and that of the characteristic function estimator. A similar result can be derived for the standard deviation derivative estimator.

From empirical observation, the functions $f(x)$ often encountered in VI seem to satisfy the variance relation (1) > (2). For this reason, and since we will make use of distributions other than Gaussian, we continue our work using the pathwise derivative estimator.

3.2 Practical inference in Bayesian neural networks

We now derive what would hopefully be a practical inference method for Bayesian neural networks. Inspired by the work of Graves [2011], we propose an inference technique that satisfies our definition of *practicality*, making use of the tools above. The work in this section and the coming sections was previously presented in [Gal, 2015; Gal and Ghahramani, 2015a,b,c,d, 2016a,b,c].

In his work, Graves [2011] used both delta approximating distributions, as well as fully factorised Gaussian approximating distributions. As such, Graves [2011] relied on Opper and Archambeau [2009]'s characteristic function estimator in his approximation of eq. (3.2). Further, Graves [2011] factorised the approximating distribution for each weight scalar, losing weight correlations. This approach has led to the limitations discussed in §2.2.2, hurting the method's performance and practicality.

Using the tools above, and relying on the pathwise derivative estimator instead of the characteristic function estimator in particular, we can make use of more interesting non-Gaussian approximating distributions. Further, to avoid losing weight correlations, we factorise the distribution for each weight row $\mathbf{w}_{l,i}$ in each weight matrix $\mathbf{W}_l$, instead of factorising over each *weight scalar*. The reason for this will be given below. Using these two key changes, we will see below how our approximate inference can be closely tied to SRTs, suggesting a practical, well performing, implementation.

To use the pathwise derivative estimator we need to re-parametrise each $q_{\theta_{l,i}}(\mathbf{w}_{l,i})$ as $\mathbf{w}_{l,i} = g(\theta_{l,i}, \epsilon_{l,i})$ and specify some $p(\epsilon_{l,i})$ (this will be done at a later time). For simplicity of notation we will write $p(\boldsymbol{\epsilon}) = \prod_{l,i} p(\epsilon_{l,i})$, and $\boldsymbol{\omega} = g(\theta, \boldsymbol{\epsilon})$ collecting all model random variables. Starting from the data sub-sampling objective (eq. (3.2)), we re-parametrise each integral to integrate w.r.t. $p(\boldsymbol{\epsilon})$:

$$\begin{aligned}\widehat{\mathcal{L}}_{\text{VI}}(\theta) &= -\frac{N}{M}\sum_{i\in S}\int q_\theta(\boldsymbol{\omega})\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \\ &= -\frac{N}{M}\sum_{i\in S}\int p(\boldsymbol{\epsilon})\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i))\mathrm{d}\boldsymbol{\epsilon} + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))\end{aligned}$$

and then replace each expected log likelihood term with its stochastic estimator (eq. (3.5)), resulting in a new MC estimator:

$$\widehat{\mathcal{L}}_{\text{MC}}(\theta) = -\frac{N}{M}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i)) + \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \tag{3.7}$$

s.t. $\mathbb{E}_{S,\boldsymbol{\epsilon}}(\widehat{\mathcal{L}}_{\text{MC}}(\theta)) = \mathcal{L}_{\text{VI}}(\theta)$.

Following results in stochastic non-convex optimisation [Rubin, 1981], optimising $\widehat{\mathcal{L}}_{\text{MC}}(\theta)$ w.r.t. θ would converge to the same optima as optimising our original objective $\mathcal{L}_{\text{VI}}(\theta)$. One thus follows algorithm 1 for inference.

Predictions with this approximation follow equation (2.4) which replaces the posterior $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ with the approximate posterior $q_\theta(\boldsymbol{\omega})$. We can then approximate the predictive

Algorithm 1 Minimise divergence between $q_\theta(\boldsymbol{\omega})$ and $p(\boldsymbol{\omega}|X, Y)$

1: Given dataset $\mathbf{X}, \mathbf{Y}$,
2: Define learning rate schedule η,
3: Initialise parameters θ randomly.
4: **repeat**
5: Sample M random variables $\widehat{\boldsymbol{\epsilon}}_i \sim p(\boldsymbol{\epsilon})$, S a random subset of $\{1, .., N\}$ of size M.
6: Calculate stochastic derivative estimator w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{N}{M}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x}_i)) + \frac{\partial}{\partial\theta}\mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})).$$

7: Update θ:
$\theta \leftarrow \theta + \eta\widehat{\Delta\theta}$.
8: **until** θ has converged.

distribution with MC integration as well:

$$\begin{aligned}\tilde{q}_\theta(\mathbf{y}^*|\mathbf{x}^*) := \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \widehat{\boldsymbol{\omega}}_t) &\xrightarrow[T\to\infty]{} \int p\Big(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}\Big)q_\theta(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\ &\approx \int p\Big(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}\Big)p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})\mathrm{d}\boldsymbol{\omega} \\ &= p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})\end{aligned} \tag{3.8}$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta(\boldsymbol{\omega})$.

We next present distributions $q_\theta(\boldsymbol{\omega})$ corresponding to several SRTs, s.t. standard techniques in the deep learning literature could be seen as identical to executing algorithm 1 for approximate inference with $q_\theta(\boldsymbol{\omega})$. This means that existing models that use such SRTs can be interpreted as performing approximate inference. As a result, uncertainty information can be extracted from these, as we will see in the following sections.

3.2.1 Stochastic regularisation techniques

First, what are SRTs? Stochastic regularisation techniques are techniques used to regularise deep learning models through the injection of stochastic noise into the model. By far the most popular technique is *dropout* [Hinton et al., 2012; Srivastava et al., 2014], but other techniques exist such as multiplicative Gaussian noise (MGN, also referred to as *Gaussian dropout*) [Srivastava et al., 2014], or dropConnect [Wan et al., 2013], among many others [Huang et al., 2016; Krueger et al., 2016; Moon et al., 2015; Singh et al.,

2016]. We will concentrate on dropout for the moment, and discuss alternative SRTs below.

Notation remark. In this section and the next, in order to avoid confusion between matrices (used as weights in a NN) and stochastic random matrices (which are random variables inducing a distribution over BNN weights), we change our notation slightly from §1.1. Here we use $\mathbf{M}$ to denote a *deterministic* matrix over the reals, $\mathbf{W}$ to denote a *random variable* defined over the set of real matrices[2], and use $\widehat{\mathbf{W}}$ to denote a realisation of $\mathbf{W}$.

Dropout is a technique used to avoid over-fitting in neural networks. It was suggested as an ad-hoc technique, and was motivated with sexual breeding metaphors rather than through theoretical foundations [Srivastava et al., 2014, page 1932]. It was introduced several years ago by Hinton et al. [2012] and studied more extensively in [Srivastava et al., 2014]. We will describe the use of dropout in simple single hidden layer neural networks (following the notation of §1.1 with the adjustments above). To use dropout we sample two binary vectors $\widehat{\boldsymbol{\epsilon}}_1, \widehat{\boldsymbol{\epsilon}}_2$ of dimensions Q (input dimensionality) and K (intermediate layer dimensionality) respectively. The elements of the vector $\widehat{\boldsymbol{\epsilon}}_i$ take value 0 with probability $0 \leq p_i \leq 1$ for $i = 1, 2$. Given an input $\mathbf{x}$, we set p_1 proportion of the elements of the input to zero in expectation: $\widehat{\mathbf{x}} = \mathbf{x} \odot \widehat{\boldsymbol{\epsilon}}_1$[3]. The output of the first layer is given by $\mathbf{h} = \sigma(\widehat{\mathbf{x}}\mathbf{M}_1 + \mathbf{b})$, in which we randomly set p_2 proportion of the elements to zero: $\widehat{\mathbf{h}} = \mathbf{h} \odot \widehat{\boldsymbol{\epsilon}}_2$, and linearly transform the vector to give the dropout model's output $\widehat{\mathbf{y}} = \widehat{\mathbf{h}}\mathbf{M}_2$. We repeat this for multiple layers.

We sample new realisations for the binary vectors $\widehat{\boldsymbol{\epsilon}}_i$ for every input point and every forward pass through the model (evaluating the model's output), and use the same values in the backward pass (propagating the derivatives to the parameters to be optimised $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$). At test time we do not sample any variables and simply use the original units $\mathbf{x}, \mathbf{h}$ scaled by $\frac{1}{1-p_i}$.

Multiplicative Gaussian noise is similar to dropout, where the only difference is that $\widehat{\boldsymbol{\epsilon}}_i$ are vectors of draws from a Gaussian distribution $\mathcal{N}(1, \alpha)$ with a positive parameter α, rather than draws from a Bernoulli distribution.

[2]The reason for this notation is that $\mathbf{M}$ will often coincide with the mean of the random matrix $\mathbf{W}$.
[3]Here $\odot$ is the element-wise product.

3.2.2 Stochastic regularisation techniques as approximate inference

Dropout and most other SRTs view the injected noise as applied in the feature space (the input features to each layer: $\mathbf{x}, \mathbf{h}$). In Bayesian NNs, on the other hand, the stochasticity comes from our uncertainty over the model parameters. We can transform dropout's noise from the feature space to the parameter space as follows[4]:

$$\begin{aligned}
\widehat{\mathbf{y}} &= \widehat{\mathbf{h}}\mathbf{M}_2 \\
&= (\mathbf{h} \odot \widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2 \\
&= (\mathbf{h} \cdot \text{diag}(\widehat{\boldsymbol{\epsilon}}_2))\mathbf{M}_2 \\
&= \mathbf{h}(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2) \\
&= \sigma\Big(\widehat{\mathbf{x}}\mathbf{M}_1 + \mathbf{b}\Big)(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2) \\
&= \sigma\Big((\mathbf{x} \odot \widehat{\boldsymbol{\epsilon}}_1)\mathbf{M}_1 + \mathbf{b}\Big)(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2) \\
&= \sigma\Big(\mathbf{x}(\text{diag}(\widehat{\boldsymbol{\epsilon}}_1)\mathbf{M}_1) + \mathbf{b}\Big)(\text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2)
\end{aligned}$$

writing $\widehat{\mathbf{W}}_1 := \text{diag}(\widehat{\boldsymbol{\epsilon}}_1)\mathbf{M}_1$ and $\widehat{\mathbf{W}}_2 := \text{diag}(\widehat{\boldsymbol{\epsilon}}_2)\mathbf{M}_2$ we end up with

$$\widehat{\mathbf{y}} = \sigma\Big(\mathbf{x}\widehat{\mathbf{W}}_1 + \mathbf{b}\Big)\widehat{\mathbf{W}}_2 =: \mathbf{f}^{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}}(\mathbf{x})$$

with random variable realisations as weights, and write $\widehat{\boldsymbol{\omega}} = \{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}\}$.

This allows us to write dropout's objective in a more convenient form. Recall that a neural network's optimisation objective is given by eq. (1.3). For dropout it will simply be:

$$\widehat{\mathcal{L}}_{\text{dropout}}(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}) := \frac{1}{M} \sum_{i \in S} E^{\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i, \mathbf{b}}(\mathbf{x}_i, \mathbf{y}_i) + \lambda_1 ||\mathbf{M}_1||^2 + \lambda_2 ||\mathbf{M}_2||^2 + \lambda_3 ||\mathbf{b}||^2, \tag{3.9}$$

with $\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i$ corresponding to new masks $\widehat{\boldsymbol{\epsilon}}_1^i, \widehat{\boldsymbol{\epsilon}}_2^i$ sampled for each data point i. Here we used data sub-sampling with a random index set S of size M, as is common in deep learning.

[4]Here the diag(·) operator maps a vector to a diagonal matrix whose diagonal is the elements of the vector.

As shown by Tishby et al. [1989], in regression $E^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x},\mathbf{y})$ can be rewritten as the negative log-likelihood scaled by a constant:

$$E^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x},\mathbf{y}) = \frac{1}{2}||\mathbf{y} - \mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})||^2 = -\frac{1}{\tau}\log p(\mathbf{y}|\mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})) + \text{const} \tag{3.10}$$

where $p(\mathbf{y}|\mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})) = \mathcal{N}(\mathbf{y}; \mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x}), \tau^{-1}I)$ with τ^{-1} observation noise. It is simple to see that this holds for classification as well (in which case we should set $\tau = 1$).

Recall that $\widehat{\boldsymbol{\omega}} = \{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}\}$ and write

$$\widehat{\boldsymbol{\omega}}_i = \{\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i, \mathbf{b}\} = \{\text{diag}(\widehat{\boldsymbol{\epsilon}}_1^i)\mathbf{M}_1, \text{diag}(\widehat{\boldsymbol{\epsilon}}_2^i)\mathbf{M}_2, \mathbf{b}\} =: g(\theta, \widehat{\boldsymbol{\epsilon}}_i)$$

with $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$, $\widehat{\boldsymbol{\epsilon}}_1^i \sim p(\boldsymbol{\epsilon}_1)$, and $\widehat{\boldsymbol{\epsilon}}_2^i \sim p(\boldsymbol{\epsilon}_2)$ for $1 \leq i \leq N$. Here $p(\boldsymbol{\epsilon}_l)$ $(l = 1, 2)$ is a product of Bernoulli distributions with probabilities $1 - p_l$, from which a realisation would be a vector of zeros and ones.

We can plug identity (3.10) into objective (3.9) and get

$$\widehat{\mathcal{L}}_{\text{dropout}}(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}) = -\frac{1}{M\tau}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2 \tag{3.11}$$

with $\widehat{\boldsymbol{\epsilon}}_i$ realisations of the random variable $\boldsymbol{\epsilon}$.

The derivative of this optimisation objective w.r.t. model parameters $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$ is given by

$$\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{dropout}}(\theta) = -\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \frac{\partial}{\partial\theta}\Big(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\Big).$$

The optimisation of a NN with dropout can then be seen as following algorithm 2.

There is a great sense of similarity between algorithm 1 for approximate inference in a Bayesian NN and algorithm 2 for dropout NN optimisation. Specifically, note the case of a Bayesian NN with approximating distribution $q(\boldsymbol{\omega})$ s.t. $\boldsymbol{\omega} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{M}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{M}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ $(l = 1, 2)$ a product of Bernoulli distributions with probability $1 - p_l$ (which we will refer to as a *Bernoulli variational distribution* or a *dropout variational distribution*). The only differences between algorithm 1 and algorithm 2 are

1. the regularisation term derivatives ($\text{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$ in algo. 1 and $\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2$ in algo. 2),
2. and the scale of $\widehat{\Delta\theta}$ (multiplied by a *constant* $\frac{1}{N\tau}$ in algo. 2).

Algorithm 2 Optimisation of a neural network with dropout

1: Given dataset $\mathbf{X}, \mathbf{Y}$,
2: Define learning rate schedule η,
3: Initialise parameters θ randomly.
4: **repeat**
5: Sample M random variables $\widehat{\boldsymbol{\epsilon}}_i \sim p(\boldsymbol{\epsilon})$, S a random subset of $\{1, .., N\}$ of size M.
6: Calculate derivative w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\widehat{\boldsymbol{\epsilon}}_i)}(\mathbf{x})) + \frac{\partial}{\partial\theta}\Big(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\Big).$$

7: Update θ:
$\theta \leftarrow \theta + \eta\widehat{\Delta\theta}$.
8: **until** θ has converged.

More specifically, if we define the prior $p(\boldsymbol{\omega})$ s.t. the following holds:

$$\frac{\partial}{\partial\theta}\mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) = \frac{\partial}{\partial\theta}N\tau\Big(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2\Big) \tag{3.12}$$

(referred to as the *KL condition*), we would have the following relation between the derivatives of objective (3.11) and objective (3.7):

$$\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{dropout}}(\theta) = \frac{1}{N\tau}\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{MC}}(\theta)$$

with identical optimisation procedures!

We found that for a specific choice for the approximating distribution $q_\theta(\boldsymbol{\omega})$, VI results in identical optimisation procedure to that of a dropout NN. I would stress that this means that optimising *any* neural network with dropout is equivalent to a form of approximate inference in a probabilistic interpretation of the model[5]. This means that the optimal weights found through the optimisation of a dropout NN (using algo. 2) are the same as the optimal variational parameters in a Bayesian NN with the same structure. Further, this means that a network already trained with dropout *is* a Bayesian NN, thus possesses all the properties a Bayesian NN possesses.

We have so far concentrated mostly on the dropout SRT. As to alternative SRTs, remember that an approximating distribution $q_\theta(\boldsymbol{\omega})$ is defined through its re-parametrisation $\boldsymbol{\omega} = g(\theta, \boldsymbol{\epsilon})$. Various SRTs can be recovered for different re-parametrisations. For exam-

[5]Note that to get well-calibrated uncertainty estimates we have to optimise the dropout probability p as well as θ, for example through grid-search over over validation log probability. This is discussed further in §4.3

ple, multiplicative Gaussian noise [Srivastava et al., 2014] can be recovered by setting $g(\theta, \boldsymbol{\epsilon}) = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{M}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{M}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ (for $l = 1, 2$) a product of $\mathcal{N}(1, \alpha)$ with positive-valued α[6]. This can be efficiently implemented by multiplying a network's units by i.i.d. draws from a $\mathcal{N}(1, \alpha)$. On the other hand, setting $g(\theta, \boldsymbol{\epsilon}) = \{\mathbf{M}_1 \odot \boldsymbol{\epsilon}_1, \mathbf{M}_2 \odot \boldsymbol{\epsilon}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ a product of Bernoulli random variables for each weight scalar we recover dropConnect [Wan et al., 2013]. This can be efficiently implemented by multiplying a network's weight scalars by i.i.d. draws from a Bernoulli distribution. It is interesting to note that Graves [2011]'s fully factorised approximation can be recovered by setting $g(\theta, \boldsymbol{\epsilon}) = \{\mathbf{M}_1 + \boldsymbol{\epsilon}_1, \mathbf{M}_2 + \boldsymbol{\epsilon}_2, \mathbf{b}\}$ with $p(\boldsymbol{\epsilon}_l)$ a product of $\mathcal{N}(0, \alpha)$ for each weight scalar. This SRT is often referred to as *additive Gaussian noise*.

3.2.3 KL condition

For VI to result in an identical optimisation procedure to that of a dropout NN, the KL condition (eq. (3.12)) has to be satisfied. Under what constraints does the KL condition hold? This depends on the model specification (selection of prior $p(\boldsymbol{\omega})$) as well as choice of approximating distribution $q_\theta(\boldsymbol{\omega})$. For example, it can be shown that setting the model prior to $p(\boldsymbol{\omega}) = \prod_{i=1}^{L} p(\mathbf{W}_i) = \prod_{i=1}^{L} \mathcal{N}(0, \mathbf{I}/l_i^2)$, in other words independent normal priors over each weight, with *prior length-scale*[7]

$$l_i^2 = \frac{2N\tau\lambda_i}{1 - p_i} \tag{3.13}$$

we have

$$\frac{\partial}{\partial\theta}\text{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \approx \frac{\partial}{\partial\theta} N\tau(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2)$$

for a large enough number of hidden units and a Bernoulli variational distribution. This is discussed further in appendix A. Alternatively, a discrete prior distribution

$$p(\mathbf{w}) \propto e^{-\frac{l^2}{2}\mathbf{w}^T\mathbf{w}}$$

[6] For the multiplicative Gaussian noise this result was also presented in [Kingma et al., 2015], which was done in parallel to this work.

[7] A note on *mean-squared-error* losses: the mean-squared-error loss can be seen as a scaling of the Euclidean loss (eq. (1.1)) by a factor of 2, which implies that the factor of 2 in the length-scale should be removed. The mean-squared-error loss is used in many modern deep learning packages instead (or as) the Euclidean loss.

defined over a *finite* space $\mathbf{w} \in X$ satisfies the KL condition (eq. (3.12)) exactly. This is discussed in more detail in §6.5. For multiplicative Gaussian noise, Kingma et al. [2015] have shown that an improper log-uniform prior distribution satisfies our KL condition.

Notation remark. In the rest of this work we will use $\mathbf{M}$ and $\mathbf{W}$ interchangeably, with the understanding that in deterministic NNs the random variable $\mathbf{W}$ will follow a delta distribution with mean parameter $\mathbf{M}$.

Remark (What is a prior length-scale l_i^2?)**.** It is interesting to explain why we consider the parameter l to be a prior *length-scale* in a Bayesian NN when setting a Gaussian prior $\mathcal{N}(0, I/l^2)$ over the weights. To see this, re-parametrise the input prior distribution as $\mathbf{W}_1' \sim \mathcal{N}(0, I)$, with $\mathbf{W}_1 = \mathbf{W}_1'/l$:

$$\widehat{\mathbf{y}} = \sigma\left(\mathbf{x}\frac{\mathbf{W}_1'}{l} + \mathbf{b}\right)\mathbf{W}_2 = \sigma\left(\frac{\mathbf{x}}{l}\mathbf{W}_1' + \mathbf{b}\right)\mathbf{W}_2$$

i.e. placing a prior distribution $\mathcal{N}(0, I/l^2)$ over $\mathbf{W}_1$ can be replaced by scaling the inputs by $1/l$ with a $\mathcal{N}(0, I)$ prior instead. For example, multiplying the inputs by 100 (making the function smoother) and placing a prior length-scale $l = 100$ would give identical model output to placing a prior length-scale $l = 1$ with the original inputs. This means that the length-scale's unit of measure is identical to the inputs' one.
What does the prior length-scale mean? To see this, consider a real valued function $f(x)$, periodic with period P, and consider its Fourier expansion with K terms:

$$f_K(x) := \frac{A_0}{2} + \sum_{k=1}^{K} A_k \cdot \sin\left(\tfrac{2\pi k}{P}x + \phi_k\right).$$

This can be seen as a single hidden layer neural network with a non-linearity $\sigma(\cdot) := \sin(\cdot)$, input weights given by the Fourier frequencies $\mathbf{W}_1 := [\frac{2\pi k}{P}]_{k=1}^K$ (which are fixed and not learnt), $\mathbf{b} := [\phi_k]_{k=1}^K$ is the bias term of the hidden layer, the Fourier coefficients are the output weights $\mathbf{W}_2 := [A_k]_{k=1}^K$, and $\frac{A_0}{2}$ is the output bias (which can be omitted for centred data). For simplicity we assume that $\mathbf{W}_1$ is composed of only the Fourier frequencies for which the Fourier coefficients are not zero. For example, $\mathbf{W}_1$ might be composed of high frequencies, low frequencies, or a combination of the two.

This view of single hidden layer neural networks gives us some insights into the role of the different quantities used in a neural network. For example, erratic functions have high frequencies, i.e. high magnitude input weights $\mathbf{W}_1$. On the other hand, smooth slow-varying functions are composed of low frequencies, and as a result the magnitude of $\mathbf{W}_1$ is small. The magnitude of $\mathbf{W}_2$ determines how much different frequencies will be used to compose the output function $f_K(x)$. High magnitude $\mathbf{W}_2$ results in a large magnitude for the function's outputs, whereas low $\mathbf{W}_2$ magnitude gives function outputs scaled down and closer to zero.

When we place a prior distribution over the input weights of a BNN, we can capture this characteristic. Having $\mathbf{W}_1 \sim \mathcal{N}(0, I/l^2)$ a priori with long length-scale l results in weights with low magnitude, and as a result slow-varying induced functions. On the other hand, placing a prior distribution with a short length-scale gives high magnitude weights, and as a result erratic functions with high frequencies. This will be demonstrated empirically in §4.1.

Given the intuition about weight magnitude above, equation (3.13) can be re-written to cast some light on the structure of the weight-decay in a neural network[a]:

$$\lambda_i = \frac{l_i^2(1-p_i)}{2N\tau}. \tag{3.14}$$

A short length-scale l_i (corresponding to high frequency data) with high precision τ (equivalently, small observation noise) results in a small weight-decay λ_i—encouraging the model to fit the data well but potentially generalising badly. A long length-scale with low precision results in a large weight-decay—and stronger regularisation over the weights. This trade-off between the length-scale and model precision results in different weight-decay values.

Lastly, I would comment on the choice of placing a distribution over the rows of a weight matrix rather than factorising it over each row's elements. Gal and Turner [2015] offered a derivation related to the Fourier expansion above, where a function drawn from a Gaussian process (GP) was approximated through a finite Fourier decomposition of the GP's covariance function. This derivation has many properties in common with the view above. Interestingly, in the multivariate $\mathbf{f}(\mathbf{x})$ case the Fourier frequencies are given in the columns of the equivalent weight matrix $\mathbf{W}_1$ of size Q (input dimension) by K (number of expansion terms). This generalises on the univariate case above where $\mathbf{W}_1$ is of dimensions $Q = 1$ by K and each entry (column) is a single frequency. Factorising the weight matrix approximating

distribution $q_\theta(\mathbf{W}_1)$ over its *rows* rather than columns captures correlations over the function's frequencies.

[a]Note that with a *mean-squared-error* loss the factor of 2 should be removed.

3.3 Model uncertainty in Bayesian neural networks

We next derive results extending on the above showing that model uncertainty can be obtained from NN models that make use of SRTs such as dropout.

Recall that our approximate predictive distribution is given by eq. (2.4):

$$q_\theta^*(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}^*|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*))q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \tag{3.15}$$

where $\boldsymbol{\omega} = \{\mathbf{W}_i\}_{i=1}^L$ is our set of random variables for a model with L layers, $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)$ is our model's stochastic output, and $q_\theta^*(\boldsymbol{\omega})$ is an optimum of eq. (3.7).

We will perform moment-matching and estimate the first two moments of the predictive distribution empirically. The first moment can be estimated as follows:

Proposition 2. *Given* $p(\mathbf{y}^*|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)) = \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I})$ *for some* $\tau > 0$, $\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*]$ *can be estimated with the unbiased estimator*

$$\widetilde{\mathbb{E}}[\mathbf{y}^*] := \frac{1}{T}\sum_{t=1}^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) \xrightarrow[T\to\infty]{} \mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] \tag{3.16}$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$.

Proof.

$$\begin{aligned}
\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] &= \int \mathbf{y}^* q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)\mathrm{d}\mathbf{y}^* \\
&= \int\int \mathbf{y}^* \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I}) q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\mathrm{d}\mathbf{y}^* \\
&= \int \left(\int \mathbf{y}^* \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I})\mathrm{d}\mathbf{y}^*\right) q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\
&= \int \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*) q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega},
\end{aligned}$$

giving the unbiased estimator $\widetilde{\mathbb{E}}[\mathbf{y}^*] := \frac{1}{T}\sum_{t=1}^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)$ following MC integration with T samples. □

When used with dropout, we refer to this Monte Carlo estimate (3.16) as *MC dropout*. In practice MC dropout is equivalent to performing T stochastic forward passes through the network and averaging the results. For dropout, this result has been presented in the literature before as *model averaging* [Srivastava et al., 2014]. We have given a new derivation for this result which allows us to derive mathematically grounded uncertainty estimates as well, and generalises to all SRTs (including SRTs such as multiplicative Gaussian noise where the model averaging interpretation would result in infinitely many models). Srivastava et al. [2014, section 7.5] have reasoned based on empirical experimentation that the model averaging can be approximated by multiplying each network unit $\mathbf{h}_i$ by $1/(1-p_i)$ at test time, referred to as *standard dropout*. This can be seen as propagating the mean of each layer to the next. Below (in section §4.4) we give results showing that there exist models in which standard dropout gives a bad approximation to the model averaging.

We estimate the second raw moment (for regression) using the following proposition:

Proposition 3.
Given $p(\mathbf{y}^*|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)) = \mathcal{N}(\mathbf{y}^*; \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*), \tau^{-1}\mathbf{I})$ *for some* $\tau > 0$, $\mathbb{E}_{q^*_\theta(\mathbf{y}^*|\mathbf{x}^*)}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right]$ *can be estimated with the unbiased estimator*

$$\widetilde{\mathbb{E}}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) \xrightarrow[T\to\infty]{} \mathbb{E}_{q^*_\theta(\mathbf{y}^*|\mathbf{x}^*)}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right]$$

with $\widehat{\boldsymbol{\omega}}_t \sim q^*_\theta(\boldsymbol{\omega})$ *and* $\mathbf{y}^*, \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)$ *row vectors (thus the sum is over the outer-products).*

Proof.

$$\begin{aligned}
\mathbb{E}_{q^*_\theta(\mathbf{y}^*|\mathbf{x}^*)}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right] &= \int \left(\int (\mathbf{y}^*)^T(\mathbf{y}^*) p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) \mathrm{d}\mathbf{y}^* \right) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\
&= \int \left(\mathrm{Cov}_{p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})}[\mathbf{y}^*] + \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})}[\mathbf{y}^*]^T \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega})}[\mathbf{y}^*] \right) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\
&= \int \left(\tau^{-1}\mathbf{I} + \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*)^T \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}^*) \right) q^*_\theta(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}
\end{aligned}$$

giving the unbiased estimator $\widetilde{\mathbb{E}}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)$ following MC integration with T samples. □

To obtain the model's predictive variance we use the unbiased estimator:

$$\widetilde{\mathrm{Var}}[\mathbf{y}^*] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) - \widetilde{\mathbb{E}}[\mathbf{y}^*]^T \widetilde{\mathbb{E}}[\mathbf{y}^*]$$

$$\xrightarrow[T\to\infty]{} \text{Var}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*]$$

which equals the sample variance of T stochastic forward passes through the NN plus the inverse model precision.

How can we find the model precision? In practice in the deep learning literature we often grid-search over the weight-decay λ to minimise validation error. Then, given a weight-decay λ_i (and prior length-scale l_i), eq. (3.13) can be re-written to find the model precision[8]:

$$\tau = \frac{(1-p)l_i^2}{2N\lambda_i}. \tag{3.17}$$

Remark (Predictive variance and posterior variance)**.** It is important to note the difference between the variance of the approximating distribution $q_\theta(\boldsymbol{\omega})$ and the variance of the predictive distribution $q_\theta(\mathbf{y}|\mathbf{x})$ (eq. (3.15)).
To see this, consider the illustrative example of an approximating distribution with fixed mean and variance used with the first weight layer $\mathbf{W}_1$, for example a standard Gaussian $\mathcal{N}(0, I)$. Further, assume for the sake of argument that delta distributions (or Gaussians with very small variances) are used to approximate the posterior over the layers following the first layer. Given enough follow-up layers we can capture any function to arbitrary precision—including the inverse cumulative distribution function (CDF) of any distribution (similarly to the remark in §2.2.1, but with the addition of a Gaussian CDF as we don't have a uniform on the first layer in this case). Passing the distribution from the first layer through the rest of the layers transforms the standard Gaussian with this inverse CDF, resulting in any arbitrary distribution as determined by the CDF.
In this example, even though the variance of each weight layer is constant, the variance of the predictive distribution can take any value depending on the learnt CDF. This example can be extended from a standard Gaussian approximating distribution to a mixture of Gaussians with fixed standard deviations, and to discrete distributions with fixed probability vectors (such as the dropout approximating distribution). Of course, in real world cases we would prefer to avoid modelling the

[8]Prior length-scale l_i can be fixed based on the density of the input data $\mathbf{X}$ and our prior belief as to the function's wiggliness, or optimised over as well (w.r.t. predictive log-likelihood over a validation set). The dropout probability is optimised using grid search similarly.

deep layers with delta approximating distributions since that would sacrifice our ability to capture model uncertainty.

Given a dataset $\mathbf{X}, \mathbf{Y}$ and a new data point $\mathbf{x}^*$ we can calculate the probability of possible output values $\mathbf{y}^*$ using the predictive probability $p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$. The log of the predictive likelihood captures how well the model fits the data, with larger values indicating better model fit. Our predictive log-likelihood (also referred to as *test log-likelihood*) can be approximated by MC integration of eq. (3.15) with T terms:

$$\begin{aligned}\widetilde{\log} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) := \log\left(\frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}_t)\right) &\xrightarrow[T\to\infty]{} \log \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) q_\theta^*(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &\approx \log \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega} \\ &= \log p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})\end{aligned}$$

with $\boldsymbol{\omega}_t \sim q_\theta^*(\boldsymbol{\omega})$ and since $q_\theta^*(\boldsymbol{\omega})$ is the minimiser of eq. (2.3). Note that this is a biased estimator since the expected quantity is transformed with the non-linear logarithm function, but the bias decreases as T increases.

For regression we can rewrite this last equation in a more numerically stable way[9]:

$$\widetilde{\log} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \mathrm{logsumexp}\left(-\frac{1}{2}\tau||\mathbf{y} - \mathbf{f}^{\widehat{\omega}_t}(\mathbf{x}^*)||^2\right) - \log T - \frac{1}{2}\log 2\pi + \frac{1}{2}\log\tau \tag{3.18}$$

with our precision parameter τ.

Uncertainty quality can be determined from this quantity as well. Excessive uncertainty (large observation noise, or equivalently small model precision τ) results in a large penalty from the last term in the predictive log-likelihood. On the other hand, an over-confident model with large model precision compared to poor mean estimation results in a penalty from the first term—the distance $||\mathbf{y} - \mathbf{f}^{\widehat{\omega}_t}(\mathbf{x}^*)||^2$ gets amplified by τ which drives the exponent to zero.

Note that the normal NN model itself is not changed. To estimate the predictive mean and predictive uncertainty we simply collect the results of stochastic forward passes through the model. As a result, this information can be used with existing NN models trained with SRTs. Furthermore, the forward passes can be done concurrently, resulting in constant running time identical to that of standard NNs.

[9] logsumexp is the log-sum-exp function.

3.3.1 Uncertainty in classification

In regression we summarised predictive uncertainty by looking at the sample variance of multiple stochastic forward passes. In the classification setting, however, we need to rely on alternative approaches to summarise uncertainty[10]. We will analyse three approaches to summarise uncertainty within classification: variation ratios [Freeman, 1965], predictive entropy [Shannon, 1948], and mutual information [Shannon, 1948]. These measures capture different notions of uncertainty: model uncertainty and predictive uncertainty, and will be explained below. But first, how do we calculate these quantities in our setting?

To use variation ratios we would sample a label from the softmax probabilities at the end of each *stochastic forward pass* for a test input $\mathbf{x}$. Collecting a set of T labels y_t from multiple stochastic forward passes on the same input we can find the mode of the distribution[11] $c^* = \arg\max_{c=1,\dots,C} \sum_t \mathbb{1}[y^t = c]$, and the number of times it was sampled $f_{\mathbf{x}} = \sum_t \mathbb{1}[y^t = c^*]$. We then set

$$\text{variation-ratio}[\mathbf{x}] := 1 - \frac{f_{\mathbf{x}}}{T}. \tag{3.19}$$

The variation ratio is a measure of dispersion—how "spread" the distribution is around the mode. In the binary case, the variation ratio attains its maximum of 0.5 when the two classes are sampled equally likely, and its minimum of 0 when only a single class is sampled.

Remark (Variation ratios and approximate inference)**.** The variation ratio as it was formulated in [Freeman, 1965] and used above can be seen as approximating the quantity

$$1 - p(y = c^*|\mathbf{x}, \mathcal{D}_{\text{train}})$$

with $c^* = \arg\max_{c=1,\dots,C} p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$. This is because for y^t the t'th class sampled for input $\mathbf{x}$ we have:

$$\frac{f_{\mathbf{x}}}{T} = \frac{1}{T}\sum_t \mathbb{1}[y^t = c^*] \xrightarrow[T\to\infty]{} \mathbb{E}_{q_\theta^*(y|\mathbf{x})}\Big[\mathbb{1}[y = c^*]\Big]$$

[10] These approaches are necessary since the probability vector resulting from a *deterministic* forward pass through the model does not capture confidence, as explained in figure 1.3.

[11] Here $\mathbb{1}[\cdot]$ is the indicator function.

$$
\begin{aligned}
&= q_\theta^*(y = c^*|\mathbf{x}) \\
&\approx p(y = c^*|\mathbf{x}, \mathcal{D}_{\text{train}})
\end{aligned}
$$

and,

$$
\begin{aligned}
c^* = \underset{c=1,\dots,C}{\arg\max} \sum_t \mathbb{1}[y^t = c] &= \underset{c=1,\dots,C}{\arg\max} \frac{1}{T} \sum_t \mathbb{1}[y^t = c] \\
&\xrightarrow[T\to\infty]{} \underset{c=1,\dots,C}{\arg\max} \mathbb{E}_{q_\theta^*(y|\mathbf{x})}[\mathbb{1}[y = c]] \\
&= \underset{c=1,\dots,C}{\arg\max} q_\theta^*(y = c|\mathbf{x}). \\
&\approx \underset{c=1,\dots,C}{\arg\max} p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})
\end{aligned}
$$

since $q_\theta^*(\boldsymbol{\omega})$ is a minimiser of the KL divergence to $p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})$ and therefore $q_\theta^*(y|\mathbf{x}) \approx \int p(y|\mathbf{f}^\omega(\mathbf{x}))p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})\mathrm{d}\boldsymbol{\omega}$ (following eq. (3.15)).

Unlike variation ratios, predictive entropy has its foundations in information theory. This quantity captures the average amount of information contained in the predictive distribution:

$$
\mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{train}}] := -\sum_c p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \tag{3.20}
$$

summing over all possible classes c that y can take. Given a test point $\mathbf{x}$, the predictive entropy attains its maximum value when all classes are predicted to have equal uniform probability, and its minimum value of zero when one class has probability 1 and all others probability 0 (i.e. the prediction is certain).

In our setting, the predictive entropy can be approximated by collecting the probability vectors from T stochastic forward passes through the network, and for each class c averaging the probabilities of the class from each of the T probability vectors, replacing $p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$ in eq. (3.20). In other words, we replace $p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$ with $\frac{1}{T}\sum_t p(y = c|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t)$, where $p(y = c|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t)$ is the probability of input $\mathbf{x}$ to take class c with model parameters $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$:

$$
[p(y = 1|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t), ..., p(y = C|\mathbf{x}, \widehat{\boldsymbol{\omega}}_t)] := \text{Softmax}(\mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x})).
$$

Then,

$$\begin{aligned}
\tilde{\mathbb{H}}[y|\mathbf{x},\mathcal{D}_{\text{train}}] &:= -\sum_c \left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\log\left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\\
&\xrightarrow[T\to\infty]{} -\sum_c \left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\right)\log\left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\right)\\
&\approx -\sum_c \left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})\mathrm{d}\boldsymbol{\omega}\right)\log\left(\int p(y=c|\mathbf{x},\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})\mathrm{d}\boldsymbol{\omega}\right)\\
&= -\sum_c p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\log p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\\
&=\mathbb{H}[y|\mathbf{x},\mathcal{D}_{\text{train}}]
\end{aligned}$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$ and since $q_\theta^*(\boldsymbol{\omega})$ is the optimum of eq. (3.7). Note that this is a biased estimator since the unbiased estimator $\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t) \xrightarrow[T\to\infty]{} \int p(y=c|\mathbf{x},\boldsymbol{\omega})q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}$ is transformed through the non-linear function $\mathbb{H}[\cdot]$. The bias of this estimator will decrease as T increases.

As an alternative to the predictive entropy, the mutual information between the prediction y and the posterior over the model parameters $\boldsymbol{\omega}$ offers a different measure of uncertainty:

$$\begin{aligned}
\mathbb{I}[y,\boldsymbol{\omega}|\mathbf{x},\mathcal{D}_{\text{train}}] &:= \mathbb{H}[y|\mathbf{x},\mathcal{D}_{\text{train}}] - \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})}\Big[\mathbb{H}[y|\mathbf{x},\boldsymbol{\omega}]\Big]\\
&= -\sum_c p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\log p(y=c|\mathbf{x},\mathcal{D}_{\text{train}})\\
&\quad + \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})}\Big[\sum_c p(y=c|\mathbf{x},\boldsymbol{\omega})\log p(y=c|\mathbf{x},\boldsymbol{\omega})\Big]
\end{aligned}$$

with c the possible classes y can take. This tractable view of the mutual information was suggested in [Houlsby et al., 2011] in the context of active learning. Test points $\mathbf{x}$ that maximise the mutual information are points on which the model is uncertain on average, yet there exist model parameters that erroneously produce predictions with high confidence.

The mutual information can be approximated in our setting in a similar way to the predictive entropy approximation:

$$\begin{aligned}
\tilde{\mathbb{I}}[y,\boldsymbol{\omega}|\mathbf{x},\mathcal{D}_{\text{train}}] &:= -\sum_c \left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\log\left(\frac{1}{T}\sum_t p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\right)\\
&\quad + \frac{1}{T}\sum_{c,t} p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\log p(y=c|\mathbf{x},\widehat{\boldsymbol{\omega}}_t)\\
&\xrightarrow[T\to\infty]{} \mathbb{H}[y|\mathbf{x},\mathcal{D}_{\text{train}}] - \mathbb{E}_{q_\theta^*(\boldsymbol{\omega})}\Big[\mathbb{H}[y|\mathbf{x},\boldsymbol{\omega}]\Big]
\end{aligned}$$

$$\approx \mathbb{I}[y, \boldsymbol{\omega} | \mathbf{x}, \mathcal{D}_{\text{train}}]$$

with $\widehat{\boldsymbol{\omega}}_t \sim q_\theta^*(\boldsymbol{\omega})$.

Some intuition. To understand the different measures for uncertainty, we shall look at three concrete examples in binary classification of dogs and cats given an input image. More specifically, we will look at the sets of probability vectors obtained from multiple stochastic forward passes, and the uncertainty measures resulting from these sets. The three examples are where the probabilities for the class "dog" in all vectors are

1. all equal to 1 (i.e. the probability vectors collected are $\{(1,0), ..., (1,0)\}$),
2. all equal to 0.5 (i.e. the probability vectors collected are $\{(0.5,0.5), ..., (0.5,0.5)\}$), and
3. half of the probabilities sampled equal to 0 and half of the probabilities equal to 1 (i.e. the probability vectors collected are $\{(1,0),(0,1),(0,1), ..., (1,0)\}$ for example).

In example (1) the prediction has high confidence, whereas in examples (2) and (3) the prediction has low confidence. These are examples of *predictive* uncertainty. Compared to this notion of confidence, in examples (1) and (2) the *model* is confident about its output since it gives identical probabilities in multiple forward passes. On the other hand, in the last example (3) the model is uncertain about its output, corresponding to the case in figure 1.3 (where the layer before the softmax has high uncertainty). This is an example of *model uncertainty*.

In example (1), both variation ratios, predictive entropy, and the mutual information would return value 0, all measures indicating high *confidence*. In example (3) variation ratios, predictive entropy, and the mutual information would all return value 0.5, all measures indicating high *uncertainty*. All three measures of uncertainty agree on these two examples.

However, in example (2) variation ratios and predictive entropy would return value 0.5, whereas the mutual information would return value 0. In this case variation ratios and predictive entropy capture the *uncertainty in the prediction*, whereas the mutual information captures the *model's confidence* in its output. This information can be used for example in *active learning*, and will be demonstrated in section §5.2.

3.3.2 Difficulties with the approach

Our technique is simple: perform several stochastic forward passes through the model, and look at the sample mean and variance. But it has several shortcomings worth discussing.

First, even though the training time of our model is identical to that of existing models in the field, the test time is scaled by T—the number of averaged forward passes through the network. This may not be of real concern in some real world applications, as NNs are often implemented on distributed hardware. Distributed hardware allows us to obtain MC estimates in constant time almost trivially, by transferring an input to a GPU and setting a mini-batch composed of the same input multiple times. In dropout for example we sample different Bernoulli realisations for each output unit and each mini-batch input, which results in a matrix of probabilities. Each row in the matrix is the output of the dropout network on the same input generated with different random variable realisations (dropout masks). Averaging over the rows results in the MC dropout estimate.

Another concern is that the model's uncertainty is not calibrated. A calibrated model is one in which the predictive probabilities match the empirical frequency of the data. The lack of calibration can be seen through the derivation's relation to Gaussian processes [Gal and Ghahramani, 2015b]. Gaussian processes' uncertainty is known to not be calibrated—the Gaussian process's uncertainty depends on the covariance function chosen, which is shown in [Gal and Ghahramani, 2015b] to be equivalent to the non-linearities and prior over the weights. The choice of a GP's covariance function follows from our assumptions about the data. If we believe, for example, that the model's uncertainty should increase far from the data we might choose the squared exponential covariance function.

For many practical applications the lack of calibration means that model uncertainty can increase for large magnitude data points or be of different scale for different datasets. To calibrate model uncertainty in regression tasks we can scale the uncertainty linearly to remove data magnitude effects, and manipulate uncertainty percentiles to compare among different datasets. This can be done by finding the number of validation set points having larger uncertainty than that of a test point. For example, if a test point has predictive standard deviation 5, whereas almost all validation points have standard deviation ranging from 0.2 to 2, then the test point's uncertainty value will be in the top percentile of the validation set uncertainty measures, and the model will be considered as very uncertain about the test point compared to the validation data. However, another model might give the same test point predictive standard deviation of 5 with most of the validation data given predictive standard deviation ranging from 10 to 15. In this model the test point's uncertainty measure will be in the lowest percentile of validation set uncertainty measures, and the model will be considered as fairly confident about the test point with respect to the validation data.

One last concern is a known limitation of VI. Variational inference is known to underestimates predictive variance [Turner and Sahani, 2011], a property descendent from our objective (which penalises $q_\theta(\boldsymbol{\omega})$ for placing mass where $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ has no mass, but less so for not placing mass where it should). Several solutions exist for this (such as [Giordano et al., 2015]), with different limitations for their practicality. Uncertainty under-estimation does not seem to be of real concern in practice though, and the variational approximation seems to work well in practical applications as we will see in the next chapter.

3.4 Approximate inference in complex models

We finish the chapter by extending the approximate inference technique above to more complex models such as convolutional neural networks and recurrent neural networks. This will allow us to obtain model uncertainty for models defined over sequence based datasets or for image data. We describe the approximate inference using Bernoulli variational distributions for convenience of presentation, although any SRT could be used instead.

3.4.1 Bayesian convolutional neural networks

In existing convolutional neural networks (CNNs) literature dropout is mostly used after inner-product layers at the end of the model alone. This can be seen as applying a finite deterministic transformation to the data before feeding it into a Bayesian NN. As such, model uncertainty can still be obtained for such models, but an interesting question is whether we could use approximate Bayesian inference over the full CNN. Here we wish to integrate over the convolution layers (kernels) of the CNN as well. To implement a Bayesian CNN we could apply dropout after all convolution layers as well as inner-product layers, and evaluate the model's predictive posterior using eq. (3.8) at test time. Note though that generalisations to SRTs other than dropout are also possible and easy to implement.

Recall the structure of a CNN described in §1.1 and in figure 1.1. In more detail, the input to the i'th convolution layer is represented as a 3 dimensional tensor $\mathbf{x} \in \mathbb{R}^{H_{i-1} \times W_{i-1} \times K_{i-1}}$ with height H_{i-1}, width W_{i-1}, and K_{i-1} channels. A convolution layer is then composed of a sequence of K_i kernels (weight tensors): $\mathbf{k}_k \in \mathbb{R}^{h \times w \times K_{i-1}}$ for $k = 1, ..., K_i$. Here we assume kernel height h, kernel width w, and the last dimension to match the number of channels in the input layer: K_{i-1}. Convolving

the kernels with the input (with a given stride s) then results in an output layer of dimensions $\mathbf{y} \in \mathbb{R}^{H'_{i-1} \times W'_{i-1} \times K_i}$ with H'_{i-1} and W'_{i-1} being the new height and width, and K_i channels—the number of kernels. Each element $y_{i,j,k}$ is the sum of the element-wise product of kernel $\mathbf{k}_k$ with a corresponding patch in the input image $\mathbf{x}$: $[[x_{i-h/2,j-w/2,1}, ..., x_{i+h/2,j+w/2,1}], ..., [x_{i-h/2,j-w/2,K_{i-1}}, ..., x_{i+h/2,j+w/2,K_{i-1}}]]$.

To integrate over the kernels, we reformulate the convolution as a linear operation. Let $\mathbf{k}_k \in \mathbb{R}^{h \times w \times K_{i-1}}$ for $k = 1, ..., K_i$ be the CNN's kernels with height h, width w, and K_{i-1} channels in the i'th layer. The input to the layer is represented as a 3 dimensional tensor $\mathbf{x} \in \mathbb{R}^{H_{i-1} \times W_{i-1} \times K_{i-1}}$ with height H_{i-1}, width W_{i-1}, and K_{i-1} channels. Convolving the kernels with the input with a given stride s is equivalent to extracting patches from the input and performing a matrix product: we extract $h \times w \times K_{i-1}$ dimensional patches from the input with stride s and vectorise these. Collecting the vectors in the rows of a matrix we obtain a new representation for our input $\overline{\mathbf{x}} \in \mathbb{R}^{n \times hwK_{i-1}}$ with n patches. The vectorised kernels form the columns of the weight matrix $\mathbf{W}_i \in \mathbb{R}^{hwK_{i-1} \times K_i}$. The convolution operation is then equivalent to the matrix product $\overline{\mathbf{x}}\mathbf{W}_i \in \mathbb{R}^{n \times K_i}$. The columns of the output can be re-arranged into a 3 dimensional tensor $\mathbf{y} \in \mathbb{R}^{H_i \times W_i \times K_i}$ (since $n = H_i \times W_i$). Pooling can then be seen as a non-linear operation on the matrix $\mathbf{y}$. Note that the pooling operation is a non-linearity applied after the linear convolution counterpart to ReLU or Tanh non-linearities.

To make the CNN into a probabilistic model we place a prior distribution over each kernel and approximately integrate each kernels-patch pair with Bernoulli variational distributions. We sample Bernoulli random variables $\boldsymbol{\epsilon}_{i,j,n}$ and multiply patch n by the weight matrix $\mathbf{W}_i \cdot \text{diag}([\boldsymbol{\epsilon}_{i,j,n}]_{j=1}^{K_i})$. This product is equivalent to an approximating distribution modelling each kernel-patch pair with a distinct random variable, tying the means of the random variables over the patches. The distribution randomly sets kernels to zero for different patches. This approximating distribution is also equivalent to applying dropout for each element in the tensor $\mathbf{y}$ before pooling. Implementing our Bayesian CNN is therefore as simple as using dropout after every convolution layer before pooling.

The standard dropout test time approximation (scaling hidden units by $1/(1 - p_i)$) does not perform well when dropout is applied after convolutions—this is a negative result we identified empirically. We solve this by approximating the predictive distribution following eq. (3.8), averaging stochastic forward passes through the model at test time (using MC dropout). We assess the model above with an extensive set of experiments studying its properties in §4.4.

3.4.2 Bayesian recurrent neural networks

We next develop inference with Bernoulli variational distributions for recurrent neural networks (RNNs), although generalisations to SRTs other than dropout are trivial. We will concentrate on simple RNN models for brevity of notation. Derivations for LSTM and GRU follow similarly. Given input sequence $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ of length T, a simple RNN is formed by a repeated application of a deterministic function $\mathbf{f_h}$. This generates a hidden state $\mathbf{h}_t$ for time step t:

$$\mathbf{h}_t = \mathbf{f_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) = \sigma(\mathbf{x}_t \mathbf{W_h} + \mathbf{h}_{t-1} \mathbf{U_h} + \mathbf{b_h})$$

for some non-linearity σ. The model output can be defined, for example, as

$$\mathbf{f_y}(\mathbf{h}_T) = \mathbf{h}_T \mathbf{W_y} + \mathbf{b_y}.$$

To view this RNN as a probabilistic model we regard $\boldsymbol{\omega} = \{\mathbf{W_h}, \mathbf{U_h}, \mathbf{b_h}, \mathbf{W_y}, \mathbf{b_y}\}$ as random variables (following Gaussian prior distributions). To make the dependence on $\boldsymbol{\omega}$ clear, we write $\mathbf{f}_\mathbf{y}^\boldsymbol{\omega}$ for $\mathbf{f_y}$ and similarly for $\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}$. We define our probabilistic model's likelihood as above (section 2.1). The posterior over random variables $\boldsymbol{\omega}$ is rather complex, and we approximate it with a variational distribution $q(\boldsymbol{\omega})$. For the dropout SRT for example we may use a Bernoulli approximating distribution.

Recall our VI optimisation objective eq. (3.1). Evaluating each log likelihood term in eq. (3.1) with our RNN model we have

$$\begin{aligned}\int q(\boldsymbol{\omega}) \log p(\mathbf{y}|\mathbf{f}_\mathbf{y}^\boldsymbol{\omega}(\mathbf{h}_T)) \mathrm{d}\boldsymbol{\omega} &= \int q(\boldsymbol{\omega}) \log p\left(\mathbf{y} \middle| \mathbf{f}_\mathbf{y}^\boldsymbol{\omega}\left(\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(\mathbf{x}_T, \mathbf{h}_{T-1})\right)\right) \mathrm{d}\boldsymbol{\omega} \\ &= \int q(\boldsymbol{\omega}) \log p\left(\mathbf{y} \middle| \mathbf{f}_\mathbf{y}^\boldsymbol{\omega}\left(\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(\mathbf{x}_T, \mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(...\mathbf{f}_\mathbf{h}^\boldsymbol{\omega}(\mathbf{x}_1, \mathbf{h}_0)...))\right)\right) \mathrm{d}\boldsymbol{\omega}\end{aligned}$$

with $\mathbf{h}_0 = \mathbf{0}$. We approximate this with MC integration with a single sample:

$$\approx \log p\left(\mathbf{y} \middle| \mathbf{f}_\mathbf{y}^{\widehat{\boldsymbol{\omega}}}\left(\mathbf{f}_\mathbf{h}^{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_T, \mathbf{f}_\mathbf{h}^{\widehat{\boldsymbol{\omega}}}(...\mathbf{f}_\mathbf{h}^{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_1, \mathbf{h}_0)...))\right)\right),$$

with $\widehat{\boldsymbol{\omega}} \sim q(\boldsymbol{\omega})$, resulting in an unbiased estimator to each sum term[12].

[12]Note that for brevity we did not re-parametrise the integral, although this should be done to obtain low variance derivative estimators.

This estimator is plugged into equation (3.1) to obtain our minimisation objective

$$\widehat{\mathcal{L}}_{\text{MC}} = -\sum_{i=1}^{N} \log p\left(\mathbf{y}_i \middle| \mathbf{f}_{\mathbf{y}}^{\widehat{\boldsymbol{\omega}}_i}\left(\mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}(\mathbf{x}_{i,T}, \mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}(...\mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}(\mathbf{x}_{i,1}, \mathbf{h}_0)...))\right)\right) + \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega})). \quad (3.21)$$

Note that for each sequence $\mathbf{x}_i$ we sample a new realisation $\widehat{\boldsymbol{\omega}}_i = \{\widehat{\mathbf{W}}_{\mathbf{h}}^i, \widehat{\mathbf{U}}_{\mathbf{h}}^i, \widehat{\mathbf{b}}_{\mathbf{h}}^i, \widehat{\mathbf{W}}_{\mathbf{y}}^i, \widehat{\mathbf{b}}_{\mathbf{y}}^i\}$, and that each symbol in the sequence $\mathbf{x}_i = [\mathbf{x}_{i,1}, ..., \mathbf{x}_{i,T}]$ is passed through the function $\mathbf{f}_{\mathbf{h}}^{\widehat{\boldsymbol{\omega}}_i}$ with *the same weight realisations* $\widehat{\mathbf{W}}_{\mathbf{h}}^i, \widehat{\mathbf{U}}_{\mathbf{h}}^i, \widehat{\mathbf{b}}_{\mathbf{h}}^i$ *used at every time step* $t \leq T$.

In the dropout case, evaluating the model output $\mathbf{f}_{\mathbf{y}}^{\widehat{\boldsymbol{\omega}}}(\cdot)$ with sample $\widehat{\boldsymbol{\omega}}$ corresponds to randomly zeroing (masking) rows in each weight matrix $\mathbf{W}$ during the forward pass. In our RNN setting with a sequence input, each weight matrix row is randomly masked once, and importantly the same mask is used through all time steps. When viewed as a stochastic regularisation technique, our induced dropout variant is therefore identical to implementing dropout in RNNs with the *same network units dropped at each time step*, randomly dropping inputs, outputs, and recurrent connections. Predictions can be approximated by either propagating the mean of each layer to the next (referred to as the *standard dropout approximation*), or by performing dropout at test time and averaging results (MC dropout, eq. (3.16)).

Remark (Bayesian versus ensembling interpretation of dropout)**.** Apart from our Bayesian approximation interpretation, dropout in deep networks can also be seen as following an ensembling interpretation [Srivastava et al., 2014]. This interpretation also leads to MC dropout at test time. But the ensembling interpretation does not determine whether the ensemble should be over the network units or the weights. For example, in an RNN this view will *not* lead to our dropout variant, unless the ensemble is *defined to tie the weights of the network* ad hoc. This is in comparison to the Bayesian approximation view where the weight tying is forced by the probabilistic interpretation of the model.
The same can be said about the latent variable model view of dropout [Maeda, 2014] where a constraint over the weights would have to be added ad hoc to derive the results presented here.

Certain RNN models such as LSTMs [Graves et al., 2013; Hochreiter and Schmidhuber, 1997] and GRUs [Cho et al., 2014] use different *gates* within the RNN units. For example, an LSTM is defined by setting four gates: "input", "forget", "output", and an "input modulation gate",

$$\underline{\mathbf{i}} = \text{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_i + \mathbf{x}_t\mathbf{W}_i\right) \qquad \underline{\mathbf{f}} = \text{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_f + \mathbf{x}_t\mathbf{W}_f\right)$$

$$\underline{\mathbf{o}} = \text{sigm}\Big(\mathbf{h}_{t-1}\mathbf{U}_o + \mathbf{x}_t\mathbf{W}_o\Big) \qquad \underline{\mathbf{g}} = \tanh\Big(\mathbf{h}_{t-1}\mathbf{U}_g + \mathbf{x}_t\mathbf{W}_g\Big)$$
$$\mathbf{c}_t = \underline{\mathbf{f}} \odot \mathbf{c}_{t-1} + \underline{\mathbf{i}} \odot \underline{\mathbf{g}} \qquad \mathbf{h}_t = \underline{\mathbf{o}} \odot \tanh(\mathbf{c}_t) \tag{3.22}$$

with $\boldsymbol{\omega} = \{\mathbf{W}_i, \mathbf{U}_i, \mathbf{W}_f, \mathbf{U}_f, \mathbf{W}_o, \mathbf{U}_o, \mathbf{W}_g, \mathbf{U}_g\}$ weight matrices and sigm the sigmoid non-linearity. Here an internal state $\mathbf{c}_t$ (also referred to as *cell*) is updated additively.

Alternatively, the model could be re-parametrised as in [Graves et al., 2013]:

$$\begin{pmatrix} \underline{\mathbf{i}} \\ \underline{\mathbf{f}} \\ \underline{\mathbf{o}} \\ \underline{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \left(\mathbf{W} \cdot \begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix} \right) \tag{3.23}$$

with $\boldsymbol{\omega} = \{\mathbf{W}\}$, $\mathbf{W} = [\mathbf{W}_i, \mathbf{U}_i; \mathbf{W}_f, \mathbf{U}_f; \mathbf{W}_o, \mathbf{U}_o; \mathbf{W}_g, \mathbf{U}_g]$ a matrix of dimensions $4K$ by $2K$ (K being the dimensionality of $\mathbf{x}_t$). We name this parametrisation a *tied-weights* LSTM (compared to the *untied-weights* LSTM parametrisation in eq. (3.22)).

Even though these two parametrisations result in the same deterministic model output, they lead to different approximating distributions $q(\boldsymbol{\omega})$. With the first parametrisation one could use different dropout masks for different gates (even when the same input $\mathbf{x}_t$ is used). This is because the approximating distribution is placed over the matrices rather than the inputs: we might drop certain rows in one weight matrix $\mathbf{W}$ applied to $\mathbf{x}_t$ and different rows in another matrix $\mathbf{W}'$ applied to $\mathbf{x}_t$. With the second parametrisations we would place a distribution over the single matrix $\mathbf{W}$. This leads to a faster forward-pass, but with slightly diminished results (this tradeoff is examined in section §4.5).

Remark (Word embeddings dropout)**.** In datasets with continuous inputs we often apply SRTs such as dropout to the input layer—i.e. to the input vector itself. This is equivalent to placing a distribution over the weight matrix which follows the input and approximately integrating over it (the matrix is optimised, therefore prone to overfitting otherwise).
But for models with discrete inputs such as words (where every word is mapped to a continuous vector—a *word embedding*)—this is seldom done. With word embeddings the input can be seen as either the word embedding itself, or, more conveniently, as a "one-hot" encoding (a vector of zeros with 1 at a single position). The product of the one-hot encoded vector with an embedding matrix $\mathbf{W}_E \in \mathbb{R}^{V \times D}$ (where D is the embedding dimensionality and V is the number of words in the vocabulary) then gives a word embedding. Curiously, this parameter layer is the largest layer in

most language applications, yet it is often not regularised. Since the embedding matrix is optimised it can lead to overfitting, and it is therefore desirable to apply dropout to the one-hot encoded vectors. This in effect is identical to *dropping words at random* throughout the input sentence, and can also be interpreted as encouraging the model to not "depend" on single words for its output.
Note that as before, we randomly set rows of the matrix $\mathbf{W}_E \in \mathbb{R}^{V \times D}$ to zero. Since we repeat the same mask at each time step, we drop the same words throughout the sequence—i.e. we drop word types at random rather than word tokens (as an example, the sentence "the dog and the cat" might become "— dog and — cat" or "the — and the cat", but never "— dog and the cat"). A possible inefficiency implementing this is the requirement to sample V Bernoulli random variables, where V might be large. This can be solved by the observation that for sequences of length T, at most T embeddings could be dropped (other dropped embeddings have no effect on the model output). For $T \ll V$ it is therefore more efficient to first map the words to the word embeddings, and only then to zero-out word embeddings based on their word type.

In the next chapter we will study the techniques above empirically and analyse them quantitatively. This is followed by a survey of recent literature making use of the techniques in real-world problems concerning AI safety, image processing, sequence processing, active learning, and other examples.

Chapter 4

Uncertainty Quality

In this chapter we assess the techniques developed in the previous chapters, concentrating on questions such as what our model uncertainty looks like. We experiment with different model architectures and approximating distributions, and use various regression and classification settings. Assessing the models' confidence quantitatively we can see how much we sacrifice in our attempt at deriving *practical* inference techniques in Bayesian neural networks (NNs).

More specifically, in this chapter we will look at how model structure affects model uncertainty by sampling functions from our prior. This can help in choosing an appropriate model for a given dataset. Given a dataset, we will then assess how different approximating distributions capture noisy data and uncertainty far from the data. We will compare the uncertainty estimates obtained from several stochastic regularisation techniques (SRTs) such as dropout and multiplicative Gaussian noise, as well as more traditional approximating distributions in variational inference (VI) such as a fully factorised Gaussian and a mixture of Gaussians. This is followed by a study of model uncertainty in classification, as well as specialised models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). We defer real-world examples to the next chapter.

4.1 Effects of model structure on uncertainty

We start by looking at how model structure affects uncertainty, i.e. the variation in functions captured by a given model structure. We look at the *a priori* distribution over functions, before performing inference. This distribution weighs some functions as being more likely by giving them a higher prior probability, and weighs other functions as being less likely by giving them a lower prior probability. In practice this results in a higher

penalty for the a priori less likely functions when we optimise our variational objective, trying to find a function to fit our data.

The distribution over functions can be visualised by sampling weight matrices from the prior with different non-linearities, model sizes, and number of hidden layers. We then evaluate the network with the sampled weights over a grid of points (for example $[-2, 2]$) and plot the results. Such a plot depicts a single draw from the prior over functions. Repeating this procedure multiple times we can get an idea of what the prior over functions looks like for a particular prior over weight matrices.

As a starting point, we evaluate networks with four hidden layers (fig. 4.1) and a choice of three non-linearities (ReLU in fig. 4.1g, TanH in fig. 4.1n, or Sigmoid in fig. 4.1u). We use the prior $\mathcal{N}(0, 1/l^2)$ over both the weights and biases, with different length-scale values l. We evaluate three model sizes: networks with 32 units in each hidden layer, 512 units, or 4096 units. We demonstrate the effects of prior choice by varying the prior length-scale. We experiment with length-scales 1 and 10 for the ReLU and TanH non-linearities, and length-scales 0.1 and 1 for the Sigmoid non-linearity. It is clear from the plots that longer length-scales result in smoother functions, and larger models result in more erratic functions[1]. ReLU networks seem to be mostly invariant to prior length-scale, with the bias magnitude changing the spread of the functions. A similar plot for a network with a single hidden layer is given in appendix B (fig. B.1).

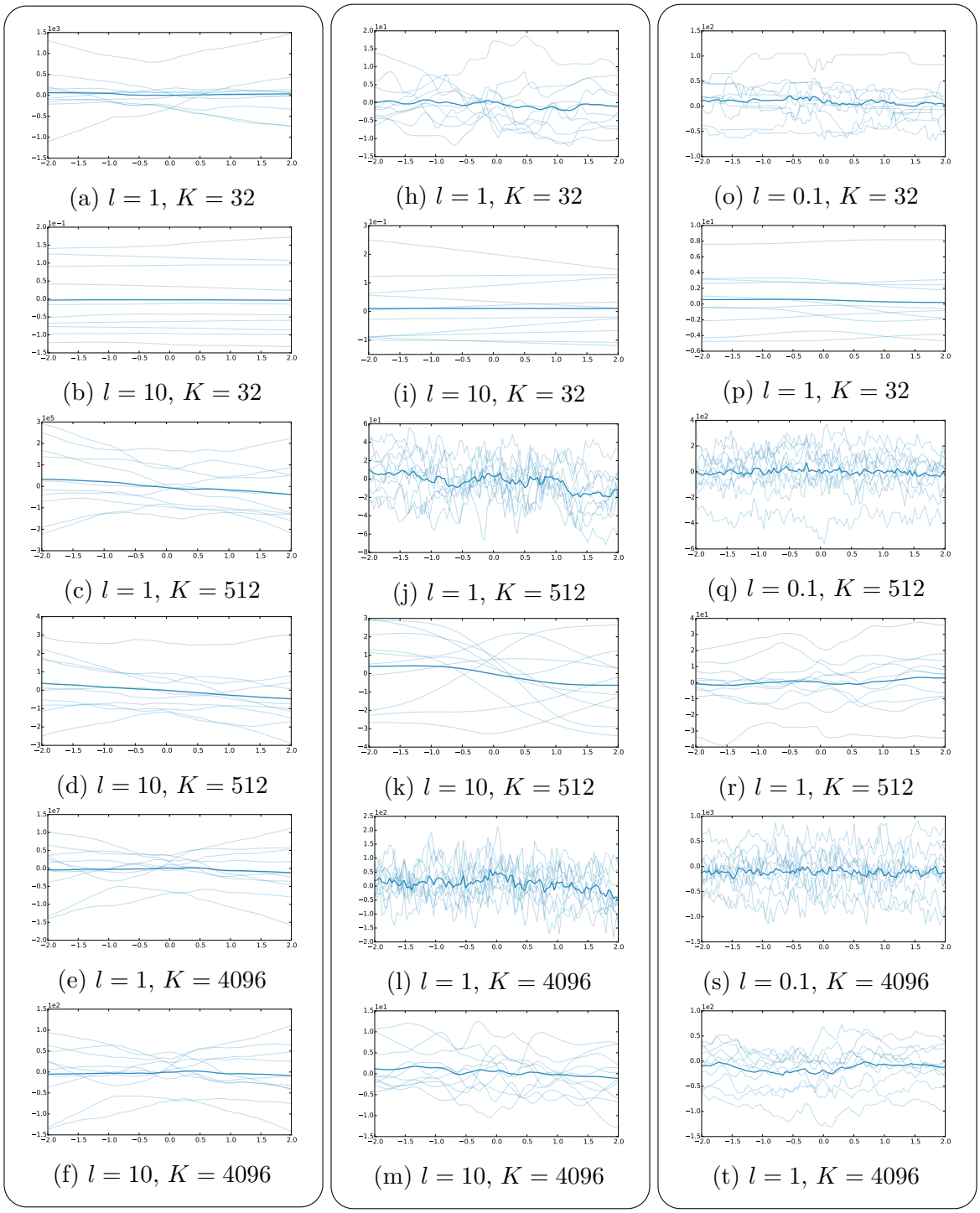
It is interesting to see that short length-scales in the lower layers yield more erratic functions, whereas the length-scale of the output layer ($l_5^{\mathbf{W}}$ for a network with 4 hidden layers) controls the output magnitude (fig. 4.2). The effect of the bias length-scale can be seen in fig. 4.3. Vertical distance between the functions changes with a short bias length-scale, as well as the frequency magnitude for the longer bias length-scale.

4.2 Effects of approximate posterior on uncertainty

We next assess the uncertainty estimates obtained from various approximating distributions on the task of regression. We compare the uncertainty obtained from different model architectures and non-linearities, both on tasks of extrapolation and interpolation. We use three regression datasets and model scalar functions which are easy to visualise. These are tasks one would often come across in real-world data analysis.

We start with a small synthetic dataset based on the one presented in [Snelson and Ghahramani, 2005]. This dataset consists of 5000 data points, with a fairly large amount

[1] This was also observed by [Neal, 1995]. Larger NNs can capture a larger class of functions, thus will have higher uncertainty as well. Note that a function's smoothness depends on its X-axis range.



(a) $l = 1$, $K = 32$ (h) $l = 1$, $K = 32$ (o) $l = 0.1$, $K = 32$

(b) $l = 10$, $K = 32$ (i) $l = 10$, $K = 32$ (p) $l = 1$, $K = 32$

(c) $l = 1$, $K = 512$ (j) $l = 1$, $K = 512$ (q) $l = 0.1$, $K = 512$

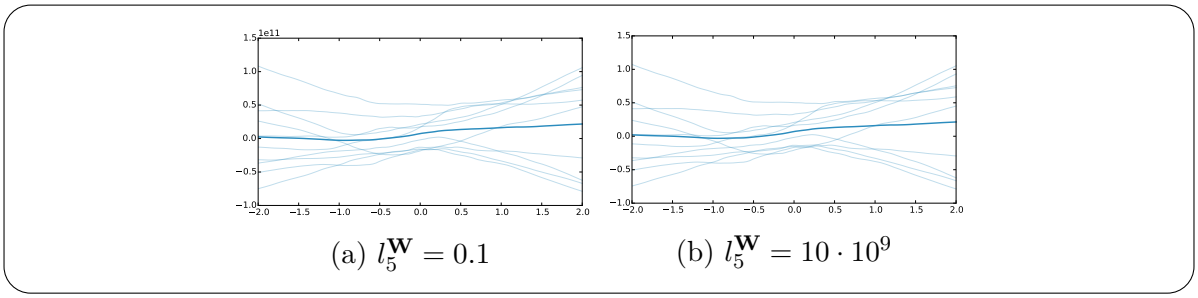
(d) $l = 10$, $K = 512$ (k) $l = 10$, $K = 512$ (r) $l = 1$, $K = 512$

(e) $l = 1$, $K = 4096$ (l) $l = 1$, $K = 4096$ (s) $l = 0.1$, $K = 4096$

(f) $l = 10$, $K = 4096$ (m) $l = 10$, $K = 4096$ (t) $l = 1$, $K = 4096$

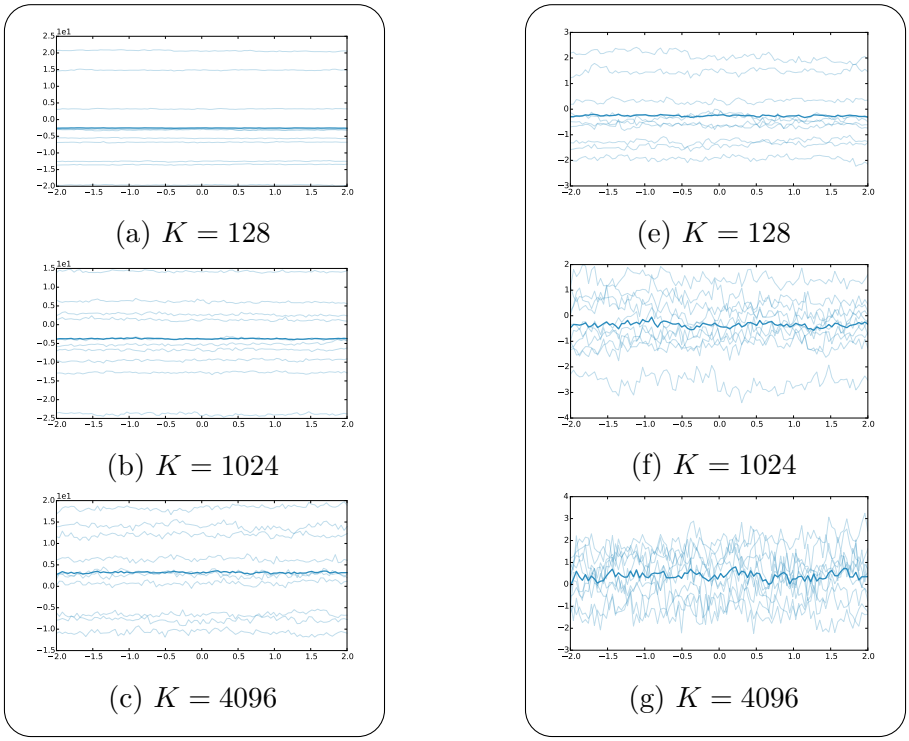
(g) ReLU (n) TanH (u) Sigmoid

Fig. 4.1 Draws from a Bayesian neural network prior ($\mathbf{W} \sim \mathcal{N}(0, 1/l^2)$) with $L = 4$ hidden layers, for various non-linearities, model sizes (K), and length-scales (l). Thick line is the mean of 20 samples. Best viewed on a computer screen.



(a) $l_5^{\mathbf{W}} = 0.1$ (b) $l_5^{\mathbf{W}} = 10 \cdot 10^9$

Fig. 4.2 Draws from a Bayesian neural network prior with $L = 4$ hidden layers, $K = 1024$ units, short length-scale $l = 0.1$, and ReLU non-linearity; Weight length-scale in the last layer $l_5^{\mathbf{W}}$ affects the Y axis magnitude.



(a) $K = 128$ (e) $K = 128$

(b) $K = 1024$ (f) $K = 1024$

(c) $K = 4096$ (g) $K = 4096$

(d) $l_5^{\mathbf{b}} = 0.1$ (h) $l_5^{\mathbf{b}} = 1$

Fig. 4.3 Effect of bias length-scale on draws from a Bayesian neural network prior. All networks use $L = 4$ hidden layers, short length-scale $l = 0.1$, and TanH non-linearity; We use $l_5^{\mathbf{W}} = 100$ to decreasing the output magnitude for drawing purposes. Note the change in vertical distance between the functions with short bias length-scale, and frequency magnitude for the longer bias length-scale.

of noise in the data. We evaluate model extrapolation as well as confidence on the noisy data. We then use a subset of the atmospheric CO_2 concentrations dataset derived from in situ air samples collected at Mauna Loa Observatory, Hawaii [Keeling et al., 2004] (referred to as CO_2) to evaluate model extrapolation on noise*less* data. We give further results on the reconstructed solar irradiance dataset [Lean, 2004] assessing model interpolation. The last two datasets are fairly small, with each dataset consisting of about 200 data points. We centred and normalised both datasets.

In the following experiments we assess VI approximate inference with various approximating distributions:

1. Bernoulli approximating distribution, implemented as **dropout**:

$$\boldsymbol{\omega} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{W}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{W}_2, \mathbf{b}\}$$

   with variational parameters $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}$ and $p(\boldsymbol{\epsilon}_l)$ $(l = 1, 2)$ a product of Bernoulli distributions with probability p_l.

2. Multiplicative Gaussian approximating distribution, implemented as **multiplicative Gaussian noise (MGN)**: $\boldsymbol{\omega} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{W}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{W}_2, \mathbf{b}\}$ with variational parameters $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}$ and $p(\boldsymbol{\epsilon}_l)$ (for $l = 1, 2$) a product of draws from $\mathcal{N}(1, p_i/(1 - p_i))$.

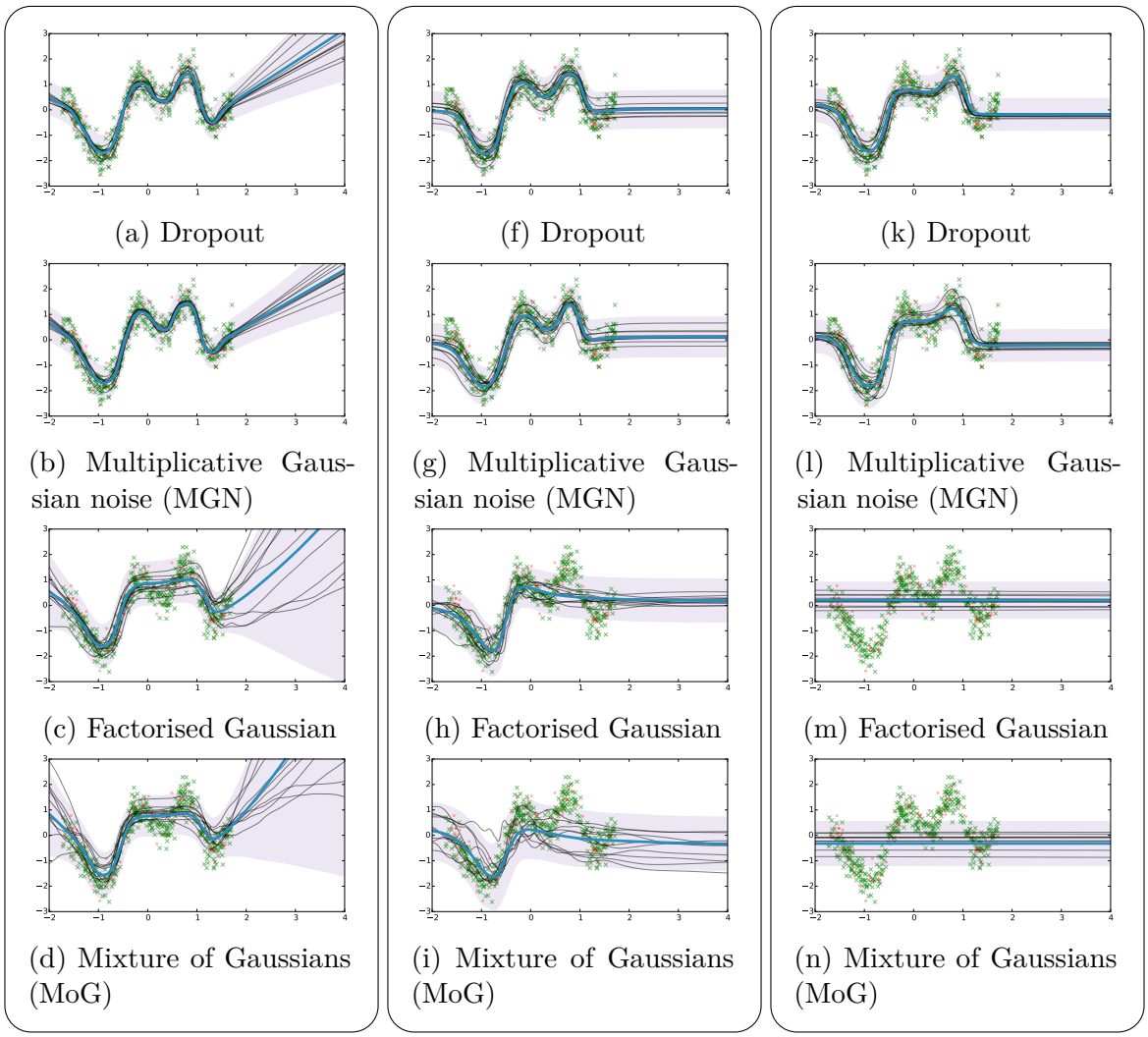
3. A fully **factorised Gaussian** distribution, similar to the one used in [Graves, 2011], but following the pathwise derivative estimator brought above.

4. A **mixture of Gaussians (MoG)** with two mixture components, factorised over the rows of the weight matrices (similar to the factorisation assumptions in the Bernoulli approximating distribution; this is identical to the mixture of Gaussians approximation in appendix A, but with the standard deviations not fixed at small values).

For the first two methods we assumed a delta approximating distribution over the biases, whereas in the last two we placed a factorised Gaussian over the bias. To avoid a big increase in the number of parameters we tied the standard deviations at each layer in these models, and used a single scalar parameter.

4.2.1 Regression

We start by assessing model uncertainty on the small and noisy [Snelson and Ghahramani, 2005] dataset. Figure 4.4 shows the fits of the two SRT approximating distributions



(a) Dropout (f) Dropout (k) Dropout

(b) Multiplicative Gaussian noise (MGN) (g) Multiplicative Gaussian noise (MGN) (l) Multiplicative Gaussian noise (MGN)

(c) Factorised Gaussian (h) Factorised Gaussian (m) Factorised Gaussian

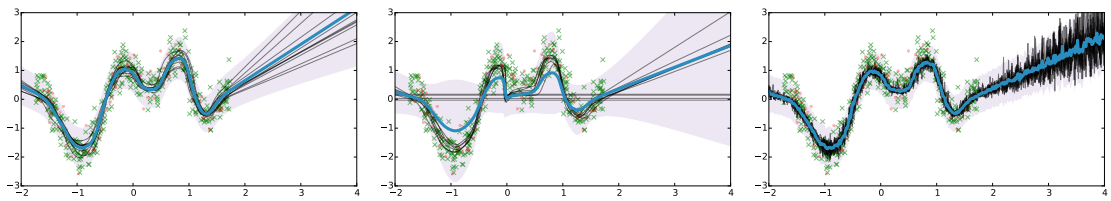
(d) Mixture of Gaussians (MoG) (i) Mixture of Gaussians (MoG) (n) Mixture of Gaussians (MoG)

(e) ReLU (j) TanH (o) Sigmoid

Fig. 4.4 Draws from a Bayesian neural network posterior with various approximating distributions; The networks used have $L = 4$ layers, each with $K = 1024$ units, length-scale $l = 5$, $p = 0.5$ for the mixture models, with different non-linearities. Shown are predictive mean (thick blue line), predictive uncertainty (shaded area, showing 2 standard deviations), and draws from the posterior (thin black lines). Scattered are training points. Best viewed on a computer screen.

(dropout and multiplicative Gaussian noise), as well as the factorised Gaussian and the mixture of Gaussians. This is shown for a network with $L = 4$ hidden layers, $K = 1024$ units in each layer, and various non-linearities. Each SRT model was optimised for 50 epochs using the Adam optimiser [Kingma and Ba, 2014], and the VI models were optimised for 100 epochs. Prior length-scale for all layers was set to $l = 5$, with weight decay set to $4 \cdot 10^{-5}$ (following eq. (3.14), with model precision fixed to its true value at $\tau = 10$). Note how for the ReLU models uncertainty increases far from the data for all approximating distributions, which is not the case with the other non-linearities. Note also how the factorised Gaussian and mixture of Gaussians seem to underfit compared to the SRTs. All models increase their uncertainty to capture the magnitude of the noisy data, with draws from the posteriors depicting the possible functions that can explain the data. Note also how dropout and MGN result in practically identical model uncertainty. Further results are given in appendix B, comparing model fits with different numbers of hidden layers ($L = 1$ and $L = 4$), model sizes ($K = 128$ and $K = 1024$), and non-linearities (ReLU, TanH, and Sigmoid) for dropout (fig. B.2), multiplicative Gaussian noise (fig. B.3), fully factorised Gaussian (fig. B.4), and mixture of Gaussians (fig. B.5). Interestingly, with the smaller models ($K = 128$) MoG and the factorised Gaussian do not underfit.

In the above in each model we set the SRT probability for the first layer to zero: $p_1 = 0$. To see why, observe the difference between fig. 4.5a and fig. 4.5b. Both depict the posterior by sampling a set of weights from the approximate posterior ($\mathbf{W}_i \sim q(\mathbf{W}_i)$)



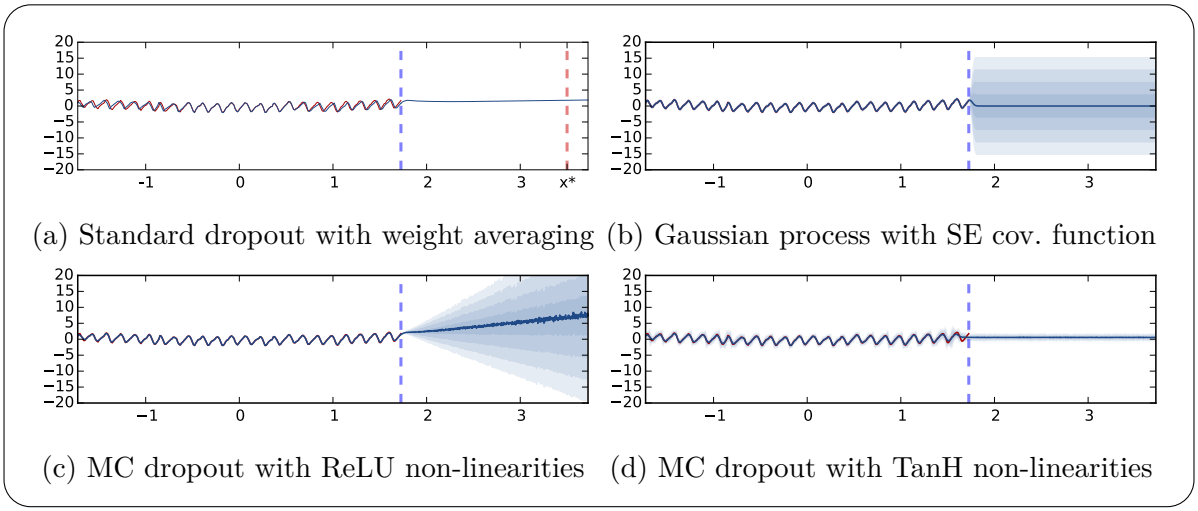
(a) i.i.d. draws per test **set** and $p_1 = 0$ (b) i.i.d. draws per test **set** and $p_1 = 0.5$ (c) i.i.d. draws per test **point** and $p_1 = 0$

Fig. 4.5 Draws from a Bayesian neural network posterior with **dropout** approximating distribution; The networks used have $L = 4$ layers, each with $K = 1024$ units, weight decay $\lambda = 4 \cdot 10^{-5}$ (equivalent to $l = 5$ with $N = 5000$ and $\tau = 10$), and ReLU non-linearities. We have $p = 0.5$ for all layers apart from the first (where we have either $p_1 = 0$ or $p_1 = 0.5$). All networks were trained with dropout masks sampled i.i.d. for each data point, but tested differently. Some were tested by drawing a single mask for the entire test set multiple times (**i.i.d. draws per test set**), whereas others by drawing a different mask for each test point multiple times (**i.i.d. draws per test point**).

and evaluating the network with the sampled weights on the entire space (this is in contrast to sampling a new set of realisations for each test point, as is depicted in figure 4.5c). In fig. 4.5b we set all dropout probabilities to 0.5. As a result, with probability 0.5, the sampled functions from the posterior would be identically zero. This is because a zero draw from the Bernoulli distribution together with a scalar input leads the model to completely drop its input. This is a behaviour we might not believe the posterior should exhibit, and could change this by setting a different probability for the first layer. Setting $p_1 = 0$ for example is identical to placing a delta approximating distribution over the first weight layer. Note that the dropout probability could be optimised instead.

Next, we trained several models on the noiseless CO_2 dataset. We use NNs with either 4 or 5 hidden layers and 1024 hidden units. We use either ReLU non-linearities or TanH non-linearities in each network, and use dropout probabilities of either 0.1 or 0.2. Extrapolation results are shown in figure 4.6. The model is trained on the training data (left of the dashed blue line), and tested on the entire dataset. Fig. 4.6a shows the results for standard dropout (i.e. with weight averaging and without assessing model uncertainty) for the 5 layer ReLU model. Fig. 4.6b shows the results obtained from a Gaussian process with a squared exponential covariance function for comparison. Fig. 4.6c shows the results of the same network as in fig. 4.6a, but with MC dropout used to evaluate the predictive mean and uncertainty for the training and test sets. Lastly, fig. 4.6d shows the same using the TanH network with 5 layers (plotted with 8 times the standard deviation for visualisation purposes). The shades of blue represent model uncertainty: each colour gradient represents half a standard deviation (in total, predictive mean plus/minus 2 standard deviations are shown, representing 95% confidence). Not plotted are the models with 4 layers as these converge to the same results.

Extrapolating the observed data, none of the models can capture the periodicity (although with a suitable covariance function the Gaussian process (GP) will capture it well). The standard dropout NN model (fig. 4.6a) predicts value 0 for point x^* (marked with a dashed red line) with high confidence, even though it is clearly not a sensible prediction. The GP model represents this by increasing its predictive uncertainty—in effect declaring that the predictive value might be 0 but the model is uncertain. This behaviour is captured in MC dropout as well. Even though the models in figures 4.6 have an incorrect predictive mean, the increased standard deviation expresses the models' uncertainty about the point. Note that as before the uncertainty is increasing far from the data for the ReLU model, whereas for the TanH model it stays bounded. For the TanH model we assessed the uncertainty using both dropout probability 0.1 and dropout probability 0.2. Models initialised with dropout probability 0.1 initially exhibit



(a) Standard dropout with weight averaging (b) Gaussian process with SE cov. function

(c) MC dropout with ReLU non-linearities (d) MC dropout with TanH non-linearities

Fig. 4.6 **Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset, for various models.** In red is the observed function (left of the dashed blue line); in blue is the predictive mean plus/minus two standard deviations (8 for fig. 4.6d). Different shades of blue represent half a standard deviation. Marked with a dashed red line is a point far away from the data: standard dropout confidently predicts an unreasonable value for the point; the other models predict unreasonable values as well but with the additional information that the models are uncertain about their predictions.

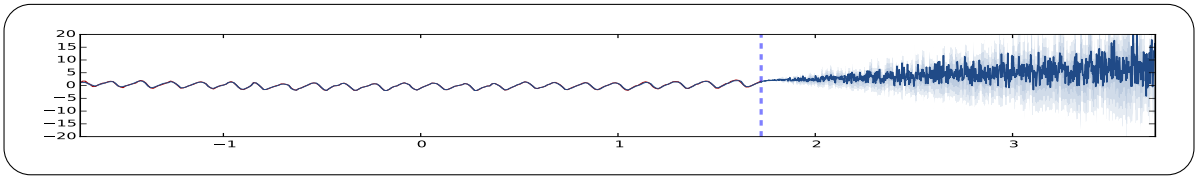


Fig. 4.7 Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset for the MC dropout model with ReLU non-linearities, approximated with 10 samples.

smaller uncertainty than the ones initialised with dropout probability 0.2, but towards the end of the optimisation when the model has converged the uncertainty is almost indistinguishable. It is worth mentioning that we attempted to fit the data with models with a smaller number of layers unsuccessfully.

The number of forward iterations used to estimate the uncertainty (T) was 1000 for drawing purposes. A much smaller numbers can be used to get a reasonable estimation to the predictive mean and uncertainty (see fig. 4.7 for example with $T = 10$).

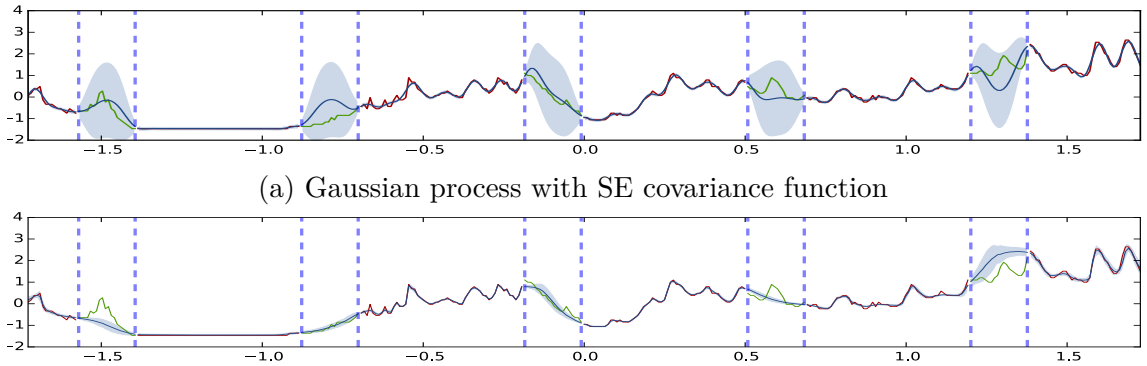
For interpolation we repeat the experiment with ReLU networks with 5 hidden layers and the same setup on a new dataset: solar irradiance. Here we use weight decay of $5 \cdot 10^{-7}$. Interpolation results are shown in fig. 4.8. Fig. 4.8a shows interpolation of missing sections (bounded between pairs of dashed blue lines) for the Gaussian process with squared exponential covariance function, as well the function value on the training set. In red is the observed function, in green are the missing sections, and in blue is the model predictive mean. Fig. 4.8b shows the same for the ReLU dropout model with 5 layers. Both models interpolate the data well, with increased uncertainty over the missing segments. However, VI underestimates model uncertainty considerably here[2]. This experiment demonstrates the dangers with model uncertainty resulting from VI approaches. Note however that all variational inference techniques would under-estimate model uncertainty unless the true posterior is in the class of approximating variational distributions.

4.2.2 Classification

To assess model confidence in classification we test a convolutional neural network trained on the MNIST dataset [LeCun and Cortes, 1998]. We trained the LeNet convolutional neural network model [LeCun et al., 1998] with dropout applied before the last fully connected inner-product layer (the usual way dropout is used in CNNs). We used dropout probability of 0.5. We trained the model for 10^6 iterations. We used Caffe [Jia et al., 2014] reference implementation for this experiment.

We evaluated the trained model on a continuously rotated image of the digit 1 (shown on the X axis of fig. 4.9). We scatter 100 stochastic forward passes of the softmax input (the output from the last fully connected layer, fig. 4.9a), as well as of the softmax output for each of the top classes (fig. 4.9b). The plots show the softmax input value and softmax output value for the 3 digits with the largest values for each corresponding input. When the softmax input for a class is larger than that of all other classes (class 1 for the first

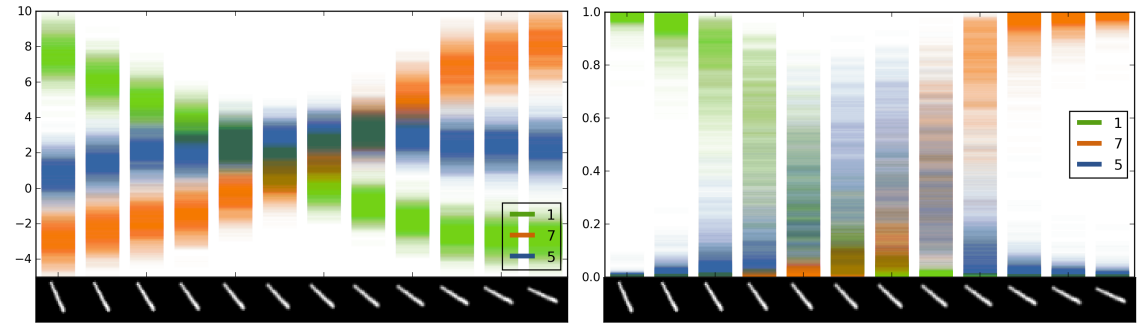
[2]Note the extremely high model precision used: $\tau = \frac{(1-p)l^2}{2N\lambda} = \frac{l^2}{4}10^4$ based on eq. (3.17).



(a) Gaussian process with SE covariance function

(b) MC dropout with ReLU non-linearities

Fig. 4.8 **Predictive mean and uncertainties on the reconstructed solar irradiance dataset with missing segments, for the GP and MC dropout approximation.** In red is the observed function and in green are the missing segments. In blue is the predictive mean plus/minus two standard deviations of the various approximations.



(a) Softmax *input* scatter

(b) Softmax *output* scatter

Fig. 4.9 **A scatter of 100 forward passes of the softmax input and output for dropout LeNet.** On the X axis is a rotated image of the digit 1. The input is classified as digit 5 for images 6-7, even though model uncertainty is extremly large (best viewed in colour).

5 images, class 5 for the next 2 images, and class 7 for the rest in fig 4.9a), the model predicts the corresponding class. For the 12 images, the model predicts classes [1 1 1 1 1 5 5 7 7 7 7 7]. Looking at the softmax input values, if the range of high uncertainty of a class is far from that of other classes' (for example the left most image) then the input is classified with high confidence. On the other hand, if the range of high uncertainty intersects that of other classes (such as in the case of the middle input image), then even though the softmax output can be arbitrarily high (as far as 1 if the mean is far from the means of the other classes), the softmax output uncertainty can be as large as the entire space. This signifies the model's uncertainty in its softmax output value—i.e. in the prediction. In this scenario it would not be reasonable to use argmax to return class 5 for the middle image when its uncertainty is so high. One would expect the model to ask an external annotator for a label for this input.

4.3 Quantitative comparison

We next assess the models' confidence quantitatively to see how much we sacrifice in our attempt at deriving *practical* inference techniques in Bayesian NNs. We compare the Bernoulli approximating distribution (corresponding to dropout) to two existing inference methods: [Graves, 2011] and [Hernandez-Lobato and Adams, 2015]. Quite surprisingly, we show that by using dropout's uncertainty we can obtain a considerable improvement in predictive log-likelihood and root mean square error (RMSE) compared to these techniques.

Predictive log-likelihood captures how well a model fits the data, with larger values indicating better model fit. Uncertainty quality can be determined from this quantity as well. We replicate the experiment set-up in Hernandez-Lobato and Adams [2015] and compare the RMSE and predictive log-likelihood of dropout (referred to as "Dropout" in the experiments) to that of Probabilistic Back-propagation (referred to as "PBP", [Hernandez-Lobato and Adams, 2015]) and to a popular variational inference technique in Bayesian NNs (referred to as "VI", [Graves, 2011]). The aim of this experiment is to compare the uncertainty quality obtained from a *naive* application of dropout in NNs to that of specialised methods developed to capture uncertainty.

Following our Bayesian interpretation of dropout we need to define a prior length-scale, and find an optimal model precision parameter τ which will allow us to evaluate the predictive log-likelihood (eq. (3.18)). Similar to [Hernandez-Lobato and Adams, 2015] we use Bayesian optimisation (BO, [Snoek et al., 2012, 2015]) over validation log-likelihood to find optimal τ, and set the prior length-scale to 10^{-2} for most datasets based on the

range of the data. Note that this is a standard dropout NN, where the prior length-scale l and model precision τ are simply used to define the model's weight decay through eq. (3.17). We used dropout with probabilities 0.05 and 0.005 since the network size is very small (with 50 units following [Hernandez-Lobato and Adams, 2015]) and the datasets are fairly small as well. The BO runs used 40 iterations following the original setup, but after finding the optimal parameter values we used 10x more iterations, as dropout takes longer to converge. Even though the model doesn't converge within 40 iterations, it gives BO a good indication of whether a parameter is good or not. Finally, we used mini-batches of size 32 and the Adam optimiser [Kingma and Ba, 2014]. Further details about the various datasets are given in [Hernandez-Lobato and Adams, 2015].

The results are shown in table 4.1. Dropout significantly outperforms all other models both in terms of RMSE as well as test log-likelihood on all datasets apart from Yacht, for which PBP obtains better RMSE. All experiments were averaged on 20 random splits of the data (apart from Protein for which only 5 splits were used and Year for which one split was used). It is interesting to note that the median for most datasets gives much better performance than the mean. For example, on the Boston Housing dataset dropout achieves median RMSE of 2.68 with an IQR interval of [2.45, 3.35] (compared to mean 2.97) and predictive log-likelihood median of -2.34 with IQR [-2.54, -2.29] (compared to mean -2.46). In the Concrete Strength dataset dropout achieves median RMSE of 5.15 (compared to mean 5.23)[3].

To implement the model we used Keras [Chollet, 2015], an open source deep learning package based on Theano [Bergstra et al., 2010]. In [Hernandez-Lobato and Adams, 2015] BO for VI seems to require a considerable amount of additional time compared to PBP. However our model's running time (including BO) is comparable to PBP's Theano implementation. On the Naval Propulsion dataset for example our model takes 276 seconds on average per split (start-to-finish, divided by the number of splits). Out of that, with the optimal parameters BO found, model training took 95 seconds. This is in comparison to PBP's 220 seconds. For Kin8nm our model requires 188 seconds on average including BO, 65 seconds without, compared to PBP's 156 seconds.

Dropout's RMSE in table 4.1 is given by averaging stochastic forward passes through the network following eq. (3.16) (MC dropout). We observed an improvement using this estimate compared to the standard dropout weight averaging, and also compared to much smaller dropout probabilities (near zero). For the Boston Housing dataset for example, repeating the same experiment with dropout probability 0 results in RMSE of

[3]Full raw results and code are available online at `https://github.com/yaringal/DropoutUncertaintyExps`.

			Avg. Test RMSE and Std. Errors			Avg. Test LL and Std. Errors		
Dataset	N	Q	**VI**	**PBP**	**Dropout**	**VI**	**PBP**	**Dropout**
Boston Housing	506	13	4.32 ±0.29	3.01 ±0.18	**2.97 ±0.19**	-2.90 ±0.07	-2.57 ±0.09	**-2.46 ±0.06**
Concrete Strength	1,030	8	7.19 ±0.12	5.67 ±0.09	**5.23 ±0.12**	-3.39 ±0.02	-3.16 ±0.02	**-3.04 ±0.02**
Energy Efficiency	768	8	2.65 ±0.08	1.80 ±0.05	**1.66 ±0.04**	-2.39 ±0.03	-2.04 ±0.02	**-1.99 ±0.02**
Kin8nm	8,192	8	**0.10 ±0.00**	**0.10 ±0.00**	**0.10 ±0.00**	0.90 ±0.01	0.90 ±0.01	**0.95 ±0.01**
Naval Propulsion	11,934	16	**0.01 ±0.00**	**0.01 ±0.00**	**0.01 ±0.00**	3.73 ±0.12	3.73 ±0.01	**3.80 ±0.01**
Power Plant	9,568	4	4.33 ±0.04	4.12 ±0.03	**4.02 ±0.04**	-2.89 ±0.01	-2.84 ±0.01	**-2.80 ±0.01**
Protein Structure	45,730	9	4.84 ±0.03	4.73 ±0.01	**4.36 ±0.01**	-2.99 ±0.01	-2.97 ±0.00	**-2.89 ±0.00**
Wine Quality Red	1,599	11	0.65 ±0.01	0.64 ±0.01	**0.62 ±0.01**	-0.98 ±0.01	-0.97 ±0.01	**-0.93 ±0.01**
Yacht Hydrodynamics	308	6	6.89 ±0.67	**1.02 ±0.05**	1.11 ±0.09	-3.43 ±0.16	-1.63 ±0.02	**-1.55 ±0.03**
Year Prediction MSD	515,345	90	9.034 ±NA	8.879 ±NA	**8.849 ±NA**	-3.622 ±NA	-3.603 ±NA	**-3.588 ±NA**

Table 4.1 **Average test performance in RMSE and predictive log likelihood** for a popular variational inference method (**VI**, Graves [2011]), Probabilistic back-propagation (**PBP**, Hernandez-Lobato and Adams [2015]), and dropout uncertainty (**Dropout**). Dataset size (N) and input dimensionality (Q) are also given.

	Avg. Test RMSE and Std. Errors			Avg. Test LL and Std. Errors		
Dataset	**Dropout**	**10x Epochs**	**2 Layers**	**Dropout**	**10x Epochs**	**2 Layers**
Boston Housing	2.97 ±0.19	2.80 ±0.19	2.80 ±0.13	-2.46 ±0.06	-2.39 ±0.05	-2.34 ±0.02
Concrete Strength	5.23 ±0.12	4.81 ±0.14	4.50 ±0.18	-3.04 ±0.02	-2.94 ±0.02	-2.82 ±0.02
Energy Efficiency	1.66 ±0.04	1.09 ±0.05	0.47 ±0.01	-1.99 ±0.02	-1.72 ±0.02	-1.48 ±0.00
Kin8nm	0.10 ±0.00	0.09 ±0.00	0.08 ±0.00	0.95 ±0.01	0.97 ±0.01	1.10 ±0.00
Naval Propulsion	0.01 ±0.00	0.00 ±0.00	0.00 ±0.00	3.80 ±0.01	3.92 ±0.01	4.32 ±0.00
Power Plant	4.02 ±0.04	4.00 ±0.04	3.63 ±0.04	-2.80 ±0.01	-2.79 ±0.01	-2.67 ±0.01
Protein Structure	4.36 ±0.01	4.27 ±0.01	3.62 ±0.01	-2.89 ±0.00	-2.87 ±0.00	-2.70 ±0.00
Wine Quality Red	0.62 ±0.01	0.61 ±0.01	0.60 ±0.01	-0.93 ±0.01	-0.92 ±0.01	-0.90 ±0.01
Yacht Hydrodynamics	1.11 ±0.09	0.72 ±0.06	0.66 ±0.06	-1.55 ±0.03	-1.38 ±0.01	-1.37 ±0.02

Table 4.2 **Average test performance in RMSE and predictive log likelihood** for dropout uncertainty as above (**Dropout**), the same model optimised with 10 times the number of epochs and identical model precision (**10x epochs**), and the same model again with 2 layers instead of 1 (**2 Layers**).

3.07 and predictive log-likelihood of -2.59. This demonstrates that dropout significantly affects the predictive log-likelihood and RMSE, even though the dropout probability is fairly small.

Remark. We used dropout following the same way the method would be used in current research—without adapting model structure and without optimising the objective using a derivative w.r.t. the weight decay[a]. We used this standard methodology to demonstrate the results that could be obtained from existing models, with the only change being the MC dropout evaluation. Further, in the results above we attempted to match PBP's run time (hence used only 10x more epochs compared to PBP's 40 epochs). Experimenting with 100x more epochs compared to PBP (10x more epochs compared to the results in table 4.1) gives a considerable improvement both in terms of test RMSE as well as test log-likelihood over the results in table 4.1. We further assessed a model with the same number of units and 2 hidden layers instead of 1. Both experiments are shown in table 4.2. Experimenting with different network architectures we expect the method to give further improved uncertainty estimates.

It is also important to mention the results collected by Bui et al. [2016] extending on the above. Bui et al. [2016] have repeated the experiment setup above, evaluating many more models including the Gaussian process, deep Gaussian processes, and Bayesian neural networks trained with SGLD [Welling and Teh, 2011] as well as HMC [Neal, 1995]. Bui et al. [2016] found[b] SGLD's performance to be similar to dropout in table 4.1, with SGLD outperforming dropout on 5 out of the 10 datasets, and dropout outperforming SGLD on the other 5. HMC seemed to supersede all other techniques on average (although both SGLD and HMC require much longer run-times). I would mention that these experiments were done on single hidden layer NNs with 50 units, and the results might not be the same with larger models.

[a] Note that in the Gaussian processes literature the objective would be optimised w.r.t. the model precision τ, corresponding to our NN's weight decay through eq. (3.17). In the deep learning literature though the weight decay would often be grid-searched over to minimise validation error. We repeated this standard approach here. However to be more efficient we used Bayesian optimisation to determine the weight decay instead of searching over a grid. Alternatively, the objective could be optimised w.r.t. the model precision τ as in the GP case, which we demonstrate in section §4.6.

[b] Note that our reported dropout standard error was erroneously scaled-up by a factor of 4.5 at the time of publication (i.e. for Boston RMSE we reported standard error 0.85 instead of 0.19). The erroneous results are the ones used in [Bui et al., 2016].

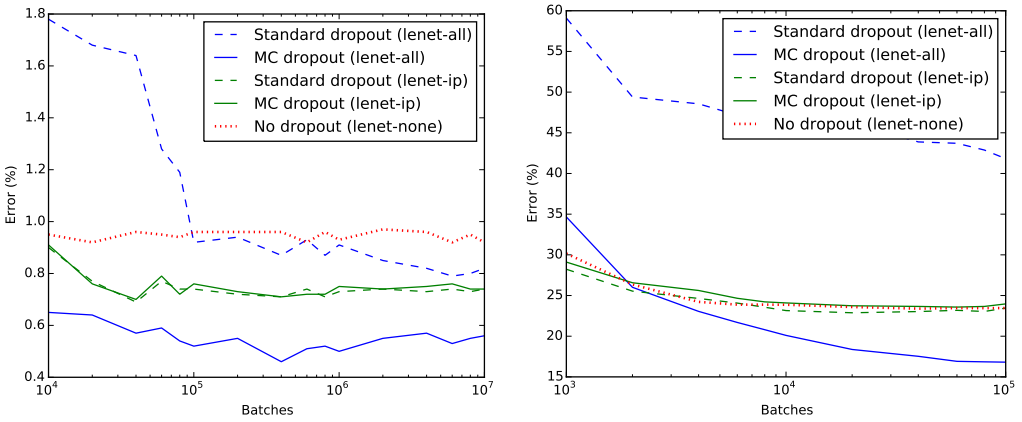
4.4 Bayesian convolutional neural networks

We give further empirical evaluation with specialised model structures—Bayesian convolutional neural networks (CNNs). This extends on the experiments above where dropout was applied only after the inner-product layers of a CNN. In comparison, here we perform dropout after all convolution and weight layers, viewed as approximate inference in a Bayesian CNN following the results of the previous chapter. We assess the LeNet network structure [LeCun et al., 1998] on MNIST [LeCun and Cortes, 1998] and CIFAR-10 [Krizhevsky and Hinton, 2009] with different settings, and show improvement in test accuracy compared to existing techniques in the literature. We evaluate the number of samples needed to obtain an improvement in results using MC dropout (eq. (3.8)), and finish with improved results on CIFAR-10 obtained by an almost trivial change of an existing model. All experiments were done using the Caffe framework [Jia et al., 2014], requiring identical training time to that of standard CNNs[4].

In this section we refer to our Bayesian CNN implementation with dropout used after every parameter layer as "**lenet-all**". We compare this model to a CNN with dropout used after the inner-product layers at the end of the network alone—the traditional use of dropout in the literature. We refer to this model as "**lenet-ip**". Additionally we compare to LeNet as described originally in [LeCun et al., 1998] with no dropout at all, referred to as "**lenet-none**". We evaluate each dropout network structure (lenet-all and lenet-ip) using two testing techniques. The first is using weight averaging, the standard way dropout is used in the literature (referred to as "**Standard dropout**"). We use the Caffe [Jia et al., 2014] reference implementation for this. The second testing technique interleaves Bayesian methodology into deep learning. We average T stochastic forward passes through the model following the Bayesian interpretation of dropout derived in eq. (3.8). This technique is referred to as "**MC dropout**". In this experiment we average $T = 50$ forward passes through the network.

Figure 4.10 shows classification error as a function of batches *on log scale* for all three models (lenet-all, lenet-ip, and lenet-none) with the two different testing techniques (Standard dropout and MC dropout) for MNIST (fig. 4.10a) and CIFAR-10 (fig. 4.10b). It seems that Standard dropout in lenet-ip results in improved results compared to lenet-none, with the results more pronounced on the MNIST dataset than on CIFAR-10. When the Standard dropout testing technique is used with our Bayesian CNN (with dropout applied after every parameter layer—lenet-all) performance suffers. However, by averaging the forward passes of the same network—using the same network weights—the

[4]The configuration files are available online at
https://github.com/yaringal/DropoutUncertaintyCaffeModels.



(a) MNIST (b) CIFAR-10

Fig. 4.10 **Test error for LeNet with dropout applied after every weight layer (lenet-all—our Bayesian CNN implementation, blue), dropout applied after the fully connected layer alone (lenet-ip, green), and without dropout (lenet-none, dotted red line).** Standard dropout is shown with a dashed line, MC dropout is shown with a solid line. Note that although Standard dropout lenet-all performs very badly on both datasets (dashed blue line), when evaluating *the same network* with MC dropout (solid blue line) the model outperforms all others.

performance of lenet-all supersedes that of all other models ("MC dropout lenet-all" in both 4.10a and 4.10b).

Most existing CNN literature uses Standard dropout after the fully-connected layers alone, equivalent to "Standard dropout lenet-ip" in our experiment [Krizhevsky et al., 2012]. Srivastava et al. [2014] claim, based on empirical observations, that Standard dropout gives very close results to MC dropout in standard NN architectures, but the claim was not verified with CNN architectures. Dropout is not used in CNNs after convolution layers in existing literature perhaps because empirical results with Standard dropout suggested deteriorated performance (as can also be seen in our experiments). Standard dropout approximates model output during test time by propagating the mean of each layer to the next. However this can be a crude approximation to the predictive mean of the model, which can otherwise be obtained by Monte Carlo averaging of stochastic forward passes through the model (eq. (3.8)). The empirical results given in Srivastava et al. [2014, section 7.5] suggested that Standard dropout is equivalent to MC dropout, and it seems that most research has followed this approximation. However the results we obtained in our experiments suggest that the Standard dropout approximation can fail in some model architectures.

4.4.1 Model over-fitting

We evaluate the models' tendency to over-fit on training sets decreasing in size. We use the same experiment set-up as above, without changing the dropout ratio for smaller datasets and without increasing model weight decay. We randomly split the MNIST dataset into smaller training sets of sizes 1/4 and 1/32 fractions of the full set. We evaluated the lenet-all model with MC dropout compared to lenet-ip with Standard dropout—the standard approach in the field. We did not compare to lenet-none as it is known to over-fit even on the full MNIST dataset.

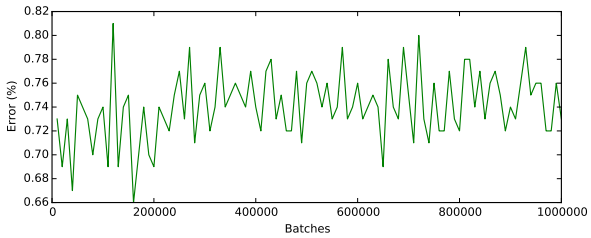
The results are shown in fig. 4.11. For the entire MNIST dataset (figs. 4.11a and 4.11b) none of the models seem to over-fit (with lenet-ip performing worse than lenet-all). It seems that even for a quarter of the MNIST dataset (15,000 data points) the Standard dropout technique starts over-fitting (fig. 4.11c). In comparison lenet-all performs well on this dataset (obtaining better classification accuracy than the best result of Standard dropout on lenet-ip). When using a smaller dataset with 1,875 training examples it seems that both techniques over-fit, and other forms of regularisation are needed.

4.4.2 MC dropout in standard convolutional neural networks

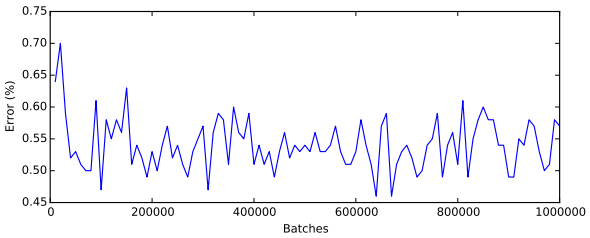
We next evaluate the use of Standard dropout compared to MC dropout on existing well known[5] CNN architectures in the literature. We evaluated two well known models that have achieved state-of-the-art results on CIFAR-10 in the past several years. The first is Network in network (NIN) [Lin et al., 2013]. The model was extended by [Lee et al., 2014] who added multiple loss functions after some of the layers—in effect "encouraging" the bottom layers to explain the data better. The new model was named a Deeply supervised network (DSN). The same idea was used in [Szegedy et al., 2014] to achieve state-of-the-art results on ImageNet.

We assessed these models on the CIFAR-10 dataset, as well as on an augmented version of the dataset for the DSN model [Lee et al., 2014]. We replicated the experiment set-up as it appeared in the original papers, and evaluated the models' test error using Standard dropout as well as using MC dropout, averaging $T = 100$ forward passes. MC dropout testing gives us a noisy estimate, with potentially different test results over different runs. To get faithful results one would need to repeat each experiment several times to get a mean and standard deviation for the test error. We therefore repeated the experiment 5 times and report the average test error. In table 4.3 we report predictive mean, as well as the standard deviation resulting from the multiple repetitions

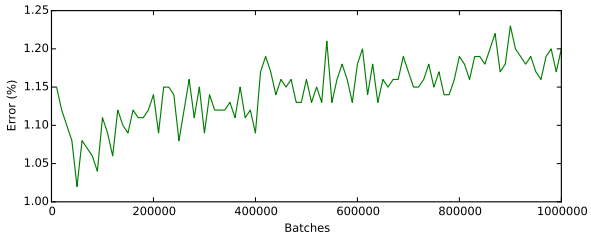
[5] Using http://rodrigob.github.io as a reference.



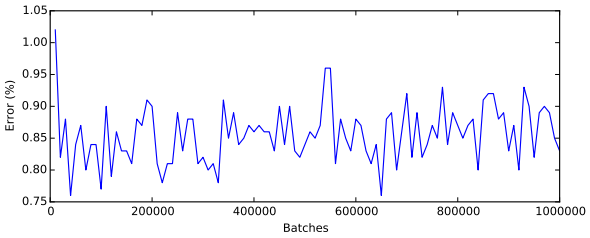
(a) Entire MNIST, Standard dropout+lenet-ip



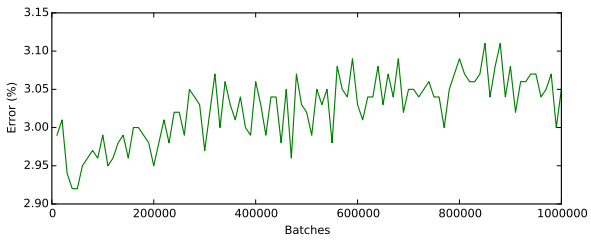
(b) Entire MNIST, MC dropout+lenet-all



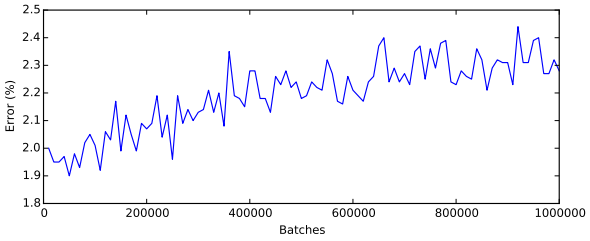
(c) 1/4 of MNIST, Standard dropout+lenet-ip



(d) 1/4 of MNIST, MC dropout+lenet-all



(e) 1/32 of MNIST, Standard dropout+lenet-ip



(f) 1/32 of MNIST, MC dropout+lenet-all

Fig. 4.11 **Test error of LeNet trained on random subsets of MNIST decreasing in size.** To the left in green are networks with dropout applied after the last layer alone (lenet-ip) and evaluated with Standard dropout (the standard approach in the field), to the right in blue are networks with dropout applied after every weight layer (lenet-all) and evaluated with MC dropout—our Bayesian CNN implementation. Lenet-ip starts over-fitting even with a quarter of the dataset (also, note the different Y-axis scales). With a small enough dataset, both models over-fit. MC dropout was used with 10 samples.

	CIFAR Test Error (and Std.)	
Model	**Standard Dropout**	**MC Dropout**
NIN	10.43	$\mathbf{10.27 \pm 0.05}$
DSN	9.37	$\mathbf{9.32 \pm 0.02}$
Augmented-DSN	7.95	$\mathbf{7.71 \pm 0.09}$

Table 4.3 **Test error on CIFAR-10 with the same networks (model structure and weights) evaluated using both Standard dropout as well as MC dropout** ($T = 100$, averaged with 5 repetitions and given with standard deviation). MC dropout achieves consistent improvement in test error compared to Standard dropout.

of the experiment to see if the improvement is statistically significant. It seems that MC dropout results in a statistically significant improvement for all three models (NIN, DSN, and Augmented-DSN), with the largest increase for Augmented-DSN.

Remark. It is interesting to note that we observed no improvement on ImageNet [Deng et al., 2009] using the same models. From initial experimentation, it seems that the large number of classes in ImageNet results in a predictive mean for the true class which is only slightly higher than the other softmax outputs. As a result, the ranges of high uncertainty overlap considerably and MC dropout does not perform well.

4.4.3 MC estimate convergence

Lastly, we assessed the usefulness of the proposed method in practice for applications in which efficiency during test time is important. We give empirical results suggesting that 20 samples are enough to improve performance on some datasets. We evaluated the last model (Augmented-DSN) with MC dropout for $T = 1, ..., 100$ samples. We repeated the experiment 5 times and averaged the results. In fig. 4.12 we see that within 20 samples the error is reduced by more than one standard deviation. Within 100 samples the error converges to 7.71.

This replicates the experiment in [Srivastava et al., 2014, section 7.5], here with the augmented CIFAR-10 dataset and the DSN CNN model. Compared to [Srivastava et al., 2014, section 7.5] we obtained a significant reduction in test error. This might be because CNNs exhibit different characteristics from standard NNs. We speculate that the non-linear pooling layer affects the dropout approximation considerably.

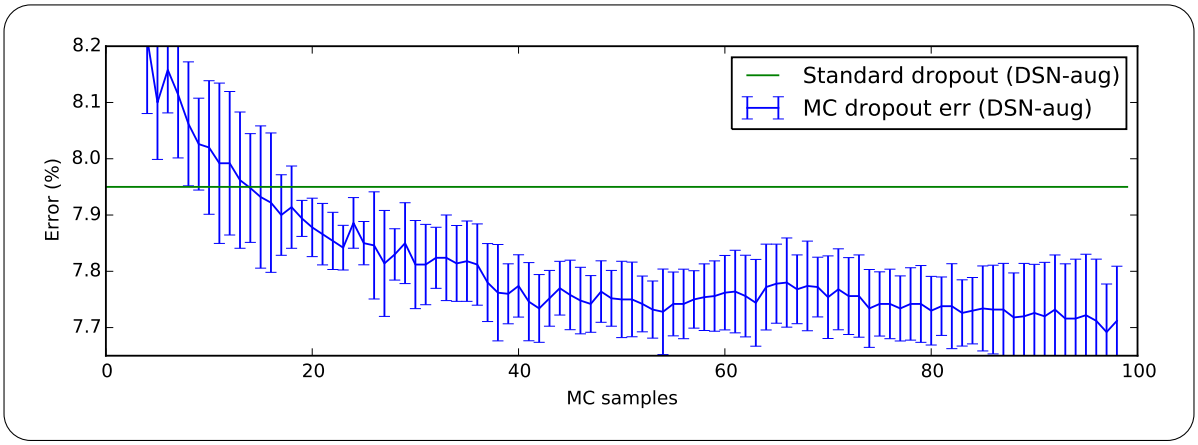


Fig. 4.12 **Augmented-DSN test error for different number of averaged forward passes in MC dropout** (blue) averaged with 5 repetitions, shown with 1 standard deviation. In green is test error with Standard dropout. MC dropout achieves a significant improvement (more than 1 standard deviation) after 20 samples.

4.5 Recurrent neural networks

Extending on the work above with Bayesian CNNs, next we assess a second specialised NN model—the Bayesian recurrent neural network (RNN). Here we take a different approach though. We start with the Bayesian model developed in §3.4.2 and derive a new SRT, or more specifically derive a new dropout variant. This is an interesting use of the ideas above; existing literature in RNNs has established that dropout cannot be applied in RNNs apart from the forward connections [Bayer et al., 2013; Bluche et al., 2015; Pachitariu and Sahani, 2013; Pham et al., 2014; Zaremba et al., 2014]. In these papers different network units are dropped at different time steps in the forward connections, and no dropout is applied to the recurrent connections. However, based on the developments in the previous chapter we propose a new dropout variant that can be successfully applied in RNNs, and assess the model's predictive mean.

We replicate the language modelling experiment of [Zaremba et al., 2014]. The experiment uses the Penn Treebank, a standard benchmark in the natural language processing field. This dataset is considered to be a small one in the community, with $887,521$ tokens (words) in total, making overfitting a considerable concern. Throughout the experiments we refer to LSTMs with the dropout technique proposed following our Bayesian interpretation in §3.4.2 as *Variational LSTMs*, and refer to existing dropout techniques as *naive dropout LSTMs* (different masks at different steps, applied to the input and output of the LSTM alone). We refer to LSTMs with no dropout as *standard LSTMs*.

	Medium LSTM			Large LSTM		
	Validation	Test	WPS	Validation	Test	WPS
Non-regularized (early stopping)	121.1	121.7	5.5K	128.3	127.4	2.5K
Zaremba et al. [2014]	86.2	82.7	5.5K	82.2	78.4	2.5K
Variational (tied weights)	81.8 ± 0.2	79.7 ± 0.1	4.7K	77.3 ± 0.2	75.0 ± 0.1	2.4K
Variational (tied weights, MC)	–	79.0 ± 0.1	–	–	74.1 ± 0.0	–
Variational (untied weights)	81.9 ± 0.2	79.7 ± 0.1	2.7K	77.9 ± 0.3	75.2 ± 0.2	1.6K
Variational (untied weights, MC)	–	$\mathbf{78.6 \pm 0.1}$	–	–	$\mathbf{73.4 \pm 0.0}$	–

Table 4.4 Single model perplexity (on test and validation sets) for the Penn Treebank language modelling task. Two model sizes are compared (a medium and a large LSTM, following [Zaremba et al., 2014]'s setup), with number of processed words per second (WPS) reported. Both dropout approximation and MC dropout are given for the test set with the Variational model. A common approach for regularisation is to reduce model complexity (necessary with the non-regularised LSTM). With the Variational models however, a significant reduction in perplexity is achieved by using larger models.

We implemented a Variational LSTM for both the medium model of [Zaremba et al., 2014] (2 layers with 650 units in each layer) as well as their large model (2 layers with 1500 units in each layer)[6]. The only changes we have made to Zaremba et al. [2014]'s setting are 1) using our proposed dropout variant instead of naive dropout, and 2) tuning weight decay (which was chosen to be zero in [Zaremba et al., 2014]). All other hyper-parameters are kept identical to [Zaremba et al., 2014]: learning rate decay was not tuned for our setting and is used following Zaremba et al. [2014]. Dropout parameters were optimised with grid search (tying the dropout probability over the embeddings together with the one over the recurrent layers, and tying the dropout probability for the inputs and outputs together as well). These are chosen to minimise validation perplexity[7].

Our results are given in table 4.4. For the variational LSTM we give results using both the tied weights model (eq. (3.23), *Variational (tied weights)*), and without weight tying (eq. (3.22), *Variational (untied weights)*). For each model we report performance using both the standard dropout approximation and using MC dropout (obtained by performing dropout at test time 1000 times, denoted *MC*). For each model we report average perplexity and standard deviation (each experiment was repeated 3 times with

[6]Full raw results and code are available online at `https://github.com/yaringal/BayesianRNN`.

[7]Optimal probabilities are 0.3 and 0.5 respectively for the large model, compared [Zaremba et al., 2014]'s 0.6 dropout probability, and 0.2 and 0.35 respectively for the medium model, compared [Zaremba et al., 2014]'s 0.5 dropout probability.

different random seeds and the results were averaged). Model training time is given in *words per second* (WPS).

It is interesting that using the dropout approximation, weight tying results in lower validation error and test error than the untied weights model. But with MC dropout the untied weights model performs much better. Validation perplexity for the large model is improved from [Zaremba et al., 2014]'s 82.2 down to 77.3 (with weight tying), or 77.9 without weight tying. Test perplexity is reduced from 78.4 down to 73.4 (with MC dropout and untied weights).

Comparing our results to the non-regularised LSTM (evaluated with early stopping, giving similar performance as the early stopping experiment in [Zaremba et al., 2014]) we see that for either model size an improvement can be obtained by using our dropout variant. Comparing the medium sized Variational model to the large one we see that a significant reduction in perplexity can be achieved by using a larger model. This cannot be done with the non-regularised LSTM, where a larger model leads to worse results. This shows that reducing the complexity of the model, a possible approach to avoid overfitting, actually leads to a worse fit when using dropout.

We also see that the tied weights model achieves very close performance to that of the untied weights one when using the dropout approximation. Assessing model run time though (on a Titan X GPU), we see that tying the weights results in a more time-efficient implementation. This is because the single matrix product is implemented as a single GPU kernel, instead of the four smaller matrix products used in the untied weights model (where four GPU kernels are called sequentially). Note though that a low level implementation should give similar run times.

4.6 Heteroscedastic uncertainty

We finish this chapter with a discussion of homoscedastic versus heteroscedastic aleatoric uncertainty. Homoscedastic regression assumes identical observation noise for every input point $\mathbf{x}$. Heteroscedastic regression, on the other hand, assumes that observation noise can vary with input $\mathbf{x}$ [Le et al., 2005]. Heteroscedastic models are useful in cases where parts of the observation space might have higher noise levels than others.

Using the developments in the previous chapter we obtained models with homoscedastic aleatoric uncertainty. This can be seen from the model definition in eq. (2.1). The likelihood in our derivations is defined as $\mathbf{y}_i \sim N(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i), \tau^{-1} I)$ with $\mathbf{f}^{\boldsymbol{\omega}}(\cdot)$ the network output, dependent on the weights random variable $\boldsymbol{\omega}$. Here our model precision τ (which

is the same as the inverse observation noise) is a constant, which has to be tuned for the data.

We can easily adapt the model to obtain data-dependent noise. This simply involves making τ into a function of the data, very much like $\mathbf{f}^{\boldsymbol{\omega}}(\cdot)$ is a function of the data. A practical way to obtain this is to tie the two functions together, splitting the top layers of a network between predictive mean $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x})$ and model precision $\mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x})$ (of course we would want to re-parametrise this to make sure $\mathbf{g}^{\boldsymbol{\omega}}(\cdot)$ is positive, and in the multivariate case to make sure the precision matrix $\mathbf{g}^{\boldsymbol{\omega}}(\cdot)$ is positive definite). Thus the new (now heteroscedastic!) model likelihood is given by $\mathbf{y}_i \sim \mathcal{N}(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i), \mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x}_i)^{-1})$. We put a prior over the weights used for the precision as well. The prior length-scale for these weights (equivalently, weight decay) controls the precision smoothness. A long prior length-scale for the precision weights would correspond to a slowly varying precision for example.

We can implement this new model by adapting the loss function of the original model, changing eq. (1.1) to:

$$\begin{aligned} E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x},\mathbf{y}) &:= \frac{1}{2}\Big(\mathbf{y} - \mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\Big)\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\Big(\mathbf{y} - \mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\Big)^T \\ &\quad - \frac{1}{2}\log\det\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x}) + \frac{D}{2}\log 2\pi \\ &= -\log\mathcal{N}(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i), \mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x}_i)^{-1}) \end{aligned}$$

with D the output dimensionality[8].

We estimate our predictive variance like before by averaging stochastic forward passes through the model, both for $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x})$ and for $\mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x})$. We use the unbiased estimator:

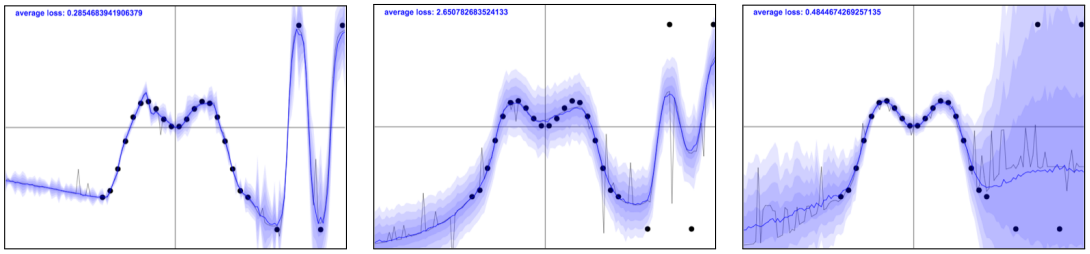
$$\begin{aligned} \widetilde{\mathrm{Var}}[\mathbf{y}^*] &:= \frac{1}{T}\sum_{t=1}^{T}\mathbf{g}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x})\mathbf{I} + \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T\mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) - \widetilde{\mathbb{E}}[\mathbf{y}^*]^T\widetilde{\mathbb{E}}[\mathbf{y}^*] \\ &\xrightarrow[T\to\infty]{} \mathrm{Var}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] \end{aligned}$$

which equals the sample variance of T stochastic forward passes through the NN plus the averaged inverse model precision.

Predictive uncertainty for the heteroscedastic model compared to the homoscedastic model[9] (both with large fixed observation noise as well as small fixed observation noise) is shown in fig. 4.13.

[8] In the multivariate output case it is convenient to assume $\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})$ to be a diagonal precision matrix, in which case the log determinant would reduce to a sum of the logs over each element of $\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})$.

[9] Code is available online at
https://github.com/yaringal/HeteroscedasticDropoutUncertainty.



(a) Homoscedastic model with small observation noise.

(b) Homoscedastic model with large observation noise.

(c) Heteroscedastic model with data-dependent observation noise.

Fig. 4.13 Homoscedastic and heteroscedastic uncertainty with Bernoulli approximating distributions (dropout). Here $N = 24$ data points are generated from the scalar function $y = x\sin(x)$ with no observation noise, and 4 points with large noise are added on the right-hand side. The homoscedastic models use $l^2 = 10$ and $p = 0.005$, with $\tau^{-1} = 0$ for sub-figure 4.13a and $\tau = 1$ for sub-figure 4.13b. The heteroscedastic model (sub-figure 4.13c) uses $l^2 = 0.1$ and $p = 0.05$.

This experiment concludes our uncertainty quality assessment. We next examine real-world applications relying on the developments above.

Chapter 5

Applications

In previous chapters we linked stochastic regularisation techniques (SRTs) to approximate inference in Bayesian neural networks (NNs), and studied the resulting model uncertainty for popular SRTs such as dropout. We have yet to give any real-world *applications* stemming from this link though, leaving it somewhat in the realms of "theoretical work". But a theory is worth very little if we can't use it to obtain new tools, or shed light on existing ones. In this chapter we will survey recent literature making use of the tools developed in the previous chapters in fields ranging from language processing to computer vision to biomedical domains. This is followed by more use cases I have worked on recently with the help of others. We will see how model uncertainty can be used to choose what data to learn from (joint work with Riashat Islam as part of his Master's project). Switching to applications in deep reinforcement learning, we will see how model uncertainty can help exploration. This is then followed by the development of a data efficient framework in deep reinforcement learning (joint work with Rowan McAllister and Carl Rasmussen). A study of the implications of this work on our *understanding* of existing tools will be given in the next chapter.

5.1 Recent literature

We begin with a quick literature survey of applications making use of the tools brought above (and presented previously in [Gal, 2015; Gal and Ghahramani, 2015a,b,c,d, 2016a,b,c]). This survey offers support to our main claim of the tools developed being *practical*, and can be used as a starting point for further applications on related tasks. The key contributions of each work will be given, and research reproducing our results in previous chapters will be highlighted. We discuss research in three application areas: language, biology and medicine, and computer vision.

5.1.1 Language applications

Working on machine translation as part of the WMT16 shared task, the Edinburgh team [Sennrich et al., 2016] has made use of deep network architectures and specifically recurrent NNs. Machine translation is a challenging task where we wish to build a system to translate from one language to another, given only sentences in one language and their translations in the other. Sennrich et al. [2016] found that using naive dropout in RNNs their model would overfit, and reported that with the Bayesian variant of dropout proposed in §3.4.2 the model did not. Sennrich et al. [2016]'s model achieved state-of-the-art results on Czech–English, Romanian–English, and German–English language pairs, beating the other 31 submissions in the shared task [Bojar et al., 2016].

It is worth mentioning the work of Press and Wolf [2016] as well. Press and Wolf [2016], while working on language modelling, have reproduced our language modelling results presented in §3.4.2. Our language modelling results were further reproduced in the work of Léonard et al. [2015]—an open source Torch library for RNNs. Our dropout variant for RNNs is included in this library, as well as in the TensorFlow and Keras libraries [Abadi et al., 2015; Chollet, 2015] as the default RNN dropout implementation.

5.1.2 Medical diagnostics and bioinformatics

In medical diagnostics Yang et al. [2016] have worked on image registration. This task involves "stitching" together images obtained from brain scans for example. This is a challenging task as images might be taken at different points in time, and a patient's movement due to breathing for example could mean that the images would not be aligned. As a result there is an inherent uncertainty in the reconstruction process. Yang et al. [2016], relying on recent CNN tools, proposed a model that improved on existing techniques. Relying on our tools developed in section §3.4.1, Yang et al. [2016] showed how model uncertainty can be obtained in their setting, and empirically evaluated their model. For example, stitching MRI brain scans, they showed how shape changes in the anterior edge of the ventricle and the posterior brain cortex (changes due to blood flow at different points in time for example) led to high uncertainty in those regions.

As another example, Angermueller and Stegle [2015] attempted to predict methylation rate in embryonic stem cells. In DNA methylation, methyl groups are added to nucleotides and affect the transcription of genes. This process controls gene regulation, and affects the development of disease for example. Angermueller and Stegle [2015] fitted a neural network to their data, and showed an improvement over standard techniques in the field.

Using the techniques developed in §3.3 they showed that model uncertainty increased in genomic contexts which are hard to predict (e.g. LMR or H3K27me3).

5.1.3 Computer vision and autonomous driving

Numerous applications of the methods involving model uncertainty for image data (introduced in §3.3–§3.4) have been developed in recent literature [Bulò et al., 2016; Kendall and Cipolla, 2016], with specific attention paid to model uncertainty in image segmentation [Furnari et al., 2016; Kampffmeyer et al., 2016; Kendall et al., 2015]. This might be because model uncertainty is hard to obtain for image data, and with the tools developed in §3.4.1 obtaining this information is easier than with previous tools. In this section three main applications will be surveyed: new tools [Bulò et al., 2016], computer vision systems for autonomous driving [Kendall and Cipolla, 2016; Kendall et al., 2015], and image segmentation [Furnari et al., 2016; Kampffmeyer et al., 2016; Kendall et al., 2015].

Bulò et al. [2016] developed tools to efficiently approximate MC dropout. Even though MC dropout results in lower test RMSE, it comes with a price of prolonged test time. This is because we need to evaluate the network stochastically multiple times and average the results. Instead, Bulò et al. [2016] fit a secondary model to predict (with a single forward pass) the MC output of the primary model. Bulò et al. [2016] further reproduced our experimental results in §3.4.1 with the NiN model on CIFAR10. However compared to our results with lenet-all (applying MC dropout after every convolution layer on MNIST) where we observed a big improvement for MC dropout over standard dropout, Bulò et al. [2016]'s results suggested similar performance for both techniques.

Working on autonomous driving applications, Kendall and Cipolla [2016] developed tools for the localisation of a car given a photo taken from a front-facing camera installed in the vehicle. This is an important application since GPS systems cannot always be trusted and give low-accuracy localisation. When used in autonomous driving, this localisation information can be used to get information such as the distance of the car from the sidewalk for example. Kendall and Cipolla [2016] assessed model uncertainty following the tools developed in the previous chapters, and found model uncertainty to correlate to positional error. They found that test photos with strong occlusion from vehicles, pedestrians, or other objects resulted in high uncertainty, with model uncertainty showing a linear trend increasing with the distance from the training set (after calibration). In further work Kendall et al. [2015] developed models for scene understanding, mapping objects in the photos taken by the car to labels such as "pedestrian" or "cyclist" on

a pixel level. Extracting model uncertainty they showed that object edges have lower model confidence for example.

One last application is image segmentation. Kampffmeyer et al. [2016] for example looked at semantic segmentation in urban remote sensing images. They used CNNs with the techniques described in §3.4.1 and analysed the resulting segmentation uncertainty. Kampffmeyer et al. [2016] computed the standard deviation over the softmax outputs of 10 Monte Carlo samples. They then averaged the standard deviation over all classes to obtain a single scalar value[1]. They found that model uncertainty increased at object boundaries and in regions where the model misclassified, validating the hypothesis that pixels with high model confidence are classified correctly more often. Kampffmeyer et al. [2016] further gave precision-recall plots showing that when pixels with higher uncertainty were removed, classification accuracy increased. They concluded that uncertainty maps are a good measure for the pixel-wise uncertainty of the segmented remote sensing images.

I next give a more in-depth review of several applications developed in collaboration with others.

5.2 Active learning with image data

This has been joint work with Riashat Islam as part of his Master's project.

A big challenge in many applications is obtaining labelled data. This can be a long and laborious process, which often makes the development of an automated system uneconomical. A framework where a system could learn from small amounts of data, and choose by itself what data it would like the user to label, would make machine learning applicable to a wider class of problems. Such a framework is referred to as *active learning* [Cohn et al., 1996]. In this setting a model is trained on a small amount of data (the initial training set), and an *acquisition function* (often based on the model's uncertainty) decides what data points to ask an external *oracle* for a label. The acquisition function selects several points from a *pool* of data points, with the pool points lying outside of the training set. An oracle (often a human expert) labels the selected data points, these are added to the training set, and a new model is trained on the updated training set. This process is then repeated, with the training set increasing in size over time.

Even though active learning forms an important pillar of machine learning, deep learning tools are not prevalent within it. Deep learning poses several difficulties when

[1]Note that this is not necessarily the best uncertainty summary for classification, and the techniques discussed in §3.3.1 might give more sensible results. We will get back to this problem in the next section.

used in an active learning setting. First, we have to handle small amounts of data. Second, many acquisition functions rely on model uncertainty. Luckily Bayesian approaches to deep learning make this task more practical. Even more exciting, taking advantage of specialised models such as Bayesian convolutional neural networks, we can perform active learning with *image data*. Here we will develop an active learning framework for image data, a task which has been extremely challenging so far with very sparse existing literature [Holub et al., 2008; Joshi et al., 2009; Li and Guo, 2013; Zhu et al., 2003].

To perform active learning with image data we make use of the Bayesian CNNs developed in section §3.4.1. These work well with small amounts of data (as seen in section §4.4.1), and possess uncertainty information that can be used with existing acquisition functions. With regression, acquisition functions can be formed by looking at the sample variance. But CNNs are often used in the context of classification, and in this setting we shall make use of the uncertainty measures discussed in §3.3.1. More specifically we will analyse acquisition functions based on a random acquisition (referred to as *Random*), functions maximising the predictive entropy (referred to as *Max Entropy*, [Shannon, 1948]), maximising the *variation ratios* [Freeman, 1965], and maximising the mutual information between the predictions and the model posterior (referred to as *BALD*, "Bayesian active learning by disagreement", [Houlsby et al., 2011]).

Bayesian CNN approximate inference can be done using various approximating distributions. We experiment with two approximating distributions: a product of Bernoullis (implemented as dropout before each weight layer, referred to as *Dropout*), and a delta approximating distribution (implemented as a standard CNN with a deterministic softmax probability vector, referred to as *Softmax*). All acquisition functions are assessed with the same model structure: convolution-relu-convolution-relu-max pooling-dropout-dense-relu-dropout-dense-softmax, with 32 convolution kernels, 4x4 kernel size, 2x2 pooling, dense layer with 128 units, and dropout probabilities 0.25 and 0.5. All models are trained on the MNIST dataset [LeCun and Cortes, 1998] with a (random but balanced) initial training set of 20 data points, a validation set of 5K points, the standard test set of 10K points, and the rest of the points used as a pool set. All models were assessed after each acquisition using the dropout approximation, with the Softmax models using the dropout approximation to evaluate model output for acquisition, and the Dropout models using MC dropout to evaluate model output for acquisition. We repeated the acquisition process 100 times, each time acquiring the 10 points that maximised the

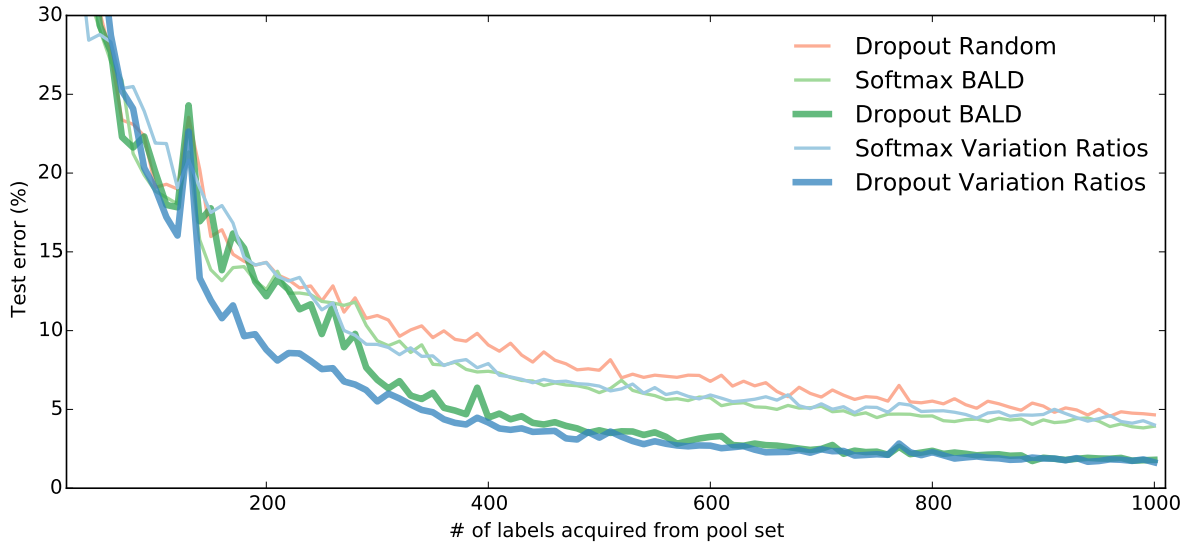


Fig. 5.1 **Test error on MNIST as a function of number of labels acquired from the pool set.** Two acquisition functions (*BALD* and *Variation Ratios*) evaluated with two approximating distributions—delta (*Softmax*) and Bernoulli (*Dropout*)—are compared to a *random* acquisition function.

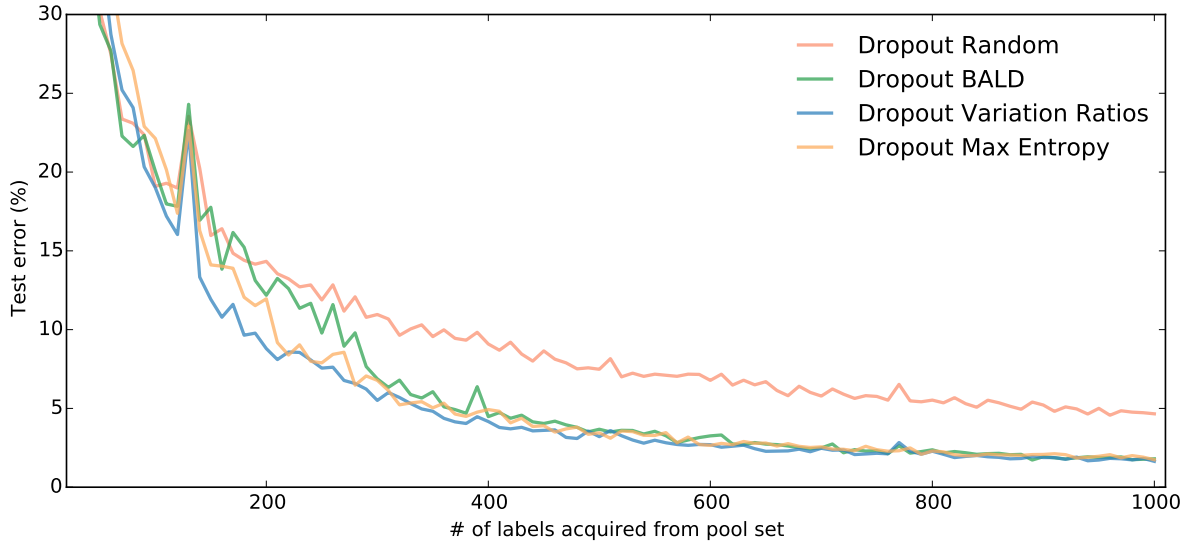


Fig. 5.2 **Test error on MNIST as a function of number of labels acquired from the pool set.** Four acquisition functions are shown (*Random*, *BALD*, *Variation Ratios*, and *Max Entropy*) evaluated with a Bernoulli approximating distribution (*Dropout*).

	Random	Max Entropy	Variation Ratios	BALD
Softmax	4.57	2.18	4.02	3.83
Dropout	NA	1.74	1.64	1.72

Table 5.1 **Model error (%)** on MNIST with 1000 training points for different acquisition functions.

acquisition function over the pool set. Each experiment was repeated 3 times and the results averaged[2].

A comparison of the Dropout acquisition functions to the Softmax acquisition functions for Variation Ratios and BALD is given in fig. 5.1. Both acquisition functions (*BALD* and *Variation Ratios*) were evaluated with both approximating distributions and compared to a random acquisition function where new points are sampled uniformly at random from within the pool set. The Dropout acquisition functions, propagating uncertainty throughout the model, attain a smaller error early on, and converge to a lower error rate overall. This demonstrates that the uncertainty propagated throughout the Bayesian models has a significant effect on the acquisition functions' measure of model confidence. Variation Ratios seems to obtain lower test error faster than BALD for acquisition points ranging between 150 and 400, but BALD outperforms Variation Ratios on the range 0 to 150 (not shown).

	Random	Max Entropy	Variation Ratios	BALD
Softmax	90	53	76	68
Dropout	NA	35	32	36

Table 5.2 **Number of acquisition steps** to get to model error of 5% on MNIST, for different acquisition functions.

We further compared the acquisition functions Random, Variation Ratios, and BALD to an acquisition function based on the predictive entropy (Max Entropy). We found Random to under-perform compared to BALD, Variation Ratios, and Max Entropy (figure 5.2). The converged test error for all acquisition functions after the acquisition of 1000 training points is given in table 5.1.

Lastly, in table 5.2 we give the number of acquisition steps needed to get to test error of 5%. As can be seen, Dropout Max Entropy, Dropout Variation Ratios and Dropout

[2]The code for these experiments is available at https://github.com/Riashat/Active-Learning-Bayesian-Convolutional-Neural-Networks/tree/master/ConvNets/FINAL_Averaged_Experiments/Final_Experiments_Run.

BALD attain test error of 5% within a much smaller number of acquisitions than their Softmax equivalents.

We next switch to the use of model uncertainty in the setting of reinforcement learning, on a task similar to that used in [Mnih et al., 2015].

5.3 Exploration in deep reinforcement learning

In reinforcement learning an agent receives various rewards from different states, and its aim is to maximise its expected reward over time. The agent tries to learn to avoid transitioning into states with low rewards, and to pick actions that lead to better states instead. Uncertainty is of great importance in this task—with uncertainty information an agent can decide when to exploit rewards it knows of, and when to explore its environment.

Recent advances in RL have made use of NNs to estimate agents' Q-value functions (referred to as Q-networks), a function that estimates the quality of different actions an agent can take at different states. This has led to impressive results on Atari game simulations, where agents superseded human performance on a variety of games [Mnih et al., 2015]. Epsilon greedy search was used in this setting, where the agent selects the best action following its current Q-function estimation with some probability, and explores otherwise. With our uncertainty estimates given by a dropout Q-network we can use techniques such as Thompson sampling [Thompson, 1933] to converge faster than with epsilon greedy while avoiding over-fitting.

We use code by Karpathy et al. [2014–2015] that replicated the results by Mnih et al. [2015] with a simpler 2D setting. We simulate an agent in a 2D world with 9 eyes pointing in different angles ahead (depicted in fig. 5.3). Each eye can sense a single pixel intensity of 3 colours. The agent navigates by using one of 5 actions controlling two motors at its base. An action turns the motors at different angles and different speeds. The environment consists of red circles which give the agent a positive reward for reaching, and green circles which result in a negative reward. The agent is further rewarded for not looking at (white) walls, and for walking in a straight line[3].

We trained the original model, and an additional model with dropout with probability 0.1 applied before every weight layer. Note that both agents use the same network structure in this experiment for comparison purposes. In a real world scenario using dropout we would use a larger model (as the original model was intentially selected to

[3]The code for this experiment is given at
https://github.com/yaringal/DropoutUncertaintyDemos.

be small to avoid over-fitting). To make use of the dropout Q-network's uncertainty estimates, we use Thompson sampling instead of epsilon greedy. In effect this means that we perform a single stochastic forward pass through the network every time we need to take an action. In replay, we perform a single stochastic forward pass and then back-propagate with the sampled Bernoulli random variables.

In fig. 5.4 we show a plot of the average reward as a function of learning time (on a log scale) obtained by both the original implementation (in green) and our approach (in blue). Not plotted is the burn-in intervals of 25 batches (random moves). Thompson sampling gets reward larger than 1 within 25 batches from burn-in. Epsilon greedy takes 175 batches to achieve the same performance[4].

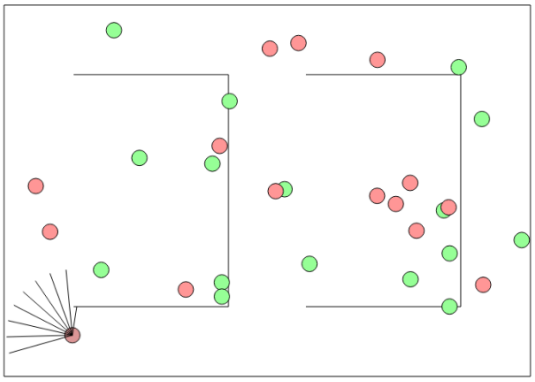


Fig. 5.3 Depiction of the reinforcement learning problem used in the experiments. The agent is in the lower left part of the maze, facing north-west.

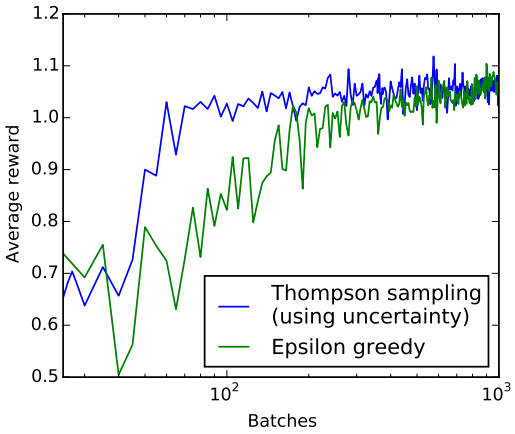


Fig. 5.4 **Log plot of average reward** obtained by both epsilon greedy (in green) and our approach (in blue), as a function of the number of batches.

Remark. We presented these results initially in [Gal and Ghahramani, 2015c]. Stadie et al. [2015] have reproduced our experiments while comparing to their model-based exploration technique (using an auto-encoder to encode the states). They compared to our model-free Thompson sampling, the epsilon greedy based DQN [Mnih et al., 2015], as well as to Boltzmann exploration (which samples an action from the softmax output, annealing the probabilities to smoothly transition from

[4] It is interesting to note that our approach seems to stop improving after 1K batches. This is because we are still sampling random moves, whereas epsilon greedy only exploits at this stage.

exploration to exploitation: $a \sim \text{Categorical}(\exp(Q(s,a)/\tau)/(\sum_a \exp(Q(s,a)/\tau))$ with $\tau \to 0$).
Stadie et al. [2015] experimented with the Atari framework, in which they chose several games that existing methods found challenging, and where human performance superseded that of DQN. They found that after 100 epochs their model-based approach outperformed existing model-free techniques. Comparing DQN to our Thompson technique, they found DQN to get higher scores than Thompson on 6 games, whereas Thompson got higher scores than DQN on 7 games. Looking at the area under the curve (AUC) for the first 100 epochs—a measure of how quickly an agent learns—Thompson beat DQN on 8 games, whereas DQN beat Thompson on 5 games, emphasising the data efficiency of Thompson sampling compared to epsilon greedy. Thompson achieved best score out of all approaches on one of the games. It is interesting that Boltzmann exploration outperformed Thompson sampling quite considerably: on 4 games Thompson got better score than Boltzmann, whereas Boltzmann got better results on 8 games. Assessing AUC, Thompson beat Boltzmann on 3 games, with Boltzmann beating Thompson on 9 games. This might be because Thompson sampling still explores after 100 epochs, whereas Boltzmann exploration exploits by then. It would be interesting to combine the two techniques, and sample from an annealed probability obtained from a Bayesian NN.

5.4 Data efficiency in deep reinforcement learning

This has been joint work with Rowan McAllister and Carl Rasmussen [Gal, McAllister, and Rasmussen, 2016].

Compared to the previous model-free approach, *PILCO* is a *model-based* policy search RL algorithm [Deisenroth and Rasmussen, 2011]. PILCO achieved unprecedented data-efficiency of several control benchmarks including the cartpole swing-up task. But PILCO is limited by its use of a Gaussian process (GP) to model its distribution over model dynamics. GPs are hard to scale to large quantities of data and high dimensional data [Gal et al., 2014]. Instead, we might want to replace the GP with a Bayesian NN. But this task poses several interesting difficulties. First, we have to handle *small data*, and neural networks are notoriously known for their tendency to overfit. Furthermore, we must retain PILCO's ability to capture 1) dynamics model output uncertainty and 2)

input uncertainty. Output uncertainty can be captured with a Bayesian neural network (BNN), but end-to-end inference poses a challenge. Input uncertainty in PILCO is obtained by analytically propagating a state distribution through the dynamics model. But this can neither be done analytically with NNs nor with BNNs. Our solution to handling output uncertainty relies on dropout as a Bayesian approximation to the dynamics model posterior. This allows us to use techniques proven to work well in the field, while following their probabilistic Bayesian interpretation. Input uncertainty in the dynamics model is captured using particle techniques. To do this we solve the difficulties encountered by McHutchon [2014] when attempting this particle technique with PILCO in the past. Interestingly, unlike PILCO, our approach allows us to sample dynamics functions, required for accurate variance estimates of future state distributions.

Our approach has several benefits compared to the existing PILCO framework. First, as we require lower time complexity (linear in trials and observation space dimensionality), we can scale our algorithm well to tasks that necessitate more trials for learning. Second, unlike PILCO we can sample dynamics function realisations. The use of a NN dynamics model comes at a price though, where we need to use a slightly higher number of trials than PILCO. Lastly, our model can be seen as a Bayesian approach to performing data efficient deep RL. In section §5.4.3 we compare our approach to that of recent deep RL algorithms [Gu et al., 2016; Lillicrap et al., 2015], showing orders of magnitude improvement in data efficiency on these.

5.4.1 PILCO

PILCO is summarised by Algorithm 3. A policy π's functional form is chosen by the user (step 1), whose parameters ψ are initialised randomly (step 2). Thereafter

Algorithm 3 PILCO

1: *Define* policy's functional form: $\pi : z_t \times \psi \to u_t$.
2: *Initialise* policy parameters ψ randomly.
3: **repeat**
4: *Execute* system, record data.
5: *Learn* dynamics model.
6: *Predict* system trajectories from $p(X_0)$ to $p(X_T)$.
7: *Evaluate* policy:
$J(\psi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_X[\text{cost}(X_t)|\psi]$.
8: *Optimise* policy:
$\psi \leftarrow \arg\min_{\psi} J(\psi)$.
9: **until** policy parameters ψ converge

PILCO executes the current policy from an initial state (sampled from initial distribution $p(X_0)$) until the time horizon T (defined as one trial, step 4). Observed transitions are recorded, and appended to the total training data. Given the additional training data, the dynamics model is re-trained (step 5). Using its probabilistic transition model, PILCO then analytically predicts state distributions from an initial state distribution $p(X_0)$ to $p(X_1)$ etc. until time horizon $p(X_T)$ making a joint Gaussian assumption (step 6). Prediction of future state distribution follows the generative model seen in Figure 5.5, where each system-state X_t defines an action U_t according to policy π, which determines the new state X_{t+1} according to the dynamics f. I.e. $X_{t+1} = f(X_t, U_t)$, where we train a GP model of f given all previous observed transition tuples $\{X_t, U_t, X_{t+1}\}$. Given the multi-state distribution $p(\{X_0, ..., X_T\})$, the expected cost $\mathbb{E}_X[\text{cost}(X_t)]$ is computed for each state distribution using a user-supplied cost function. The sum of expected costs is our minimisation objective J (step 7). Gradient information is also computed w.r.t. policy parameters $\mathrm{d}J/\mathrm{d}\psi$. Finally, the objective J is optimised using gradient decent according to $\mathrm{d}J/\mathrm{d}\psi$ (step 8). The algorithm then loops back to step 4 and executes the newly-optimised policy, which is locally optimal given all the data observed thus far.

PILCO's data-efficiency success can be attributed to its *probabilistic* dynamics model. Probabilistic models help avoid model bias—a problem that arises from selecting only a single dynamics model $\hat{f}$ from a large possible set and assuming that $\hat{f}$ is the correct model with certainty [Deisenroth et al., 2015]. Whilst such approaches can provide accurate short term state predictions e.g. $p(X_1)$, their longterm predictions (e.g. $p(X_T)$) are inaccurate due to the compounding effects of T-many prediction errors from $\hat{f}$. Since inaccurate predictions of $p(X_T)$ are made with high-confidence, changes in policy parameters ψ are (falsely) predicted to have significant effect on the expected cost at time T. Since optimising total expected cost J must balance the expected costs of states, including $p(X_1)$ and $p(X_T)$, the optimisation will compromise on the cost of $p(X_1)$ based on perceived cost of $p(X_T)$—even though the prediction $p(X_T)$ is effectively random noise with $p(X_T)$ having a broad distribution almost invariant to policy π. Such undesirable behaviour hampers data efficiency. Optimising data-efficiency exacerbates the negative effects of model bias even further, since the smaller the data, the larger the set of plausible models that can describe that data. PILCO uses probabilistic models to avoid model bias by considering *all* plausible dynamics models in prediction of all future states. In cases as the above, PILCO optimises the policy based only on the states X_t it can predict well.

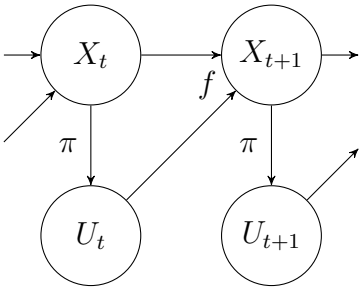


Fig. 5.5 **Prediction model of system trajectories** (step 6 in Algorithm 3). The system state X_t generates action U_t according to policy π, both of which result in a new state X_{t+1} as predicted by dynamics model f (a model trained given all previously observed X and U).

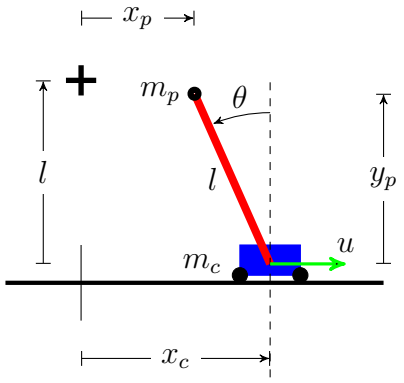


Fig. 5.6 **The cartpole swing-up task.** A pendulum of length l is attached to a cart by a frictionless pivot. The system begins with cart at position $x_c = 0$ and pendulum hanging down: $\theta = \pi$. The goal is to accelerate the cart by applying horizontal force u_t at each timestep t to invert then stabilise the pendulum's endpoint at the goal (black cross).

5.4.2 Deep PILCO

We now describe our method—*Deep PILCO*—for data-efficient deep RL. Our method is similar to PILCO: both methods follow Algorithm 3. The main difference of Deep PILCO is its dynamics model. PILCO uses a Gaussian process which can model the dynamics' output uncertainty, but cannot scale to high dimensional observation spaces. In contrast, Deep PILCO uses a deep neural network capable of scaling to high dimensional observations spaces. Like PILCO, our policy-search algorithm alternates between fitting a dynamics model to observed transitions data, evaluating the policy using dynamics model predictions of future states and costs, and then improving the policy.

Replacing PILCO's GP with a deep network is a surprisingly complicated endeavour though, as we wish our dynamics model to maintain its probabilistic nature, capturing 1) output uncertainty, and 2) input uncertainty.

Output uncertainty

First, we require output uncertainty from our dynamics model, critical to PILCO's data-efficiency. Simple NN models cannot express output model uncertainty, and thus cannot capture our ignorance of the latent system dynamics. To solve this we use the developments presented in §3.4.2. We interpret dropout as a variational Bayesian

approximation, and use the uncertainty in the weights to induce prediction uncertainty. This amounts to the regular dropout procedure only with dropout also applied at test time, giving us output uncertainty from our dynamics model.

This approach also offers insights into the use of NNs with small data. Eq. (3.14) for example shows that the network's weight decay can be parametrised as a function of dataset size, dropout probability, and observation noise. Together with adaptive learning-rate optimisation techniques, the number of parameters requiring tuning becomes negligible.

Input uncertainty

A second difficulty with NN dynamics models is handling *input* uncertainty. To plan under dynamics uncertainty, PILCO analytically propagates state distributions through the dynamics model (step 6 in Algorithm 3, depicted in Figure 5.5). To do so, the dynamics model must pass uncertain dynamics outputs from a given time step as uncertain input into the dynamics model in the next time step. This handling of input uncertainty cannot be done analytically with NNs, as is done with Gaussian processes in PILCO.

To feed a distribution into the dynamics model, we resort to particle methods (Algorithm 4). This involves sampling a set of particles from the input distribution (step 2 in Algorithm 4), passing these particles through the BNN dynamics model (step 8 in Algorithm 4), which yields an output distribution of particles.

Algorithm 4 Step 6 of Algorithm 3: *Predict* system trajectories from $p(X_0)$ to $p(X_T)$

1: *Define* time horizon T.
2: *Initialise* set of K particles $x_0^k \sim P(X_0)$.
3: **for** $k = 1$ to K **do**
4: Sample BNN dynamics model weights W^k.
5: **end for**
6: **for** time $t = 1$ to T **do**
7: **for** each particle x_t^1 to x_t^K **do**
8: Evaluate BNN with weights W^k and input particle x_t^k, obtain output y_t^k.
9: **end for**
10: Calculate mean μ_t and standard deviation σ_t^2 of $\{y_t^1, ..., y_t^K\}$.
11: Sample set of K particles $x_{t+1}^k \sim \mathcal{N}(\mu_t, \sigma_t^2)$.
12: **end for**

This approach was attempted unsuccessfully in the past with PILCO [McHutchon, 2014]. McHutchon [2014] encountered several problems optimising the policy with particle methods, the main problem being the abundance of local optima in the optimisation surface, impeding his BFGS optimisation method. McHutchon [2014] suggested that this

might be due to the finite number of particles used and their deterministic optimisation. To avoid these issues, we randomly re-sample a new set of particles at each optimisation step, giving us an unbiased estimator for the objective (step 7 in Algorithm 3). We then use the stochastic optimisation procedure Adam [Kingma and Ba, 2014] instead of BFGS.

We found that fitting a Gaussian distribution to the output state distribution at each time step, as PILCO does, is of crucial importance (steps 10-11 in Algorithm 4). This moment matching avoids multi-modality in the dynamics model. Fitting a multi-modal distribution with a (wide) Gaussian causes the objective to average over the many high-cost states the Gaussian spans [Deisenroth et al., 2015]. By forcing a unimodal fit, the algorithm penalises policies that cause the predictive states to bifurcate, often a precursor to a loss of control. This can alternatively be seen as smoothing the gradients of the expected cost when bifurcation happens, simplifying controller optimisation. We hypothesised this to be an important modelling choice done in PILCO.

Sampling functions from the dynamics model

Unlike PILCO, our approach allows sampling individual functions from the dynamics model and following a single function throughout an entire trial. This is because a repeated application of the BNN dynamics model above can be seen as a simple Bayesian recurrent neural network (RNN, where an input is only given at the first time step). Approximate inference in the Bayesian RNN is done by sampling function weights once for the dynamics model, and using the same weights at all time steps (steps 4 and 8 in Algorithm 4). With dropout, this is done by sampling and fixing the dropout mask for all time steps during the rollout (§3.4.2). PILCO does not consider such temporal correlation in model uncertainty between successive state transitions, which results in PILCO underestimating state uncertainty at future time steps [Deisenroth et al., 2015].

Another consequence of viewing our dynamics model as a Bayesian RNN is that the model could be easily extended to more interesting RNNs such as Bayesian LSTMs, capturing long-term dependencies between states. This is important for non-Markovian system dynamics, which can arise with observation noise for example. Here we restrict the model to Markovian system dynamics, where a simple Bayesian recurrent neural network model suffices to predict a single output state given a single input state.

5.4.3 Experiment

To compare our technique to PILCO we experimented with the cartpole swing-up task—a standard benchmark for nonlinear control. Lillicrap et al. [2015] and Gu et al. [2016]

use this benchmark as well, assessing their approach on the four-dimensional state-space task.

Figure 5.7 shows the average cost of 40 random runs (with two standard errors denoted by two shades for each plot). Deep PILCO successfully balances the pendulum in most experiment repetitions within 20-25 trials, compared to PILCO's 6-7 trials. Figure 5.8 shows the progression of deep PILCO's fitting as more data is collected.

Our model can be seen as a Bayesian approach to data efficient deep RL. We compare to recent deep RL algorithms (Lillicrap et al. [2015] and Gu et al. [2016]). Lillicrap et al. [2015] use an actor-critic model-free algorithm based on deterministic policy gradients. Gu et al. [2016] train a continuous version of model-free deep Q-learning using imagined trials generated with a learnt model. For their *low dimensional* cartpole swing-up task Lillicrap et al. [2015] require approximately 2.5×10^5 steps to achieve good results. This is equivalent to approximately 2.5×10^3 trials of data, based on Figure 2 in [Lillicrap et al., 2015] (note that [Lillicrap et al., 2015] used time horizon $T = 2$s and time discretisation $\Delta t = 0.02$s, slightly different from ours; they also normalised their reward, which does not allow us to compare to their converged reward directly). Gu et al. [2016] require approximately 400 trials for model convergence. These two results are denoted with vertical lines in Figure 5.7 (as the respective papers do not provide high resolution trial-cost plots).

Lastly, we report model run time for both Deep PILCO as well as PILCO. Deep PILCO can leverage GPU architecture, and took 5.85 hours to run for the first 40 iterations. This is with constant time complexity w.r.t. the number of trials, and linear time complexity in input dimensionality Q and output dimensionality D. PILCO (running on the CPU) took 20.7 hours for the 40 trials, and scales with $\mathcal{O}(N^2Q^2D^2)$ time complexity, with N number of trials. With more trials PILCO will become much slower to run. Consequently, PILCO is unsuited for tasks requiring a large number of trials or high-dimensional state tasks.

We surveyed research in three application areas: language, biology and medicine, and computer vision, and gave example applications in RL and active learning, lending support to our main claim of the tools developed in chapter 3 being *practical*. We next go in depth into some more theoretical questions analysing the developments of the previous chapters.

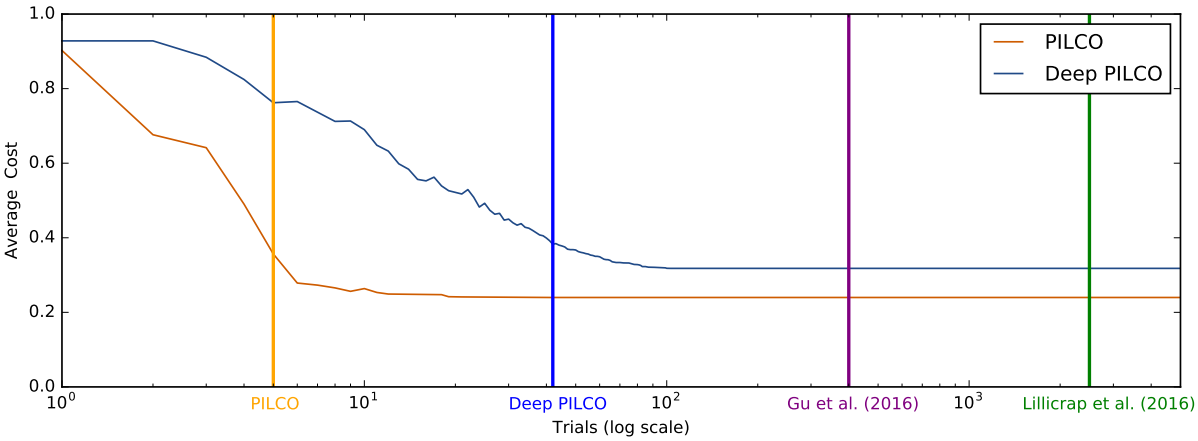
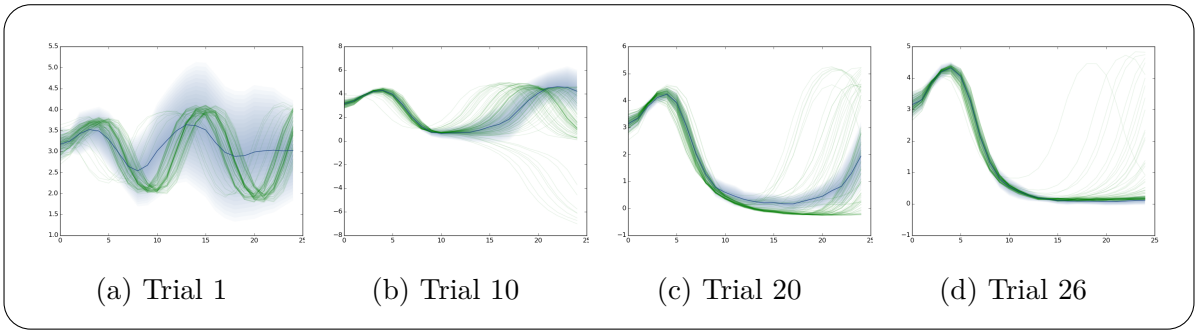


Fig. 5.7 Average cost per trial with standard error (on *log scale*) for PILCO and Deep PILCO on the cartpole swing-up task, averaging 40 experiment repetitions for both models. Vertical lines show estimates of number of trials required for model convergence (successfully balancing the pendulum with most experiment repetitions) for PILCO (yellow, 5 trials), Deep PILCO (blue, 42 trials), Gu et al. [2016] (purple, ~ 400 trials) and Lillicrap et al. [2015] (green, $\sim 2,500$ trials).



(a) Trial 1 (b) Trial 10 (c) Trial 20 (d) Trial 26

Fig. 5.8 **Progression of model fitting and controller optimisation as more trials of data are collected.** Each x-axis is timestep t, and each y-axis is the pendulum angle θ in radians (see Figure 5.6). The goal is to swing the pendulum up such that $\text{mod}(\theta, 2\pi) \approx 0$. The green lines are samples from the ground truth dynamics. The blue distribution is our Gaussian-fitted predictive distribution of states at each timestep. **(a)** After the first trial the model fit (blue) does not yet have enough data to accurately capture the true dynamics (green). Thus the policy performs poorly: the pendulum remains downwards swinging between 2π and 4π. **(b)** After 10 trials, the model fit (blue) predicts very well for the fist 13 time steps before separating from the true rollouts (green). The controller has stabilised the pendulum at 0π for about 10 time steps (1 second). **(c)** After 20 trials the model fit and policy are slightly improved. **(d)** From trial 26 onward, the dynamics model successfully captured the true dynamics and the policy successfully stabilises the pendulum upright at 0π radians most trials.

Chapter 6

Deep Insights

Until now we have mostly studied the proposed approximate inference techniques empirically. In this chapter we turn to a more theoretical analysis, and concentrate mostly on the case of dropout, as it seems to be the most widely used among the various stochastic regularisation techniques. We begin by suggesting practical considerations for getting good uncertainty estimates, followed by a review of what affects predictive uncertainty characteristics. We then offer an analytical analysis in the linear case, answering many higher-level questions about the behaviour of the inference, and analyse dropout's evidence lower bound (ELBO) correlation with test log likelihood. We continue by discussing various alternative priors to the standard Gaussian prior: we discuss the properties of a discrete prior, and (approximately) derive the optimal variational posterior with a spike and slab prior. The latter, quite surprisingly, turns out to be closely related to the structure of the dropout approximating distribution. We finish the chapter with a more philosophical discussion, examining the different types of uncertainty available to us from the dropout neural networks, and suggest a new tool to optimise the dropout probabilities under the variational setting.

6.1 Practical considerations for getting good uncertainty estimates

I will suggest some "tricks of the trade" to get good predictive uncertainty estimates. First, it seems that "over-parametrised" models result in better uncertainty estimates than smaller models. Models with a large number of parameters can capture a larger class of functions, leading to more ways of explaining the data, and as a result larger uncertainty estimates further from the data. Similar behaviour was noted in [Neal, 1995] with regard

to model size. In the dropout case, this conforms with the observation that better RMSE can be obtained when a large number of parameters is used (larger than when dropout is not used as discussed in §4.5). The dropout probability is important as well, with larger models requiring a larger dropout probability: varying the dropout probability p (through either grid-search or Bayesian optimisation) we have that large models (large K) push p towards 0.5, since the weight of the entropy w.r.t. p (eq. (A.1)) is scaled by K. For a fixed model size K, smaller probabilities p result in decreasing predictive uncertainty. Further, short model length-scale results in more erratic functions drawn from the posterior hence higher uncertainty values. Intuitively, high model precision (large τ) and large amounts of data (large N) give the expected log likelihood a higher weight than the prior KL, resulting in models that can fit the data well but might overfit (this can be seen in eq. (3.14)). On the other hand, long prior length-scale (large l) gives the prior KL a higher weight than the expected log likelihood, resulting in heavily regularised models that might not fit the data as well (this can be seen in eq. (6.6) below). The prior length-scale, model precision, and dropout probability can be optimised using Bayesian optimisation and cross validation over test log likelihood (as was done in §4.3). Lastly, it seems that model structure affects predictive uncertainty considerably. Many existing models were designed and developed in order to obtain good RMSE, but the same model structure might not be ideal to get good uncertainty estimates. Adapting model structure to result in good uncertainty estimates as well as RMSE might be helpful in improving test log likelihood. The reason for this is discussed next.

6.2 What determines what our uncertainty looks like?

Predictive uncertainty is determined through a combination of model structure, model prior, and approximating distribution. This is similar to Gaussian processes (GPs), where the choice of covariance function is determined by our prior belief as to the properties of the predictive uncertainty we expect to observe. Choosing a squared exponential covariance function for example corresponds to a prior belief that predictive uncertainty increases far away from the data, and a choice of a "neural network" covariance function corresponds to a prior belief that the predictive mean does not collapse to zero far away from the data [Rasmussen and Williams, 2006]. Bayesian NNs can be seen as Gaussian process approximations [Gal and Turner, 2015; Neal, 1995; Williams, 1997], where the GP's covariance function is determined by the Bayesian NN non-linearity and prior.

Since we are approximating the Bayesian NN posterior with an approximating distribution $q(\cdot)$, this also affects our resulting predictive uncertainty. In section §4.2 we saw that the resulting predictive uncertainty depends heavily on the non-linearity and model prior (through the prior length-scale l). It seemed to be less affected by the approximating distribution $q(\cdot)$, with dropout, multiplicative Gaussian noise, and others resulting in similar predictive uncertainty.

6.3 Analytical analysis in Bayesian linear regression

We next study some of dropout's properties through its view as an approximating distribution in VI. Some interesting questions we answer include: 1) is dropout's regularisation data dependent? 2) does the dropout probability collapse to the MAP solution with finite data? and 3) does the approximate posterior collapse to a point mass in the limit of data? These questions can be answered through an analytical analysis in the special case of Bayesian linear regression. Even though we don't *need* VI in Bayesian linear regression, we can see what dropout VI looks like as we can solve everything analytically in this case. An empirical analysis of these questions for deep models is given later in §6.7.

We start with Bayesian linear regression with N data points, mapping Q-dimensional inputs $\mathbf{X} \in \mathbb{R}^{N \times Q}$ to D dimensional outputs $Y \in \mathbb{R}^{N \times D}$,

$$\mathbf{Y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}.$$

Here we assume observation noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and place a standard normal prior over the weights $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. For simplicity we shall assume that $D = 1$, hence $\mathbf{w} \in \mathbb{R}^{Q \times 1}$ is a column vector.

We can find the posterior distribution $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ analytically for this model:

$$\begin{aligned} p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}) / p(\mathbf{Y}|\mathbf{X}) \\ &= \mathcal{N}\Big(\mathbf{w}; (\mathbf{X}^T\mathbf{X} + \mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}, (\mathbf{X}^T\mathbf{X} + \mathbf{I})^{-1}\Big) \end{aligned} \tag{6.1}$$

The MAP estimate is thus

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{6.2}$$

In the linear case, the dropout variational distribution can be derived analytically as well. This approximate posterior distribution can then be compared to the exact

posterior. Define $q_{\mathbf{m},p}(\mathbf{w}) = \prod_{i=1}^{Q} q_{m_q,p}(w_q)$ with

$$q_{m_q,p}(w_q) = p\delta(w_q - m_q) + (1-p)\delta(w_q - 0)$$

given some variational parameters $\mathbf{m}$ and retain probability[1] p. This can be rewritten as $w_q = m_q \boldsymbol{\epsilon}_q$ with $\boldsymbol{\epsilon}_q \sim \text{Bern}(p)$, and in vector form $\mathbf{w} = \text{diag}([\boldsymbol{\epsilon}_i]_{i=1}^{Q}) \cdot \mathbf{m}$. We shall write $q(\mathbf{w})$ for $q_{\mathbf{m},p}(\mathbf{w})$ for brevity.

Evaluating the log evidence lower bound (ELBO) in the variational case amounts to:

$$\begin{aligned}
\log p(\mathbf{Y}|\mathbf{X}) &= \log \int p(\mathbf{Y}|\mathbf{X},\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w} \\
&\geq \int q(\mathbf{w}) \log p(\mathbf{Y}|\mathbf{X},\mathbf{w})\mathrm{d}\mathbf{w} - \mathrm{KL}\Big(q(\mathbf{w})||p(\mathbf{w})\Big) \\
&\propto \sum_{n=1}^{N} \left(-\frac{1}{2}y_n^2 + y_n \mathbf{x}_n \mathbb{E}_q[\mathbf{w}] - \frac{1}{2}\mathbf{x}_n \mathbb{E}_q[\mathbf{w}\mathbf{w}^T]\mathbf{x}_n^T \right) - \mathrm{KL}\Big(q(\mathbf{w})||p(\mathbf{w})\Big).
\end{aligned}$$

Following the definition of $q(\mathbf{w})$, we know that $\mathbb{E}_q[\mathbf{w}] = p\mathbf{m}$. Further, $\mathbb{E}_q[\mathbf{w}\mathbf{w}^T] = \text{Cov}_q[\mathbf{w}] + \mathbb{E}_q[\mathbf{w}]\mathbb{E}_q[\mathbf{w}]^T = p(1-p)\text{diag}([m_i^2]_{i=1}^{Q}) + p^2\mathbf{m}\mathbf{m}^T$ since each element of $\mathbf{w} \sim q(\mathbf{w})$ is a product of a scalar and an i.i.d. Bernoulli (remember that $\mathbf{m}$ is a row vector, therefore $\mathbf{m}\mathbf{m}^T$ is a matrix).

The log evidence can thus be bounded by:

$$\begin{aligned}
\log p(\mathbf{Y}|\mathbf{X}) &\geq \sum_{n=1}^{N} \left(-\frac{1}{2}y_n^2 + p y_n \mathbf{x}_n \mathbf{m} - \frac{p^2}{2}\mathbf{x}_n \mathbf{m}\mathbf{m}^T \mathbf{x}_n^T - \frac{p(1-p)}{2}\mathbf{x}_n \text{diag}([m_i^2]_{i=1}^{Q})\mathbf{x}_n^T \right) \\
&\qquad - \mathrm{KL}\Big(q(\mathbf{w})||p(\mathbf{w})\Big) \\
&\propto \sum_{n=1}^{N} \left(-\frac{1}{2}(y_n - p\mathbf{x}_n\mathbf{m})^2 - \frac{p(1-p)}{2}\mathbf{x}_n \text{diag}([m_i^2]_{i=1}^{Q})\mathbf{x}_n^T \right) - \mathrm{KL}\Big(q(\mathbf{w})||p(\mathbf{w})\Big) \\
&= -\frac{1}{2}||\mathbf{Y} - p\mathbf{X}\mathbf{m}||^2 - \frac{p(1-p)}{2}\sum_n \mathbf{x}_n \text{diag}([m_i^2]_{i=1}^{Q})\mathbf{x}_n^T - \mathrm{KL}\Big(q(\mathbf{w})||p(\mathbf{w})\Big).
\end{aligned}$$

This last expression can be simplified by noting that $\text{diag}([m_i^2]_{i=1}^{Q}) = \text{diag}([m_i]_{i=1}^{Q})^2$ thus

$$\begin{aligned}
\mathbf{x}_n \text{diag}([m_i^2]_{i=1}^{Q})\mathbf{x}_n^T &= \mathrm{Tr}\Big(\mathbf{x}_n \text{diag}(\mathbf{m})^2 \mathbf{x}_n^T\Big) \\
&= \mathrm{Tr}\Big(\text{diag}(\mathbf{m})\mathbf{x}_n^T \mathbf{x}_n \text{diag}(\mathbf{m})\Big).
\end{aligned}$$

[1] Note that here we denote dropout probability as $1-p$ instead of p.

Summing these over n we obtain

$$\sum_n \mathbf{x}_n \text{diag}([m_i^2]_{i=1}^Q)\mathbf{x}_n^T = \text{Tr}\Big(\text{diag}(\mathbf{m})\mathbf{X}^T\mathbf{X}\text{diag}(\mathbf{m})\Big) = \mathbf{m}^T\text{diag}(\mathbf{X}^T\mathbf{X})\mathbf{m}.$$

Following appendix A, we approximate the KL between $q(\mathbf{w})$ and a standard Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, I)$ as

$$\begin{aligned}\text{KL}\Big(q(\mathbf{w})||p(\mathbf{w})\Big) &= \sum_{i=1}^Q \text{KL}\Big(q(w_i)||p(w_i)\Big) \\ &\approx \frac{p}{2}\mathbf{m}^T\mathbf{m} - Q\mathcal{H}(p) + C,\end{aligned} \tag{6.3}$$

defining $\mathcal{H}(p) = -p\log p - (1-p)\log(1-p)$, and with a constant C.

Maximising the ELBO can be written as a minimisation objective:

$$\mathcal{L}(\mathbf{m}, p) = \underbrace{||\mathbf{Y} - p\mathbf{X}\mathbf{m}||^2 + p(1-p)\mathbf{m}^T\text{diag}(\mathbf{X}^T\mathbf{X})\mathbf{m}}_{\text{likelihood terms}} + \underbrace{p\mathbf{m}^T\mathbf{m} - 2Q\mathcal{H}(p)}_{\text{prior terms}}.$$

Remark (Is dropout's regularisation data dependent?)**.** It was suggested in [Srivastava et al., 2014, section 9.1] that the term $p(1-p)\mathbf{m}^T\text{diag}(\mathbf{X}^T\mathbf{X})\mathbf{m}$ in the equation above is a regularisation term dependent on the data. Following the interpretation of dropout as approximate inference with our specific distribution $q(\cdot)$ and a standard Gaussian prior we have that the term is derived from the likelihood contribution, i.e. the term is part of the generative model. But one could claim that this result follows from our ad hoc choice for $q(\cdot)$. Below we will see an alternative prior under which we can (approximately) derive the variational distribution *structure* optimally. For that alternative prior we recover an optimal $q(\cdot)$ with a distribution structure similar to the dropout variational distribution. Apart from suggesting why the dropout approximating distribution structure is sensible, the new objective will also possess similar properties to the ones studied here.

The solution to this last minimisation objective can be found analytically. We rewrite the objective as

$$\begin{aligned}\mathcal{L}(\mathbf{m}, p) = \sum_{n=1}^N (y_n^2 - 2py_n\mathbf{x}_n\mathbf{m} + p^2\mathbf{m}^T\mathbf{x}_n^T\mathbf{x}_n\mathbf{m}) + \mathbf{m}^T\Big(p(1-p)\text{diag}(\mathbf{X}^T\mathbf{X}) + p\mathbf{I}\Big)\mathbf{m} \\ - 2Q\mathcal{H}(p)\end{aligned} \tag{6.4}$$

$$= \sum_{n=1}^{N}(y_n^2 - 2py_n\mathbf{x}_n\mathbf{m}) + \mathbf{m}^T\Big(p^2\mathbf{X}^T\mathbf{X} + p(1-p)\text{diag}(\mathbf{X}^T\mathbf{X}) + p\mathbf{I}\Big)\mathbf{m} - 2Q\mathcal{H}(p)$$

Differentiating w.r.t. $\mathbf{m}$ we have

$$\frac{\partial\mathcal{L}(\mathbf{m},p)}{\partial\mathbf{m}} = -2p\mathbf{X}^T\mathbf{Y} + 2\Big(p^2\mathbf{X}^T\mathbf{X} + p(1-p)\text{diag}(\mathbf{X}^T\mathbf{X}) + p\mathbf{I}\Big)\mathbf{m}$$

Setting $\frac{\partial\mathcal{L}(\mathbf{m},p)}{\partial\mathbf{m}} = \mathbf{0}$ leads to optimal $\mathbf{m}^*$ (under the constraint $p > 0$):

$$\mathbf{m}^* = \Big(pN\boldsymbol{\Sigma} + (1-p)N\boldsymbol{\Lambda} + \mathbf{I}\Big)^{-1}\mathbf{X}^T\mathbf{Y} \tag{6.5}$$

with $\boldsymbol{\Sigma} = \mathbf{X}^T\mathbf{X}/N$ and $\boldsymbol{\Lambda} = \text{diag}(\mathbf{X}^T\mathbf{X})/N$. This effectively shrinks the off-diagonal covariance terms, reducing the sensitivity of linear regression to colinear inputs[2].

Remark (Does the dropout probability collapse to the MAP solution with finite data?)**.** Here we see that the optimal variational parameter $\mathbf{m}^*$ equals the MAP estimate given in eq. (6.2) only for $p = 1$ or for $\boldsymbol{\Lambda}$ constant. We will next see that in the limit of data, optimal p^* is indeed 1.

Plugging $\mathbf{m}^*$ into $\mathcal{L}(\mathbf{m}, p)$ we obtain:

$$\begin{aligned}\mathcal{L}(\mathbf{m}^*, p) &= \mathbf{Y}^T\mathbf{Y} - 2p\mathbf{Y}^T\mathbf{X}\mathbf{m}^* + p\mathbf{m}^{*T}\Big(pN\boldsymbol{\Sigma} + (1-p)N\boldsymbol{\Lambda} + \mathbf{I}\Big)\mathbf{m}^* - 2Q\mathcal{H}(p)\\ &= \mathbf{Y}^T\mathbf{Y} - p\mathbf{Y}^T\mathbf{X}\Big(pN\boldsymbol{\Sigma} + (1-p)N\boldsymbol{\Lambda} + \mathbf{I}\Big)^{-1}\mathbf{X}^T\mathbf{Y} - 2Q\mathcal{H}(p).\end{aligned}$$

In the limit of data we have (assuming that the limit exists)

$$\begin{aligned}\frac{\mathcal{L}(\mathbf{m}^*, p)}{N} &= \frac{\mathbf{Y}^T\mathbf{Y}}{N} - p\frac{\mathbf{Y}^T\mathbf{X}}{N}\Big(p\boldsymbol{\Sigma} + (1-p)\boldsymbol{\Lambda} + \frac{\mathbf{I}}{N}\Big)^{-1}\frac{\mathbf{X}^T\mathbf{Y}}{N} - \frac{2Q\mathcal{H}(p)}{N}\\ &\xrightarrow[N\to\infty]{} a - \mathbf{b}^T\Big(\boldsymbol{\Sigma} + (p^{-1}-1)\boldsymbol{\Lambda}\Big)^{-1}\mathbf{b}\\ &=: \bar{\mathcal{L}}(p),\end{aligned}$$

with $a = \lim_{N\to\infty}\mathbf{Y}^T\mathbf{Y}/N$ and $\mathbf{b} = \lim_{N\to\infty}\mathbf{X}^T\mathbf{Y}/N$.

[2]Equation (6.5) was previously presented in [Srivastava et al., 2014; Wager et al., 2013; Wang and Manning, 2013], where the dropout objective was interpreted as a form of ridge regression with the design matrix columns normalized. Dropout in linear networks was also studied in [Baldi and Sadowski, 2013].

Remark (Does the approximate posterior collapse to a point mass in the limit of data?)**.** We have that $p = 1$ (no dropout) is a minimiser of $\bar{\mathcal{L}}(p)$ (and the only one when $\mathbf{\Lambda}$ is positive definite): $\mathbf{\Sigma}$ and $\mathbf{\Lambda}$ are positive semi-definite (PSD). For $p < 1$, we have $p^{-1} - 1 > 0$, therefore $(p^{-1} - 1)\mathbf{\Lambda}$ is PSD, and so is $\mathbf{\Sigma} + (p^{-1} - 1)\mathbf{\Lambda}$. Since $\mathbf{\Sigma} \preceq \mathbf{\Sigma} + (p^{-1} - 1)\mathbf{\Lambda}$, we have $\left(\mathbf{\Sigma} + (p^{-1} - 1)\mathbf{\Lambda}\right)^{-1} \preceq \mathbf{\Sigma}^{-1}$, and from the definition of PSD: $\mathbf{b}^T\left(\mathbf{\Sigma} + (p^{-1} - 1)\mathbf{\Lambda}\right)^{-1}\mathbf{b} \leq \mathbf{b}^T\mathbf{\Sigma}^{-1}\mathbf{b}$. As a result,

$$a - \mathbf{b}^T\left(\mathbf{\Sigma} + (p^{-1} - 1)\mathbf{\Lambda}\right)^{-1}\mathbf{b} \geq a - \mathbf{b}^T\mathbf{\Sigma}^{-1}\mathbf{b},$$

resulting in $\bar{\mathcal{L}}(p) \geq \bar{\mathcal{L}}(1)$ for all $p \in (0, 1)$. For $\mathbf{\Lambda}$ positive definite the inequalities above are strict, leading to $\bar{\mathcal{L}}(p) > \bar{\mathcal{L}}(1)$ for all $p \in (0, 1)$, i.e. $p^* = 1$ (no dropout) is the unique minimiser of our optimisation objective, and the approximate posterior collapses to a point mass in the limit of data.

6.4 ELBO correlation with test log likelihood

This has been joint work with Mark van der Wilk.

In an attempt to maximise model performance, it might be tempting to optimise dropout's probability p as a variational parameter following its VI interpretation. Interestingly enough, optimising the dropout probability can give mixed results. In this section we analyse this behaviour. We plot model ELBO versus test log likelihood for different dropout probabilities, and assess the correlation between the ELBO and the test log likelihood.

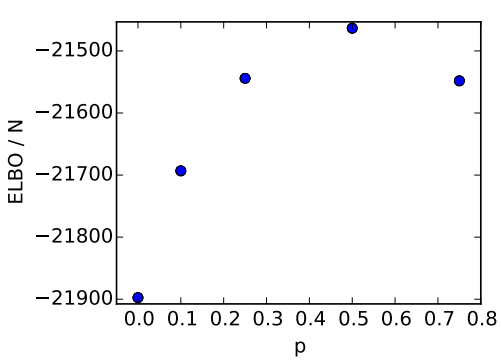
We repeat the experiment setup of §4.2 and use ReLU models with 4 hidden layers and 1024 units in each layer evaluated on Snelson and Ghahramani [2005]'s dataset with $N = 5000$ training points. We set model precision to $\tau = 50$ and assess model ELBO and test log likelihood at the end of optimisation for various probabilities[3] p of the weights to be set to zero. Each experiment was repeated three times. These can be seen in fig. 6.1. As can be seen, the ELBO correlates strongly with the test log likelihood—as the ELBO increases so does the test log likelihood. In fig. 6.2 we can see how models with dropout probability $p = 0$ (no dropout) cannot capture the full range of noise of the data, whereas by increasing the dropout probability we manage to model the noise better and better. For too large dropout probability ($p = 0.75$) the model starts underfitting.

[3]We performed a grid-search over p since we want to plot model performance for different p values.

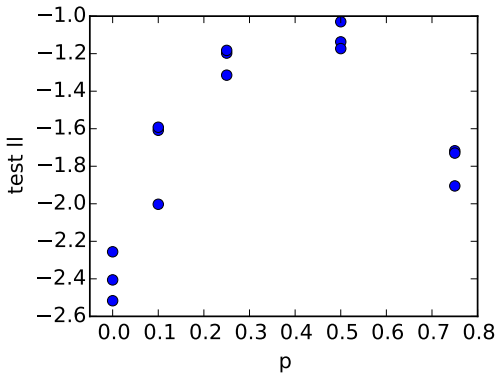
It is worth mentioning that for $p = 0.9$ the model does not converge at all. It is also interesting to note that test RMSE does not seem to correlate to the ELBO as is seen in fig. 6.3. The test log likelihood is composed of the test RMSE scaled by the uncertainty with an added uncertainty "penalty" term. This means that predictive uncertainty forms an important factor in the ELBO correlation with the test log likelihood. Repeating this experiment with $\tau = 20$ gives very similar results.

Repeating the same experiment with a different model precision value $\tau = 10$ we observe very different results though. Figure 6.4 shows that in this model when we set the dropout probability to $p = 0.5$ we obtain a worse test log likelihood than setting $p = 0$, even though the ELBO for the former is still higher than that of the latter. Note though that the model fits well with no dropout in this setting, and adding dropout with a large probability leads to deteriorated results (fig. 6.5). This might stem from the ELBO being too loose in some model setups, explaining why some attempts at optimising the dropout probability have failed in the past. However, for models in which the bound is tight enough, dropout optimisation can be done fairly well. This is explored in §6.7.

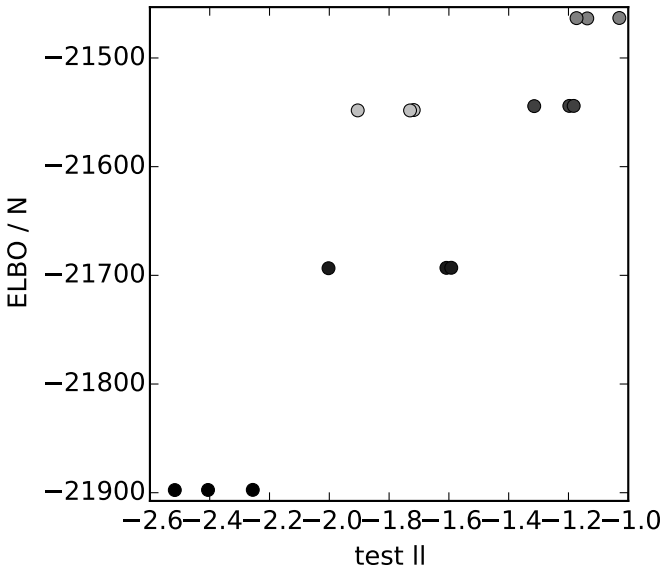
Remark (Model selection and bound tightness)**.** In a wider context, this last result above suggests some interesting questions. The assumption underlying variational inference is that models with higher ELBO correspond to "better" models (better according to what metric is a different question though). This is because the higher the ELBO, the lower the KL divergence between the approximate posterior and the true posterior would become. This fact is what drives us to look for the variational parameters that maximise the ELBO, and in turn minimise this notion of "distance" between our approximation and the true posterior. Since in our setting above the dropout probability p is a variational parameter, we would assume that maximising the ELBO w.r.t. p would lead to improved model performance. The fact that this is the case for some models ($\tau = 20, 50$) but not others ($\tau = 10$) is perplexing.
These results are related to the concept of *bound tightness*. MacKay [1992a] approximated the model evidence (which our ELBO is a lower bound to) and showed positive correlation between the evidence and test RMSE. He then performed model selection by choosing the model with the highest evidence. In VI the ELBO is often used as a proxy to the model evidence when the ELBO is tight enough [Rasmussen and Williams, 2006] (i.e. models with higher model evidence are assumed to have higher bound to that evidence). When changing model precision, different model precision values τ correspond to different models, and therefore each ELBO is a bound to a different (constant) model evidence. In Gaussian process approximations



(a) ELBO as a function of dropout p

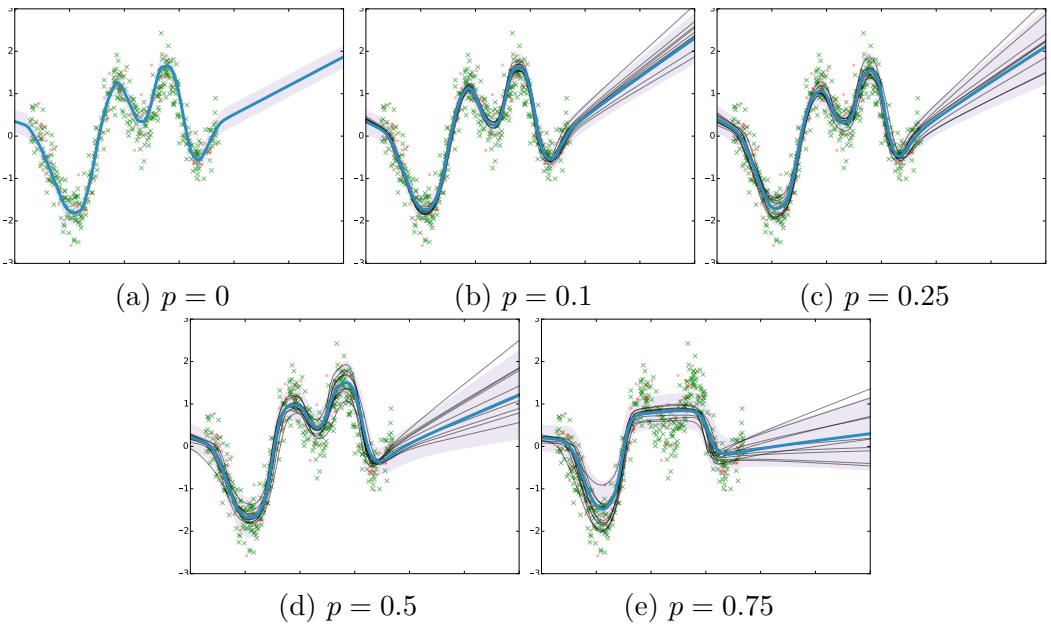


(b) Test log likelihood as a function of p



(c) ELBO as a function of test log likelihood for various p (darker = lower probability).

Fig. 6.1 ELBO (per training point) and test log likelihood (per test point) for various values of dropout probability for a model with 4 hidden layers, 1024 units, and model precision $\tau = 50$.



(a) $p = 0$ (b) $p = 0.1$ (c) $p = 0.25$

(d) $p = 0.5$ (e) $p = 0.75$

Fig. 6.2 Model fit for model precision $\tau = 50$ with various dropout probabilities

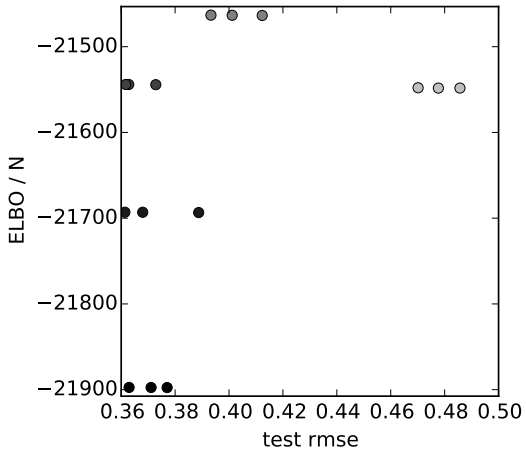
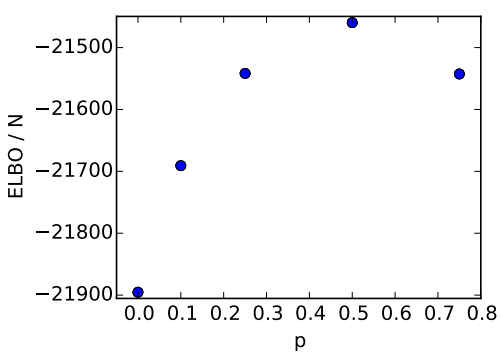
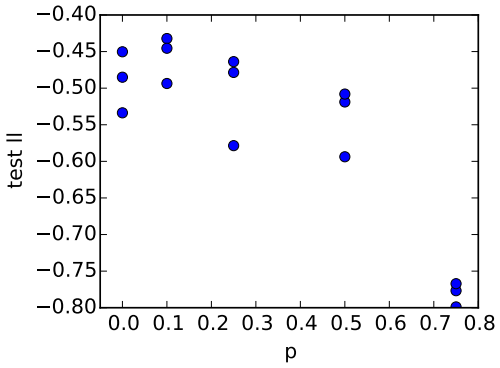


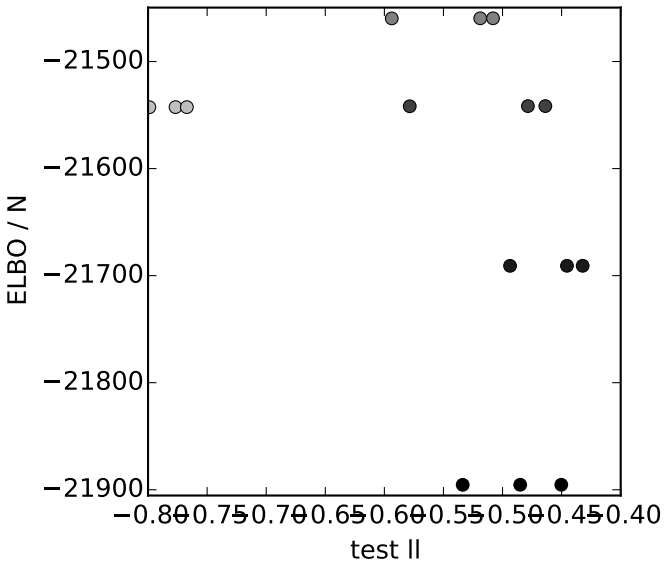
Fig. 6.3 ELBO as a function of test RMSE for $\tau = 50$



(a) ELBO as a function of dropout p

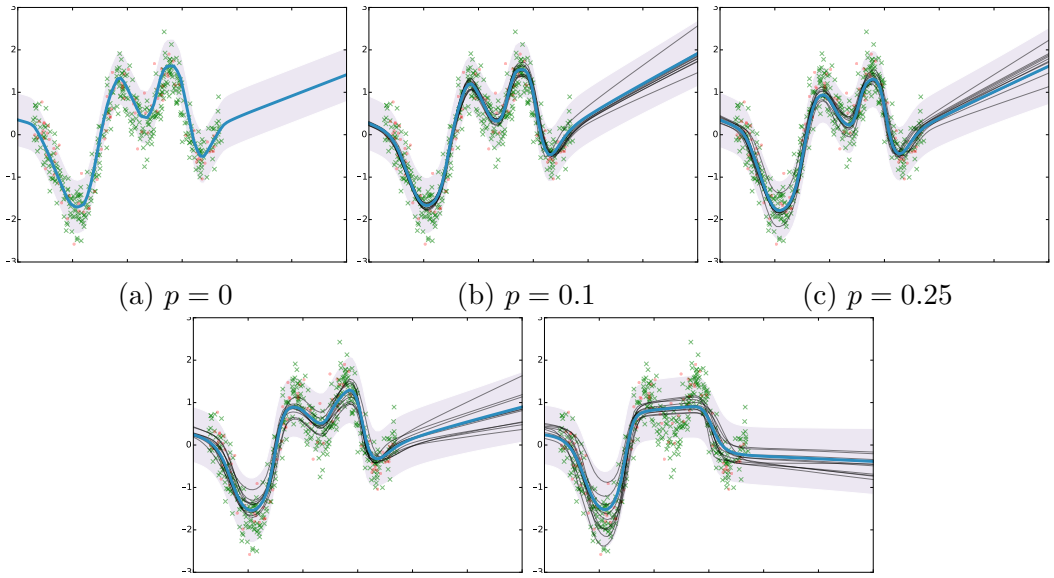


(b) Test log likelihood as a function of p



(c) ELBO as a function of test log likelihood for various p (darker = lower probability).

Fig. 6.4 ELBO (per training point) and test log likelihood (per test point) for various values of dropout probability for a model with 4 hidden layers, 1024 units, and model precision $\tau = 10$.



(a) $p = 0$ (b) $p = 0.1$ (c) $p = 0.25$

(d) $p = 0.5$ (e) $p = 0.75$

Fig. 6.5 Model fit for model precision $\tau = 10$ with various dropout probabilities

for example the bound is empirically observed to be tight enough to allow us to choose a model precision τ based on the ELBO. In our setting the ELBO seems to not be indicative of what model we should choose in some cases (fig. 6.6 shows the ELBO for all models above, with highest ELBO obtained at $p = 0.5$; further, fig. 6.7 shows that $\tau = 10$ has slightly higher ELBO than $\tau = 20$ and than $\tau = 50$). This suggests that the bound is not as tight as in the GP case.
Attempting to explain the above, there are several possible hypotheses one could propose:

1. The prior might dominate the optimisation objective in some models because there is not enough data compared to model size. The entropy in p stemming from the KL contribution (eq. (A.1)), which gives the ELBO its parabola-like shape, is scaled by the model size. In the limit of data this term would vanish. This interpretation is supported by the results in fig. 6.8 where the experiment with $\tau = 10$ above is repeated with $50 \cdot 10^6$ data points instead of $50 \cdot 10^3$, and resulting in positive correlation. Note that in this setting $\tau = 50$ has a "kink" in the ELBO-test log likelihood plot, but in both cases ELBO-test RMSE is highly correlated (not shown).

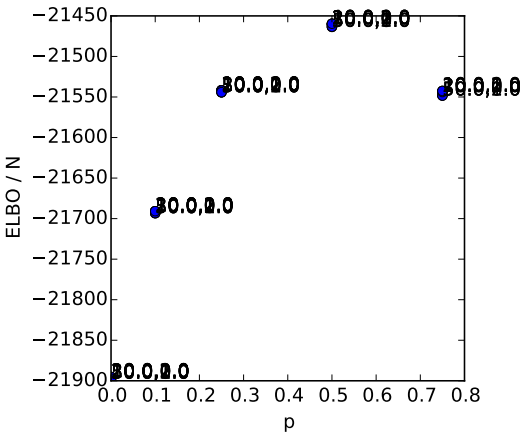


Fig. 6.6 ELBO as a function of dropout p for all model precision values $\tau = 10, 20, 50$

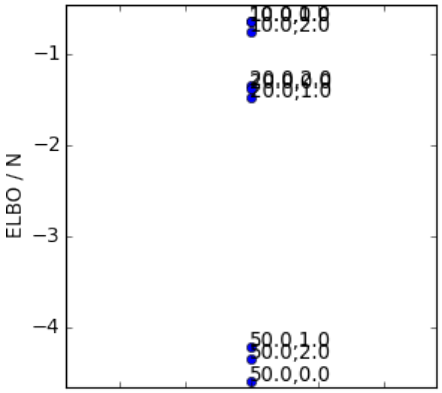


Fig. 6.7 ELBO as a function of dropout p for all model precision values $\tau = 10, 20, 50$, **zoomed in at** $p = 0.5$ (with labels of the form (τ, repetition number))

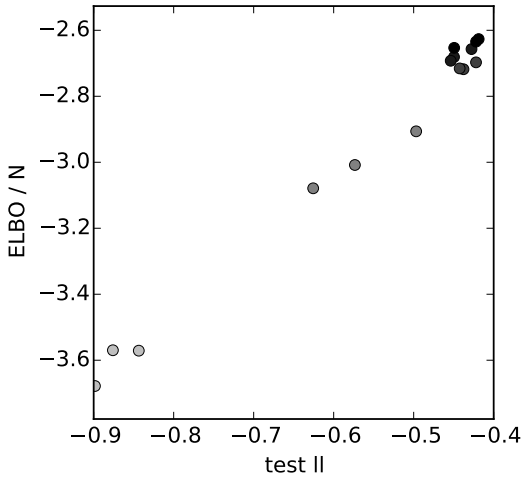


Fig. 6.8 ELBO as a function of test log likelihood for various p (darker = lower probability), with 4 hidden layers, 1024 units, and model precision $\tau = 10$ with $\mathbf{50 \cdot 10^6}$ **data points**.

2. The model might be misspecified (as discussed in [MacKay, 1992a, section 3.4]). MacKay [1992a] showed that when using inappropriate prior distributions a "kink" appeared in his evidence-test RMSE plots; he solved this by changing his prior. A lack of correlation between model evidence and test RMSE would lead to a lack of correlation between the ELBO and test RMSE even if the bound *is* tight.

3. The ELBO constant terms might not be specified correctly. The KL condition in appendix A only requires us to specify a $q(\boldsymbol{\omega})$ and $p(\boldsymbol{\omega})$ s.t. the derivatives of the ELBO and the dropout optimisation objective agree w.r.t. $\mathbf{W}$. The terms w.r.t. p however can change arbitrarily.

4. Lastly, the dropout probability might not be a variational parameter at all but rather a model parameter. Changing the dropout probability might change the model evidence—the quantity we bound.

Hypotheses 2 and 3 suggest that various terms apart from $\mathbf{W}$ in $p(\boldsymbol{\omega})$ might be affecting the optimisation objective. For example, specifying a different prior s.t. the KL condition is still satisfied w.r.t. the same approximating distribution $q(\boldsymbol{\omega})$ results in a different ELBO (as we will see next). Hypothesis 4 relates to prior selection as well. As we will see in a later section in this chapter, under some priors the dropout probability will be determined by our prior hyper-parameters, hence changing the dropout probability would change the evidence we bound (explaining the peculiar behaviour above).
In the final sections of this chapter we give some evidence in support of the various possible hypotheses suggested above. Further study of these hypotheses is left for future research.

6.5 Discrete prior models

In the previous chapters we attempted to use a discrete approximating distribution $q(\mathbf{w}_k) = p\delta(\mathbf{w}_k - \mathbf{0}) + (1-p)\delta(\mathbf{w}_k - \mathbf{m}_k)$ to recover dropout. But our prior had a continuous probability density function (a standard Gaussian distribution). We were therefore forced to approximate the KL between the approximating distribution and the prior by embedding the approximating distribution in a continuous space using a mixture of two Gaussians with small standard deviations σ (appendix A). But for a

fixed small σ the constant in the KL to the prior (eq. (A.1)) is rather large, making the lower bound quite loose. This bound becomes looser and looser for smaller and smaller σ, and diverges to negative infinity at $\sigma = 0$. This raises issues when we attempt to optimise the ELBO w.r.t. model hyper-parameters such as the length-scale for example[4]. More specifically, we will look at the bound when the prior above is set as $\mathcal{N}(0, l^{-2}I)$. For this prior the constant C contains the terms $-l^2\sigma^2 + \log l^2\sigma^2$. If we maximise the ELBO w.r.t. length-scale l, we would choose (with a very small fixed σ^2) a very large l. In fact, we can make the model completely ignore the likelihood term by setting σ^2 to be small enough: for every fixed number of points N there exists a σ^2 value such that $\log l^2$ dominates $\sum_n \frac{\tau}{2}(\mu - y_n)^2$ with μ being the mean of the data[5]. Because σ is held constant and identical in all models, if we perform model comparison based on the ELBO we would prefer the model with the longer length-scale l^2.

This suggests that the model might be misspecified w.r.t. our choice of approximating distribution (which we chose in order to recover dropout). Model misspecification is often discussed with respect to model evidence—when the evidence does not correlate to test error (see for example [MacKay, 1992a]). But in our case we compare log evidence *lower bounds*, and are interested to *define a model* in which a given approximating distribution specifies a tight bound.

A possible way to fix this issue is to specify an alternative prior, inducing a slightly altered model. Instead of using a continuous probability density function $p_l(\mathbf{w}) = \mathcal{N}(0, l^{-2}I)$, we will use a discrete probability *mass* function $p_l(\mathbf{w}) \propto e^{-\frac{l^2}{2}\mathbf{w}^T\mathbf{w}}$ defined over a *finite* space $\mathbf{w} \in X$ (a similar approach of quantising the space was used in [Hinton and Van Camp, 1993]). A continuous prior forced us to embed the discrete approximating distribution $q(\cdot)$ in a continuous space in order for the KL between the two to be properly defined. We did this using a mixture of Gaussians with small standard deviations σ. But this approach has led to our increasingly loose bound as the standard deviation σ decreased. The use of a discrete prior instead allows us to evaluate the KL divergence of both distributions over a finite space X.

In our case, since we optimise model parameters $\mathbf{w}$ on a computer, we define the space X to be a finite vector space defined over the finite field of numbers representable on a computer (for example numbers up to a certain precision). To ensure parameter gradients are properly defined, we relax the objective and embed the parameters in a continuous space *for optimisation*.

[4] This can be circumvented by grid-searching w.r.t. validation log likelihood instead of the ELBO.

[5] For large enough l^2 the optimal model parameters would be zero because of the $-\frac{l^2 p}{2}\mathbf{m}^T\mathbf{m}$ term in the prior; this leads to the model predicting the mean of the data denoted μ here.

Given the discrete approximating distribution $q(\mathbf{w}_k) = p\delta(\mathbf{w}_k - \mathbf{0}) + (1-p)\delta(\mathbf{w}_k - \mathbf{m}_k)$ the expected log likelihood terms stay the same as before, and the KL to the discrete prior distribution above is given by:

$$\begin{aligned}
\mathrm{KL}(q(\mathbf{w})||p_l(\mathbf{w})) &= \sum_{\mathbf{w}\in X} q(\mathbf{w}) \log \frac{q(\mathbf{w})}{p_l(\mathbf{w})} \\
&= \sum_{\mathbf{w}\in X} q(\mathbf{w}) \log \frac{q(\mathbf{w})}{e^{-\frac{l^2}{2}\mathbf{w}^T\mathbf{w}}} + \log Z_l \\
&= p \log \frac{p}{1} + (1-p) \log \frac{1-p}{e^{-\frac{l^2}{2}\mathbf{m}^T\mathbf{m}}} + \log Z_l \\
&\qquad + \sum_{\mathbf{w}\in X/\{0,\mathbf{m}\}} 0 \log \frac{0}{e^{-\frac{l^2}{2}\mathbf{w}^T\mathbf{w}}} \\
&= -\mathcal{H}(p) + \frac{l^2(1-p)}{2}\mathbf{m}^T\mathbf{m} + \log Z_l
\end{aligned}$$

with $\mathcal{H}(p) = -p\log p - (1-p)\log(1-p)$ and with the last transition following the identity $0 \log 0 = 0$. Since Z_l is the normaliser of $p_l(\mathbf{w})$:

$$Z_l = \sum_{\mathbf{w}\in X} e^{-\frac{l^2}{2}\mathbf{w}^T\mathbf{w}} \approx |\Delta\mathbf{w}|^{-1}(2\pi l^{-2})^{K/2}$$

with K being the dimensionality of the vector $\mathbf{w}$ and $|\Delta\mathbf{w}|$ the quantisation interval of the space X, we have:

$$\log Z_l \approx -K \log l - \log |\Delta\mathbf{w}| + \frac{K}{2} \log 2\pi.$$

This leads to

$$\mathrm{KL}(q(\mathbf{w})||p_l(\mathbf{w})) \approx \frac{l^2(1-p)}{2}\mathbf{m}^T\mathbf{m} - K \log l + \frac{K}{2} \log 2\pi - \mathcal{H}(p) - \log |\Delta\mathbf{w}|. \tag{6.6}$$

Surprisingly perhaps, the derivatives of this KL is identical to the one we used before (eq. (A.1)) for the terms $\mathbf{m}$ and p, but unlike before, there are no additional terms that diverge to infinity (terms dependent on σ). This means that approximate inference with this prior following the setting of §4.2 would give the same results observed in that chapter, where the only difference is the bound being more tight. An alternative interesting prior is discussed next, a prior that sheds light on dropout's peculiar *structure*.

6.6 Dropout as a proxy posterior in spike and slab prior models

I thank Nilesh Tripuraneni for suggesting the relation between dropout and spike and slab priors.

In his thesis from 1992, David MacKay discussed the possibility of placing a spike and slab prior over a neural network's weights [MacKay, 1992a, section 7.4]. MacKay recalled personal communication with Geoff Hinton, who suggested that an ideal prior in BNNs would set part of the weights to be exactly zero[6]. Many works at the time tried to approximate inference in similarly motivated models to varying degrees of success [Ji et al., 1990; Nowlan and Hinton, 1992; Weigend et al., 1991], none of which survived to modern day. In the following we will develop approximate inference in BNNs with spike and slab priors relying on recent VI advances, in effect formalising these ideas from 25 years ago. We will approximate the *optimal structure* of the approximating distribution, which as we will see turns out to be closely related to dropout's structure.

6.6.1 Historical context

MacKay [1992a]'s comments mentioned above were made with respect to an early draft of Nowlan and Hinton [1992], which extended on the work of Weigend et al. [1991].

Weigend et al. [1991] relied on MDL to motivate the addition of a regularisation term to the optimisation objective of a NN, in order to penalise complex models. They offered a Bayesian interpretation to their objective as maximising the log likelihood plus log prior (MAP) with a mixture prior over the weights, where the mixture was of a wide uniform and a Gaussian centred at zero, with the Gaussian's width being learnt.

Nowlan and Hinton [1992], following the Bayesian interpretation of the weight decay as a Gaussian prior, claimed that a Gaussian prior can be used to eliminate small weights. But at the same time it forces other weights to contract to the origin as well, weights that are needed to "explain the data as well" and thus should not be forced to the origin (the demand for a prior to contract "unneeded" weights to zero might have stemmed from the MDL interpretation of model complexity, but was not justified in the paper). Nowlan and Hinton [1992] proposed to use a mixture of a narrow Gaussian together with a wide Gaussian. They implemented this by placing a mixture of Gaussians prior and optimising the MAP over the means and standard deviations of the Gaussians.

[6]Note that the *prior* is constructed to be sparse here rather than the *posterior* as in dropout.

MacKay [1992a], repeating the desire to set part of the weights to be exactly zero, commented on the work of Nowlan and Hinton [1992]. MacKay [1992a] said that the non-zero width of the narrow Gaussian was a consequence of computational practicality, thus Nowlan and Hinton [1992]'s approach of inferring the width of the narrow Gaussian was not appropriate. MacKay [1992a] concluded that it will be interesting to see if a priori setting part of the weights to be exactly zero could be formalised, leading to a "well-founded" technique.

6.6.2 Spike and slab prior models

The desire to set a subset of weights to be exactly zero a priori can be materialised by placing a *spike and slab* prior over the weights. Spike and slab distributions return realisations which are identically zero with some probability (the spike) or sampled from a wide Gaussian otherwise (the slab), and have been used in the context of variable selection for example [Chipman; George and McCulloch, 1993, 1997; Ishwaran and Rao, 2005; Madigan and Raftery, 1994; Mitchell and Beauchamp, 1988]. We start by placing a spike and slab prior over each row of each weight matrix $\mathbf{w}_{ik}$, and assume that the rows are a priori independent of each other:

$$p(\mathbf{w}_{ik}) = f\delta(\mathbf{w}_{ik} - \mathbf{0}) + (1 - f)\mathcal{N}(\mathbf{w}_{ik}; \mathbf{0}, l^{-2}I) \tag{6.7}$$

with prior probability f and prior length-scale l. The first component is a point mass at a vector of zeros, setting an entire weight matrix row to zero with probability f (**a priori**). The second component draws a weight vector from a multivariate Gaussian distribution with probability $1 - f$. Draws from this distribution will give high frequencies for short length-scale l (resulting in erratic functions), and low frequencies for long length-scales (resulting in smooth functions). Placing the distribution over the rows of the matrix $\mathbf{W}$ captures correlations over the function's frequencies ($\mathbf{W}$'s columns, §3.2.3).

Given the prior above, we use a Gaussian likelihood for regression:

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega}) = \mathcal{N}(\mathbf{h}_{L-1}(...\mathbf{h}_1(\mathbf{x})...)\mathbf{W}_L, \tau^{-1}I)$$

with $\mathbf{h}_i(\mathbf{x}) = \sigma(\mathbf{x}W_i)$, L layers, $\boldsymbol{\omega} = \{W_i\}_{i=1}^L$ a set of random matrices, and observation noise τ^{-1}.

6.6.3 Related work

Some of the techniques above have been revisited in other modern literature [Blundell et al., 2015; Goodfellow et al., 2012; Louizos, 2015; Titsias and Lázaro-Gredilla, 2011]. Blundell et al. [2015] for example use a BNN prior which resembles that of Nowlan and Hinton [1992]. Louizos [2015] looks at a spike and slab prior distribution in a VI setting, but makes use of a loose lower bound in order to evaluate the KL divergence between a spike and slab approximate posterior and the prior.

6.6.4 Approximate inference with free-form variational distributions

The posterior distribution over $\boldsymbol{\omega}$ given observed data $\mathbf{X}, \mathbf{Y}$ is difficult to evaluate. Instead, we use variational inference and approximate it with a variational distribution $q(\boldsymbol{\omega})$. We define the approximating distribution to factorise over the rows of the weight matrices $\mathbf{w}_{ik}$:

$$q(\boldsymbol{\omega}) = \prod_{i=1}^{L} \prod_{k=1}^{K} q(\mathbf{w}_{ik})$$

with $\mathbf{W}_i$ composed of k rows.

Unlike previous work, here we will not specify a structure for $q(\cdot)$, but rather find the optimal variational distribution structure using *calculus of variations*. This can be done using the following lemma and the corollary following it:

Lemma 1. Let $p(W)$ and $q(W)$ be two distributions defined over the same space $W \in \mathcal{W}$. Further, assume the distribution $q(W)$ factorises over $W = [\mathbf{w}_1, ..., \mathbf{w}_K]$: $q(W) = \prod_k q(\mathbf{w}_k)$.

The optimal distribution $q^*(\mathbf{w}_k)$ minimising $\mathrm{KL}(q(W)||p(W))$ is given by:

$$q^*(\mathbf{w}_k) \propto e^{E_{q(W)/q(\mathbf{w}_k)}[\log p(W)]}.$$

Proof. Using calculus of variations,

$$\begin{aligned}
&\frac{\partial}{\partial q(\mathbf{w}_k)} \left(\mathrm{KL}(q(W)||p(W)) + \lambda \left(\int q(\mathbf{w}_k) \mathrm{d}\mathbf{w}_k - 1 \right) \right) \\
&\quad = \int \frac{q(W)}{q(\mathbf{w}_k)} \log \frac{q(W)}{p(W)} \mathrm{d}(\mathbf{w}_i)_{\neq k} + \lambda \\
&\quad = -E_{q(W)/q(\mathbf{w}_k)}[\log p(W)] + \int \frac{q(W)}{q(\mathbf{w}_k)} \log \frac{q(W)}{q(\mathbf{w}_k)} \mathrm{d}(\mathbf{w}_i)_{\neq k} + \log q(\mathbf{w}_k) + \lambda
\end{aligned}$$

Setting this last quantity to zero we obtain

$$\log q(\mathbf{w}_k) = -\lambda - E_{q(W)/q(\mathbf{w}_k)}[\log q(W)/q(\mathbf{w}_k)] + E_{q(W)/q(\mathbf{w}_k)}[\log p(W)]$$

which leads to

$$q^*(\mathbf{w}_k) \propto e^{E_{q(W)/q(\mathbf{w}_k)}[\log p(W)]}.$$

□

This result is known in the literature. From this lemma we can derive the following corollary:

Corollary 1. Given a prior of the form $p(W) = \prod_k p(\mathbf{w}_k)$ and likelihood $p(Y|W)$, the posterior $p(W|Y)$ can be approximated with an optimal factorised distribution $q^*(W) = \prod_k q^*(\mathbf{w}_k)$ given by

$$q^*(\mathbf{w}_k) \propto e^{E_{q(W)/q(\mathbf{w}_k)}[\log p(Y|W)]} p(\mathbf{w}_k).$$

This last equation is fundamental in variational message passing for example [Winn and Bishop, 2005].

6.6.5 Proxy optimal approximating distribution

In our BNN case above, the optimal approximating distribution can be split into two terms:

$$q^*(\mathbf{w}_{ik}) \propto \underbrace{e^{E_{q(\boldsymbol{\omega})/q(\mathbf{w}_{ik})}[\log p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})]}}_{\text{Nasty distribution}} \underbrace{p(\mathbf{w}_{ik})}_{\text{Nice dist.}}.$$

The nasty distribution term can be evaluated analytically for linear models. But for deep networks this distribution becomes rather complex. Instead, we will moment-match the nasty distribution. We fit it with a Gaussian parametrised with mean $\mathbf{m}_{ik}$ and diagonal variance $\sigma_{ik}^2 I$. This alters the optimal approximating distribution structure to

$$q_{\mathbf{m}_{ik},\sigma_{ik}}(\mathbf{w}_{ik}) \propto \mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) p(\mathbf{w}_{ik}).$$

We then maximise the ELBO w.r.t. the variational parameters $\mathbf{m}_{ik}, \sigma_{ik}$ for each approximating distribution $q_{\mathbf{m}_{ik},\sigma_{ik}}(\mathbf{w}_{ik})$.

The structure of the approximating distribution for the spike and slab prior (eq. (6.7)) can be evaluated analytically now:

$$\begin{aligned} q_{\mathbf{m}_{ik},\sigma_{ik}}(\mathbf{w}_{ik}) &\propto \mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I)\Big(f\delta(\mathbf{w}_{ik} - \mathbf{0}) + (1-f)\mathcal{N}(\mathbf{w}_{ik}; \mathbf{0}, l^{-2} I)\Big) \\ &\propto fC_1\delta(\mathbf{w}_{ik} - \mathbf{0}) + (1-f)C_2\mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I\right) \end{aligned}$$

with $C_1 = \mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, \sigma_{ik}^2 I)$ and $C_2 = \mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, (\sigma_{ik}^2 + l^{-2})I)$.

The normaliser for this distribution is given by

$$\begin{aligned} Z_q &= \int fC_1\delta(\mathbf{w}_{ik} - \mathbf{0}) + (1-f)C_2\mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I\right) \mathrm{d}\mathbf{w}_{ik} \\ &= f\mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) + (1-f)\mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, (\sigma_{ik}^2 + l^{-2})I). \end{aligned}$$

Writing

$$\begin{aligned} \alpha(\mathbf{m}_{ik}, \sigma_{ik}, f) &= \frac{f\mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, \sigma_{ik}^2 I)}{(1-f)\mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, (\sigma_{ik}^2 + l^{-2})I)} \\ &= \frac{f}{(1-f)}(1 + l^{-2}\sigma_{ik}^{-2})^{K/2} e^{-\frac{1}{2}(l^2\sigma_{ik}^4 + \sigma_{ik}^2)^{-1}\mathbf{m}_{ik}^T\mathbf{m}_{ik}} \end{aligned} \tag{6.8}$$

we have

$$q_{\mathbf{m}_{ik},\sigma_{ik}}(\mathbf{w}_{ik}) = \frac{\alpha}{\alpha+1}\delta(\mathbf{w}_{ik} - \mathbf{0}) + \frac{1}{\alpha+1}\mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I\right). \tag{6.9}$$

This approximating distribution sets each weight row to zero with probability $\alpha/(\alpha+1)$, which is determined by the magnitude of the variational parameters and the prior probability f. With probability $1/(\alpha+1)$, a row's weight is drawn from a normal distribution centred around mean $\mathbf{m}_{ik}$ scaled by $1/(1+l^2\sigma_{ik}^2)$, and with variance $\sigma_{ik}^2/(1+l^2\sigma_{ik}^2)$. For a large prior probability f tending towards 1, we have that α will tend towards infinity, and $q(\cdot)$ will tend towards a point estimate at 0. For small prior probability f, the distribution $q(\cdot)$ will tend towards a multivariate Gaussian distribution. Further, letting σ_{ik} tend towards 0 we recover dropout's "spike and spike" behaviour (but with data dependent dropout probability, which tends towards 1). On the other hand, by increasing σ_{ik}, the variance of the Gaussian component will tend towards l^{-2}.

6.6.6 Spike and slab and dropout

Evaluating the ELBO with our approximating distribution, we get the following objective:

$$\begin{aligned}\log p(\mathbf{Y}|\mathbf{X}) &\geq \sum_{i=1}^{N} -\frac{\tau}{2}\int q(\boldsymbol{\omega})||\mathbf{y}_i - \mu^{\boldsymbol{\omega}}(\mathbf{x}_i)||^2 \mathrm{d}\boldsymbol{\omega} \\ &\quad + \mathrm{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega})) + \frac{D}{2}\log\tau - \frac{D}{2}\log 2\pi \\ &=: \mathcal{L}\end{aligned}$$

defining $\mu^{\boldsymbol{\omega}}(\mathbf{x}_i) = \mathbf{h}_{L-1}(...\mathbf{h}_1(\mathbf{x})...)\mathbf{W}_L$ with D output dimensions. This last quantity can be approximated by Monte Carlo integration with a single draw $\widehat{\boldsymbol{\omega}}_i \sim q(\boldsymbol{\omega})$ for each integral in the summation:

$$\widehat{\mathcal{L}} := \sum_{i=1}^{N} -\frac{\tau}{2}||\mathbf{y}_i - \mu^{\widehat{\boldsymbol{\omega}}_i}(\mathbf{x}_i)||^2 + \mathrm{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega})) + \frac{D}{2}\log\tau - \frac{D}{2}\log 2\pi$$

with $\widehat{\mathcal{L}}$ forming an unbiased estimator to the exact ELBO, $E_{q(\boldsymbol{\omega})}[\widehat{\mathcal{L}}] = \mathcal{L}$. In appendix C we evaluate the KL divergence between the approximating distribution and the prior analytically, which results in a term similar to L_2 regularisation.

Dropout can be seen as a proxy to this last objective. Evaluating the last objective is identical to performing a stochastic forward pass through the deep model, where each weight row is dropped with probability determined by the magnitude of the row. Gaussian noise is added to rows which are not dropped. Apart from the added noise, this is similar to dropout, which sets each weight row to zero with a certain probability. But unlike dropout, the probability of a row being dropped is determined by the data[7]. Performing a grid-search over the dropout probabilities mimics this to a certain extent.

Remark (Dropout's probability as a model parameter)**.** Note that with this spike and slab prior, the dropout probability $p = \alpha(\mathbf{M}, \boldsymbol{\sigma}, f)/(\alpha(\mathbf{M}, \boldsymbol{\sigma}, f) + 1)$ is not a variational parameter and cannot be optimised directly, but only by changing $\mathbf{M}$ and f. However, changing the prior probability f changes the model and hence the model evidence. This means that for a fixed $\mathbf{M}$ if we were to change the posterior dropout probability p (which depends on the variational parameter $\mathbf{M}$ and model hyper parameter f) we would effectively be *changing the model evidence*—hence

[7]Note that $\frac{\alpha}{\alpha+1} \neq f$ due to the second term in eq. (6.8).

obtain bounds to different constant quantities! This can be seen as evidence towards hypothesis 4 in §6.4.

This approximate inference cannot be easily implemented because the parameter α depends on the exponent of $-\mathbf{m}^T\mathbf{m}$ which can be numerically unstable (as it decreases to zero very quickly). We leave further work on this to future research. Concentrating on the previous prior then, we next explore the dropout probability's effects on the model's epistemic uncertainty.

6.7 Epistemic, Aleatoric, and Predictive uncertainties

I thank Jiri Hron for contributing the code for the Concrete distribution used in this section.

We finish this chapter with a more philosophical discussion of the different types of uncertainty available to us, and ground our discussion in the development of new tools to better understand these. In section 1.2 we discussed the different types of uncertainty encountered in Bayesian modelling: epistemic uncertainty which captures our ignorance about the models most suitable to explain our data, aleatoric uncertainty which captures noise inherent in the environment, and predictive uncertainty which conveys the model's uncertainty in its output.

Epistemic uncertainty reduces as the amount of observed data increases—hence its alternative name "reducible uncertainty". When dealing with models over functions, this uncertainty can be captured through the range of possible functions and the probability given to each function. This uncertainty is often depicted by generating function realisations from our distribution and estimating the variance in the set of functions (for example over a finite input set $\mathbf{X}$). Aleatoric uncertainty captures noise sources such as measurement noise—noises which cannot be explained away even if more data were available (although this uncertainty *can* be reduced through the use of higher precision sensors for example). This uncertainty is often modelled as part of the likelihood, at the top of the model, where we place some noise corruption process on the model output. Gaussian corrupting noise is often assumed in regression, although other noise sources are popular as well such as Laplace noise. By inferring the Gaussian likelihood's precision parameter τ for example we can estimate the amount of aleatoric noise inherent in the data.

It can be difficult to distinguish different types of noise in a single model though, and in section 4.6 we proposed a possible model to do this. In that model we learnt both the aleatoric noise by optimising the (per point) model precision, and captured the epistemic uncertainty using dropout through a grid-search over the dropout probability (note that we could have searched over the model precision as well instead of optimising it, and indeed standard practice in the field of deep learning is to grid-search over it indirectly as part of the NN's weight decay).

Combining both types of uncertainty gives us the predictive uncertainty—the model's confidence in its prediction taking into account noise it can explain away and noise it cannot. This uncertainty is often obtained by generating multiple functions from our model and corrupting them with noise (with precision τ). Calculating the variance of these outputs on multiple inputs of interest we obtain the model's predictive uncertainty. This uncertainty has different properties for different inputs. Inputs near the training data will have a smaller epistemic uncertainty component, while inputs far away from the training data will have higher epistemic uncertainty. Similarly, some parts of the input space might have larger aleatoric uncertainty than others, with these inputs producing larger measurement error for example. These different types of uncertainty are of great importance in fields such as AI safety [Amodei et al., 2016] and autonomous decision making, where the model's epistemic uncertainty can be used to avoid making uninformed decisions with potentially life-threatening implications.

When using dropout NN (or any other SRT), we need to optimise over both the dropout probability p and the model weight decay parameters λ. This is in order to find the epistemic uncertainty and aleatoric uncertainty, respectively. This optimisation can be done by performing a grid-search over both quantities (which we performed in section 4.3 for example w.r.t. validation log-likelihood). But one of the difficulties with the approach above is that grid-searching over both parameters can be expensive and time consuming, especially when done with large models. Even worse, when operating in a continuous learning setting such as reinforcement learning, the model should collapse its epistemic uncertainty as it collects more data. This means that the data has to be set-aside such that a new model could be trained with a smaller dropout probability when the dataset is large enough. This is infeasible in many RL tasks. Instead, the model precision and dropout probability parameters can be optimised with gradient methods, where we seek to minimise some objective w.r.t. to these parameters.

In section 4.6 we specified a heteroscedastic loss which led to the optimisation objective

$$\widehat{\mathcal{L}}_{\text{dropout}}(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}, \tau) := \frac{1}{M} \sum_{i \in S} E^{\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i, \mathbf{b}}(\mathbf{x}_i, \mathbf{y}_i) + \lambda_1 ||\mathbf{M}_1||^2 + \lambda_2 ||\mathbf{M}_2||^2 + \lambda_3 ||\mathbf{b}||^2,$$

$$E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x},\mathbf{y}) := \frac{\tau}{2}\left|\left|\mathbf{y} - \mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\right|\right|_2^2 - \frac{D}{2}\log\tau$$
$$\lambda_i = \frac{l_i^2(1-p_i)}{2N}$$

with D the output dimensionality, and where we optimise the objective w.r.t. τ as well as the weights to find the model's aleatoric uncertainty. This objective was derived from eq. (3.7) with a Gaussian likelihood with precision τ, and written in the form of eq. (3.9). Compared to eq. (3.9), our optimisation objective is scaled by τ and has an added "regularisation" term $-\log\tau$, a term derived from the likelihood definition which would be omitted when the precision need not be optimised (since τ is constant w.r.t. model weights $\mathbf{W}$).

Similar to this, we can optimise our objective in eq. (3.7) w.r.t. the dropout probability to find the epistemic uncertainty as well. The KL contribution term between the approximate posterior and the prior depends on p through the entropy (eq. (6.6)):

$$\mathcal{H}(p) := -p\log p - (1-p)\log(1-p)$$

which results in an added *dropout regulariser* term to our objective in eq. (3.9). To obtain the KL contribution term between the approximate posterior and the prior *over all weights* we sum the KL per weight row in each weight matrix. The term is then divided by the number of training points in the objective (§3.2.3) to obtain the dropout regulariser in the form of eq. (3.9):

$$\frac{1}{N}\mathrm{KL}(q(\mathbf{W})||p(\mathbf{W})) \propto \frac{l^2(1-p)}{2N}||\mathbf{M}||^2 - \frac{K}{N}\mathcal{H}(p)$$

with K the input dimensionality of the layer and N the number of data points. This regularisation term depends on the dropout probability p alone, which means that the term is constant w.r.t. model weights (thus omitted when the dropout probability is not optimised, but crucial when it is optimised). The entropy of the Bernoulli random variable with probability p pushes the dropout probability towards 0.5—the highest it can attain. The scaling of the regularisation term means that large models will push the dropout probability towards 0.5 much more than smaller models, but as the amount of data increases the dropout probability will be pushed towards 0.

An issue arises when gradient methods are used for the dropout probability optimisation. The Bernoulli distribution which is used in dropout NNs in the expected log

likelihood term (first term in equation (3.7)):

$$\widehat{\mathcal{L}}_{\text{MC}}(\theta) = -\frac{N}{M}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i)) + \text{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$$

depends on p. The Bernoulli distribution is non-differentiable with respect to its parameter p, which means that the pathwise derivative estimator cannot be used with it (forcing us to use the high variance score function estimator). Instead we use the Concrete distribution relaxation [Jang et al., 2016; Maddison et al., 2016] to approximate the Bernoulli distribution when generating the dropout masks. Instead of sampling our dropout masks from the Bernoulli distribution (generating zeros and ones) we sample realisations from the Concrete distribution with temperature $t = 1/10$ which results in masks with values in the interval $[0, 1]$. This distribution concentrates most mass on the boundaries of the interval 0 and 1. In fact, for the one dimensional case here with the Bernoulli distribution, the Concrete distribution relaxation $\tilde{\mathbf{z}}$ of the Bernoulli random variable $\mathbf{z}$ reduces to a simple sigmoid distribution which has a convenient parametrisation:

$$\tilde{\mathbf{z}} = \text{sigmoid}\left(\frac{1}{t}\cdot\Big(\log p - \log(1-p) + \log u - \log(1-u)\Big)\right)$$

with uniform $u \sim \text{Unif}(0, 1)$.

These tools allow us to find both epistemic and aleatoric uncertainties with ease. To assess how different uncertainties behave with different amounts of data, we optimise both the dropout probability p as well as the (per point) model precision τ. We generated synthetic data from the function $y = 2x + 8 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$ (i.e. corrupting the observations with noise with a fixed standard deviation 1), creating datasets increasing in size ranging from 10 data points (example in figure 6.9) up to $10,000$ data points (example in figure 6.10). We used models with three hidden layers of size 1024 and relu non-linearities, and repeated each experiment three times, averaging the experiments' results. Figure 6.11 shows the epistemic uncertainty (in standard deviation) decreasing as the amount of data increases. This uncertainty was computed by generating multiple function draws and evaluating the functions over a test set generated from the same data distribution. Figure 6.12 shows that the model obtains an increasingly improved estimate to the model precision (aleatoric uncertainty) as more data is given. Finally, figure 6.13 shows the predictive uncertainty obtained by combining the variances of both plots above. This uncertainty seems to converge towards a constant trend.

Lastly, the optimised dropout probabilities corresponding to the various dataset sizes are given in figure 6.14. As can be seen, the optimal dropout probability in each layer decreases as more data is observed, starting from the smallest dataset with near 0.5 probabilities in all layers but the first (input layer, #1), and converging to values ranging between 0.1 and 0.2 when 10,000 data points are given to the model. Further, the dropout probability of the first layer converges to a near-zero value for all data sizes, supporting our observations in §4.2.1. This empirical observation further confirms with our theoretical analysis in section 6.3.

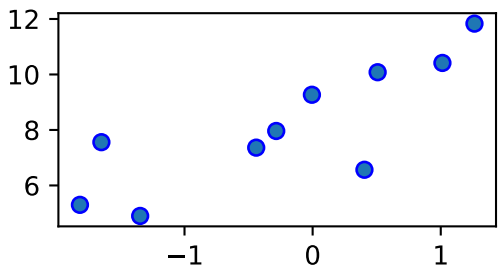


Fig. 6.9 Example dataset with 10 data points.

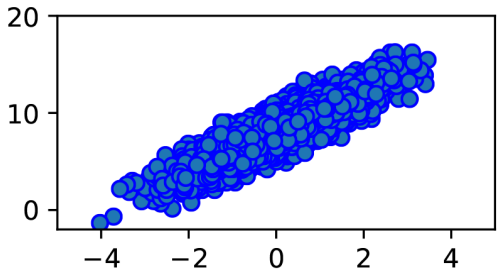


Fig. 6.10 Example dataset with $10,000$ data points.

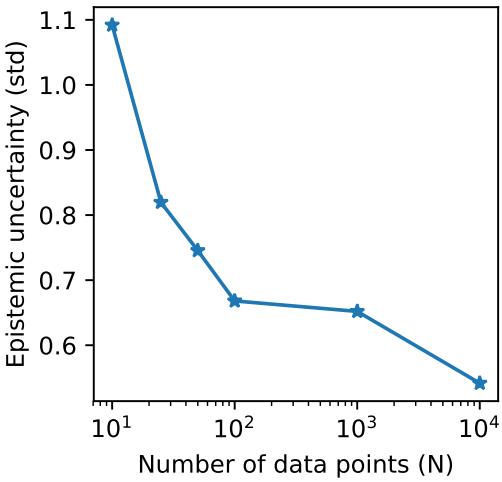


Fig. 6.11 Epistemic uncertainty (in std) as the number of data points increases.

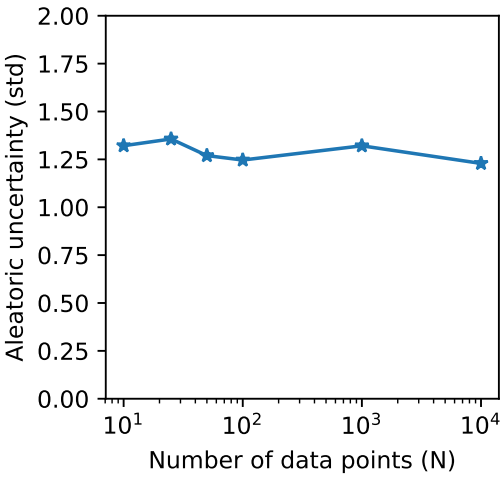


Fig. 6.12 Aleatoric uncertainty (in std) as the number of data points increases.

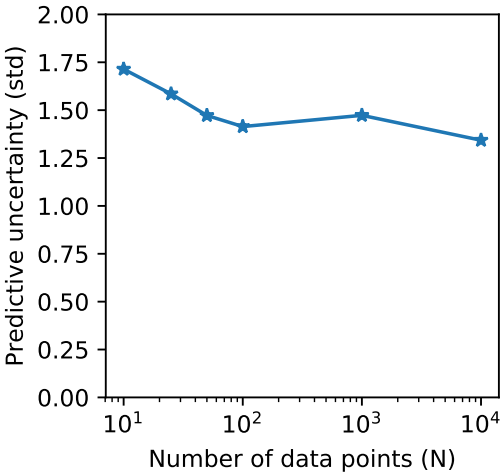


Fig. 6.13 Predictive uncertainty (in std) as the number of data points increases.

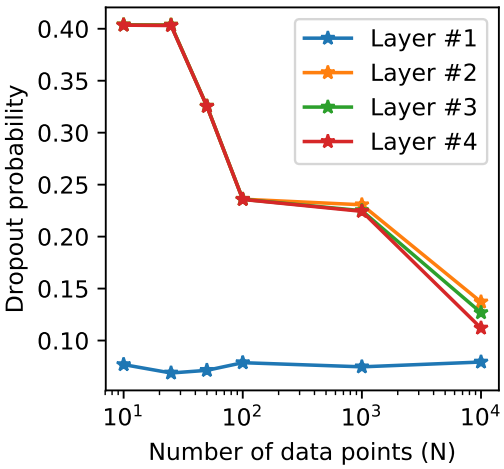


Fig. 6.14 Optimised dropout probability values (per layer) as the number of data points increases.

Chapter 7

Future Research

Bayesian modelling and deep learning have traditionally been regarded as fairly antipodal to each other: one pushed forward by theoreticians, while the other by practitioners. Bayesian modelling is based on the vast theory of Bayesian statistics, in which we aim to capture the processes assumed to have generated our data. This often results in interpretable models that can explain the data well, at least when we can perform inference in the models. Deep learning on the other hand is mostly driven by pragmatic developments of tractable models, and has fundamentally affected the way machine learning is used in real-world applications. But unlike Bayesian modelling, deep learning lacks a solid mathematical formalism, and many developments are very weakly mathematically justified. Deep learning's success is often explained by various metaphors which do not shed much light on the reasons models are built in certain ways.

In dropout for example the network's units are multiplied by Bernoulli random variables. This slows down training but circumvents over-fitting and improves model accuracy considerably. Such techniques have had tremendous success in deep learning and are used in almost all modern models [Srivastava et al., 2014]. But why do these work so well? In [Srivastava et al., 2014] dropout is suggested to work well following a sexual reproduction metaphor. But then why would multiplying a network's units by a Gaussian distribution $\mathcal{N}(1, 1)$ instead of Bernoulli random variables result in the same model performance?

Perhaps surprisingly, we gave a possible answer to the questions above using Bayesian statistics and variational inference. We have shown that dropout in deep NNs can be cast as a variational approximation in Bayesian neural networks. The implications of this result are far-reaching. Since many modern deep learning tools make use of some form of stochastic regularisation, this means that many modern deep learning systems perform approximate Bayesian inference, capturing the stochastic processes underlying

the observed data. This link opens the vast literature of Bayesian statistics to deep learning, explaining many deep learning phenomena with a mathematically rigorous theory, and extending existing tools in a principled way. We can use variational inference in deep learning, combining deep learning tools and Bayesian models in a compositional fashion. We can even assess model uncertainty in deep learning and build interpretable deep learning tools.

There are many open leads for future research:

Deep learning can be extended in a principled way. Understanding the underlying principles leading to good models allows us to improve upon them. For example, alternative approximating distributions to the ones discussed above would translate to new stochastic regularisation techniques. These can range from simple distributions to complex ones. Model compression can be achieved by forcing weights to take values from a discrete distribution over a continuous base measure of "hyper-weights" for example.

Deep learning uncertainty. Initial research above assessed the performance of dropout in terms of the predictive mean and variance. Even though the Bernoulli approximating distribution is a crude one, the model outperformed its equivalents in the field. But different non-linearity–regularisation combinations correspond to different Gaussian process covariance functions, and these have different characteristics in terms of the predictive uncertainty. Understanding the behaviour of different model structures and the resulting predictive mean and variance are of crucial importance to practitioners making use of dropout's uncertainty.

Deep learning can make use of Bayesian models. A much more interesting application of the theory above is the combination of techniques from the two fields: deep learning and Bayesian modelling. Bayesian models, often used in data analysis, strive to describe data in an interpretable way—a property that most deep learning models lack. Using the theory above we can combine deep learning with interpretable Bayesian models and build hybrid models that draw from the best that both worlds have to offer. For example, in the fields of computational linguistics and language processing we often look for models that can explain the linguistic phenomena underlying our data. Current deep learning methods work well modelling the data and have improved considerably on previous research—partly due to their tractability and ability to go beyond the bag-of-words assumptions. But the models are extremely opaque and have not been able to explain the linguistic principles they use. Interleaving Bayesian models with deep ones we could answer many of these open problems.

Bayesian models can make use of deep learning. The field of Bayesian modelling can benefit immensely from the simple data representations obtained from deep learning models. Sequence data, image data, high dimensional data—these are structures that traditional Bayesian techniques find difficult to handle. Many unjustified simplifying assumptions are often used with these data structures: bag-of-words assumptions, reducing the dimensionality of the data, etc. By interpreting deep learning models as Bayesian ones, we can combine these easily and in a principled way. Further, models can be built in a compositional fashion by propagating derivatives, forming small building blocks that can be assembled together by non-experts.

Unsupervised deep learning. One last problem discussed here is the design of unsupervised models. Bayesian statistics lends itself naturally to data analysis and unsupervised data modelling. With the Bayesian interpretation of modern deep learning new horizons open and new tools become available to solve this laborious task.

It is my hope that the framework presented in this thesis will lay the foundations of a new and exciting field of study, combining modern deep learning and Bayesian techniques in a principled and practical way.

References

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

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mane. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

C Angermueller and O Stegle. Multi-task deep neural network to predict CpG methylation profiles from low-coverage sequencing data. In *NIPS MLCB workshop*, 2015.

O Anjos, C Iglesias, F Peres, J Martínez, Á García, and J Taboada. Neural networks applied to discriminate botanical origin of honeys. *Food chemistry*, 175:128–136, 2015.

Christopher Atkeson and Juan Santamaria. A comparison of direct and model-based reinforcement learning. In *In International Conference on Robotics and Automation*. Citeseer, 1997.

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

P Baldi, P Sadowski, and D Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5, 2014.

Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

David Barber and Christopher M Bishop. Ensemble learning in Bayesian neural networks. *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES*, 168:215–238, 1998.

Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*, 2013.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

S Bergmann, S Stelzer, and S Strassburger. On the use of artificial neural networks in simulation-based manufacturing control. *Journal of Simulation*, 8(1):76–90, 2014.

James Bergstra et al. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Chris M Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Where to apply dropout in recurrent neural networks for handwriting recognition? In *ICDAR*. IEEE, 2015.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, 2015.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Thang D Bui, Daniel Hernández-Lobato, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Deep Gaussian processes for regression using approximate expectation propagation. *ICML*, 2016.

Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Dropout distillation. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 99–107, 2016.

Theophilos Cacoullos. On upper and lower bounds for the variance of a function of a random variable. *The Annals of Probability*, pages 799–809, 1982.

Hugh Chipman. Bayesian variable selection with related predictors.

Kyunghyun Cho et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP*, Doha, Qatar, October 2014. ACL.

François Chollet. Keras, 2015. URL https://github.com/fchollet/keras. GitHub repository.

David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 1996.

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

Marc Deisenroth and Carl Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Marc Deisenroth, Dieter Fox, and Carl Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(2):408–423, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

John Denker and Yann LeCun. Transforming neural-net output levels to probability distributions. In *Advances in Neural Information Processing Systems 3*. Citeseer, 1991.

John Denker, Daniel Schwartz, Ben Wittner, Sara Solla, Richard Howard, Lawrence Jackel, and John Hopfield. Large automatic learning, rule extraction, and generalization. *Complex systems*, 1(5):877–922, 1987.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

Linton G Freeman. Elementary applied statistics, 1965.

Michael C. Fu. Chapter 19 gradient estimation. In Shane G. Henderson and Barry L. Nelson, editors, *Simulation*, volume 13 of *Handbooks in Operations Research and Management Science*, pages 575 – 616. Elsevier, 2006.

Antonino Furnari, Giovanni Maria Farinella, and Sebastiano Battiato. Segmenting egocentric videos to highlight personal locations of interest. 2016.

Yarin Gal. A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*, 2015.

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv:1506.02158*, 2015a.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015b.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv:1506.02142*, 2015c.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Appendix. *arXiv:1506.02157*, 2015d.

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. *ICLR workshop track*, 2016a.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *NIPS*, 2016b.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *ICML*, 2016c.

Yarin Gal and Richard Turner. Improving the Gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015.

Yarin Gal, Mark van der Wilk, and Carl Rasmussen. Distributed variational inference in sparse Gaussian process regression and latent variable models. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3257–3265. Curran Associates, Inc., 2014.

Yarin Gal, Rowan McAllister, and Carl E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. *Data-Efficient Machine Learning workshop, ICML, April*, 2016.

Carl Friedrich Gauss. *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. 1809.

Edward I George and Robert E McCulloch. Variable selection via gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993.

Edward I George and Robert E McCulloch. Approaches for Bayesian variable selection. *Statistica sinica*, pages 339–373, 1997.

J.D Gergonne. Application de la méthode des moindre quarrés a l'interpolation des suites. *Annales des Math Pures et Appl*, 6:242–252, 1815.

Z Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521 (7553), 2015.

Z. Ghahramani and H. Attias. Online variational Bayesian learning. Slides from talk presented at NIPS 2000 Workshop on Online learning, 2000.

Ryan J Giordano, Tamara Broderick, and Michael I Jordan. Linear response methods for accurate covariance estimates from mean field variational Bayes. In *Advances in Neural Information Processing Systems*, pages 1441–1449, 2015.

Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer Science & Business Media, 2013.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

Ian J Goodfellow, Aaron Courville, and Yoshua Bengio. Spike-and-slab sparse coding for unsupervised feature discovery. *arXiv preprint arXiv:1201.3382*, 2012.

Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*. IEEE, 2013.

Alex Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. *ICML*, 2016.

Jose Miguel Hernandez-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *ICML*, 2015.

José Miguel Hernández-Lobato, Yingzhen Li, Daniel Hernández-Lobato, Thang Bui, and Richard E Turner. Black-box alpha divergence minimization. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1511–1520, 2016.

S Herzog and D Ostwald. Experimental biology: Sometimes Bayesian statistics are better. *Nature*, 494, 2013.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*, pages 5–13. ACM, 1993.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *JMLR*, 2013.

Alex Holub, Pietro Perona, and Michael C Burl. Entropy-based active learning for object recognition. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

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

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

Hemant Ishwaran and J Sunil Rao. Spike and slab variable selection: frequentist and Bayesian strategies. *Annals of Statistics*, pages 730–773, 2005.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *Bayesian Deep Learning workshop, NIPS*, 2016.

Chuanyi Ji, Robert R Snapp, and Demetri Psaltis. Generalizing smoothness constraints from discrete samples. *Neural Computation*, 2(2):188–197, 1990.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2): 183–233, 1999.

Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2372–2379. IEEE, 2009.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013.

Michael Kampffmeyer, Arnt-Borre Salberg, and Robert Jenssen. Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016.

A Karpathy et al. A Javascript implementation of neural networks. https://github.com/karpathy/convnetjs, 2014–2015.

C.D. Keeling, T.P. Whorf, and the Carbon Dioxide Research Group. Atmospheric CO2 concentrations (ppmv) derived from in situ air samples collected at Mauna Loa Observatory, Hawaii. Scripps Institution of Oceanography (SIO), University of California, La Jolla, California USA 92093-0444, June 2004.

Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4762–4769. IEEE, 2016.

Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma and Max Welling. Stochastic gradient VB and the variational auto-encoder. *2nd International Conference on Learning Representationsm (ICLR)*, 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *NIPS*. Curran Associates, Inc., 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing RNNs by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

M Krzywinski and N Altman. Points of significance: Importance of being uncertain. *Nature methods*, 10(9), 2013.

Solomon Kullback. *Information theory and statistics*. John Wiley & Sons, 1959.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Quoc V Le, Alex J Smola, and Stéphane Canu. Heteroscedastic Gaussian process regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 489–496. ACM, 2005.

J. Lean. Solar irradiance reconstruction. NOAA/NGDC Paleoclimatology Program, Boulder CO, USA, 2004. IGBP PAGES/World Data Center for Paleoclimatology. Data Contribution Series 2004-035.

Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 1998.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

Adrien Marie Legendre. *Nouvelles Methodes pour la Determination des Orbites des Come'tes*. Paris, 1805.

Nicholas Léonard, Sagar Waghmare, and Yang Wang. RNN: Recurrent library for Torch. *arXiv preprint arXiv:1511.07889*, 2015.

Xin Li and Yuhong Guo. Adaptive active learning for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 859–866, 2013.

Yingzhen Li and Richard E Turner. Variational inference with r\'enyi divergence. *arXiv preprint arXiv:1602.02311*, 2016.

Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

O Linda, T Vollmer, and M Manic. Neural network based intrusion detection system for critical infrastructures. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*. IEEE, 2009.

Christos Louizos. Smart regularization of deep architectures, 2015.

David MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992a.

David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992b.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete distribution: A continuous relaxation of discrete random variables. In *Bayesian Deep Learning workshop, NIPS*, 2016.

David Madigan and Adrian E Raftery. Model selection and accounting for model uncertainty in graphical models using Occam's window. *Journal of the American Statistical Association*, 89(428):1535–1546, 1994.

Shin-ichi Maeda. A Bayesian encourages dropout. *arXiv preprint arXiv:1412.7003*, 2014.

Andrew McHutchon. *Nonlinear modelling and control using Gaussian processes*. PhD thesis, University of Cambridge, 2014.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Tom Minka. Divergence measures and message passing. Technical report, Microsoft Research, 2005.

Toby J Mitchell and John J Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032, 1988.

V Mnih, K Kavukcuoglu, D Silver, A A Rusu, J Veness, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Taesup Moon, Heeyoul Choi, Hoshik Lee, and Inchul Song. RnnDrop: A Novel Dropout for RNNs in ASR. In *ASRU Workshop*, 2015.

Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

Radford M Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

Isaac Newton. *Philosophiae naturalis principia mathematica*, volume Adv.b.39.1. Jussu Societatis Regiæ ac Typis Joseph Streater, Londini, 1687. (in Latin).

NHTSA. PE 16-007. Technical report, U.S. Department of Transportation, National Highway Traffic Safety Administration, Jan 2017. Tesla Crash Preliminary Evaluation Report.

Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992.

Regina Nuzzo. Statistical errors. *Nature*, 506(13):150–152, 2014.

Manfred Opper and Cédric Archambeau. The variational Gaussian approximation revisited. *Neural Computation*, 21(3):786–792, 2009.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances In Neural Information Processing Systems*, pages 4026–4034, 2016.

Marius Pachitariu and Maneesh Sahani. Regularization and nonlinearities for neural language models: when are they needed? *arXiv preprint arXiv:1301.5650*, 2013.

John Paisley, David Blei, and Michael Jordan. Variational Bayesian inference with stochastic search. *ICML*, 2012.

Vu Pham, Theodore Bluche, Christopher Kermorvant, and Jerome Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *ICFHR*. IEEE, 2014.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2006. ISBN 026218253X.

Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Donald B Rubin. The Bayesian bootstrap. *The annals of statistics*, 9(1):130–134, 1981.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

Masa-Aki Sato. Online model selection based on the variational Bayes. *Neural Computation*, 13(7):1649–1681, 2001.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

Hilary L Seal. Studies in the history of probability and statistics. XXIX – The discovery of the method of least squares. *Biometrika*, 54(1-2):1–24, 1967.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Edinburgh neural machine translation systems for wmt 16. In *Proceedings of the First Conference on Machine Translation*, pages 371–376, Berlin, Germany, August 2016. Association for Computational Linguistics.

Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52 (55-66):11, 2010.

Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. *NIPS*, 2016.

Kirstine Smith. On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of the observations. *Biometrika*, 12:1–85, 1918.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2005.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Jasper Snoek et al. Spearmint. https://github.com/JasperSnoek/spearmint, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *INTERSPEECH*, 2012.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Richard Sutton and Andrew Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294, 1933.

Naftali Tishby, Esther Levin, and Sara A Solla. Consistent inference of probabilities in layered networks: Predictions and generalizations. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 403–409. IEEE, 1989.

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1971–1979, 2014.

Michalis K Titsias and Miguel Lázaro-Gredilla. Spike and slab variational inference for multi-task and multiple kernel learning. In *Advances in neural information processing systems*, pages 2339–2347, 2011.

D Trafimow and M Marks. Editorial. *Basic and Applied Social Psychology*, 37(1), 2015.

R. E. Turner and M. Sahani. Two problems with variational expectation maximisation for time-series models. In *Bayesian Time series models*, chapter 5, pages 109–130. Cambridge University Press, 2011.

Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, pages 351–359, 2013.

L Wan, M Zeiler, S Zhang, Y LeCun, and R Fergus. Regularization of neural networks using dropconnect. In *ICML-13*, 2013.

S Wang and C Manning. Fast dropout training. *ICML*, 2013.

Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems*, pages 875–882, 1991.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

Christopher KI Williams. Computing with infinite networks. *Advances in neural information processing systems*, pages 295–301, 1997.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

John Winn and Christopher M Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(Apr):661–694, 2005.

Xiao Yang, Roland Kwitt, and Marc Niethammer. Fast predictive image registration. *arXiv preprint arXiv:1607.02504*, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. In *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.

Appendix A

KL condition

We show that in the dropout case, the KL condition (eq. (3.12)) holds for a large enough number of hidden units when we specify the model prior to be a product of uncorrelated Gaussian distributions over each weight[1]:

$$p(\boldsymbol{\omega}) = \prod_{i=1}^{L} p(\mathbf{W}_i) = \prod_{i=1}^{L} \mathcal{MN}(\mathbf{W}_i; 0, \mathbf{I}/l_i^2, \mathbf{I}).$$

We set the approximating distribution to be $q_\theta(\boldsymbol{\omega}) = \int q_\theta(\boldsymbol{\omega}|\boldsymbol{\epsilon})p(\boldsymbol{\epsilon})\mathrm{d}\boldsymbol{\epsilon}$ where $q_\theta(\boldsymbol{\omega}|\boldsymbol{\epsilon}) = \delta(\boldsymbol{\omega} - g(\theta, \boldsymbol{\epsilon}))$, with $g(\theta, \boldsymbol{\epsilon}) = \{\mathrm{diag}(\boldsymbol{\epsilon}_1)\mathbf{M}_1, \mathrm{diag}(\boldsymbol{\epsilon}_2)\mathbf{M}_2, \mathbf{b}\}$, $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$, and $p(\boldsymbol{\epsilon}_i)$ defined as a product of Bernoulli distributions ($\boldsymbol{\epsilon}_i$ is a vector of draws from the Bernoulli distribution). Since we assumed $q_\theta(\boldsymbol{\omega})$ to factorise over the layers and over the rows of each weight matrix, we have

$$\mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) = \sum_{i,k} \mathrm{KL}(q_{\theta_{i,k}}(\mathbf{w}_{i,k})||p(\mathbf{w}_{i,k}))$$

with i summing over the layers and k summing over the rows in each layers' weight matrix.

We approximate each $q_{\theta_{i,k}}(\mathbf{w}_{i,k}|\boldsymbol{\epsilon}) = \delta(\mathbf{w}_{i,k} - g(\theta_{i,k}, \epsilon_{i,k}))$ as a narrow Gaussian with a small standard deviation $\Sigma = \sigma^2 I$. This means that marginally $q_{\theta_{i,k}}(\mathbf{w}_{i,k})$ is a mixture of two Gaussians with small standard deviations, and one component fixed at zero. For large enough models, the KL condition follows from this general proposition:

Proposition 4. *Fix $K, L \in \mathbb{N}$, a probability vector $\mathbf{p} = (p_1, ..., p_L)$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{K\times K}$ diagonal positive-definite for $i = 1, ..., L$, with the elements of each $\boldsymbol{\Sigma}_i$ not dependent on*

[1]Here $\mathcal{MN}(0, \mathbf{I}, \mathbf{I})$ is the standard matrix Gaussian distribution.

K. Let

$$q(\mathbf{x}) = \sum_{i=1}^{L} p_i \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

be a mixture of Gaussians with L components and $\boldsymbol{\mu}_i \in \mathbb{R}^K$, let $p(\mathbf{x}) = \mathcal{N}(0, \mathbf{I}_K)$, and further assume that $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j \sim \mathcal{N}(0, I)$ for all i, j.

The KL divergence between $q(\mathbf{x})$ and $p(\mathbf{x})$ can be approximated as:

$$KL(q(\mathbf{x})||p(\mathbf{x})) \approx \sum_{i=1}^{L} \frac{p_i}{2}\left(\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + tr(\boldsymbol{\Sigma}_i) - K(1 + \log 2\pi) - \log|\boldsymbol{\Sigma}_i|\right) - \mathcal{H}(\mathbf{p}) \tag{A.1}$$

with $\mathcal{H}(\mathbf{p}) := -\sum_{i=1}^{L} p_i \log p_i$ for large enough K.

Before we prove the proposition, we observe that a direct result from it is the following:

Corollary 2. The KL condition (eq. (3.12)) holds for a large enough number of hidden units when we specify the model prior to be

$$p(\boldsymbol{\omega}) = \prod_{i=1}^{L} p(\mathbf{W}_i) = \prod_{i=1}^{L} \mathcal{MN}(\mathbf{W}_i; 0, \mathbf{I}/l_i^2, \mathbf{I})$$

and the approximating distribution to be a *dropout variational distribution.*

Proof.

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{m}_{i,k}} \mathrm{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) &= \frac{\partial}{\partial \mathbf{m}_{i,k}} \mathrm{KL}(q_{\theta_{i,k}}(\mathbf{w}_{i,k})||p(\mathbf{w}_{i,k})) \\
&\approx \frac{(1-p_i)l_i^2}{2} \frac{\partial}{\partial \mathbf{m}_{i,k}} \mathbf{m}_{i,k}^T \mathbf{m}_{i,k} \\
&= \frac{\partial}{\partial \mathbf{m}_{i,k}} N\tau(\lambda_1||\mathbf{M}_1||^2 + \lambda_2||\mathbf{M}_2||^2 + \lambda_3||\mathbf{b}||^2)
\end{aligned}$$

for $\lambda_i = \frac{(1-p_i)l_i^2}{2N\tau}$. □

Next we prove proposition 4.

Proof. We have

$$\begin{aligned}
\mathrm{KL}(q(\mathbf{x})||p(\mathbf{x})) &= \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \mathrm{d}\mathbf{x} \\
&= \int q(\mathbf{x}) \log q(\mathbf{x}) \mathrm{d}\mathbf{x} - \int q(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}
\end{aligned}$$

$$= -\mathcal{H}(q(\mathbf{x})) - \int q(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{A.2}$$

—a sum of the entropy of $q(\mathbf{x})$ ($\mathcal{H}(q(\mathbf{x}))$) and the expected log probability of $\mathbf{x}$. The expected log probability can be evaluated analytically, but the entropy term has to be approximated.

We begin by approximating the entropy term. We write

$$\begin{aligned}\mathcal{H}(q(\mathbf{x})) &= -\sum_{i=1}^{L} p_i \int \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \log q(\mathbf{x}) \mathrm{d}\mathbf{x} \\ &= -\sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \mathbf{I}) \log q(\boldsymbol{\mu}_i + \mathbf{L}_i \boldsymbol{\epsilon}_i) \mathrm{d}\boldsymbol{\epsilon}_i\end{aligned}$$

using a change of variables $\mathbf{x} = \boldsymbol{\mu}_i + \mathbf{L}_i \boldsymbol{\epsilon}_i$ with $\mathbf{L}_i \mathbf{L}_i^T = \boldsymbol{\Sigma}_i$ and $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, I)$.

Now, the term inside the logarithm can be written as

$$\begin{aligned}q(\boldsymbol{\mu}_i + \mathbf{L}_i \boldsymbol{\epsilon}_i) &= \sum_{j=1}^{L} p_i \mathcal{N}(\boldsymbol{\mu}_i + \mathbf{L}_i \boldsymbol{\epsilon}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \\ &= \sum_{j=1}^{L} p_i (2\pi)^{-K/2} |\boldsymbol{\Sigma}_j|^{-1/2} \exp\Big\{ -\frac{1}{2} ||\boldsymbol{\mu}_j - \boldsymbol{\mu}_i - \mathbf{L}_i \boldsymbol{\epsilon}_i||^2_{\boldsymbol{\Sigma}_j} \Big\}\end{aligned}$$

where $||\cdot||_{\boldsymbol{\Sigma}}$ is the Mahalanobis distance. Since $\boldsymbol{\mu}_i, \boldsymbol{\mu}_j$ are assumed to be normally distributed, the quantity $\boldsymbol{\mu}_j - \boldsymbol{\mu}_i - \mathbf{L}_i \boldsymbol{\epsilon}_i$ is also normally distributed[2]. Since the expectation of a generalised χ^2 distribution with K degrees of freedom increases with K, we have that[3] $K \gg 0$ implies that $||\boldsymbol{\mu}_j - \boldsymbol{\mu}_i - \mathbf{L}_i \boldsymbol{\epsilon}_i||^2_{\boldsymbol{\Sigma}_j} \gg 0$ for $i \neq j$ (since the elements of $\boldsymbol{\Sigma}_j$ do not depend on K). Finally, we have for $i = j$ that $||\boldsymbol{\mu}_i - \boldsymbol{\mu}_i - \mathbf{L}_i \boldsymbol{\epsilon}_i||^2_{\boldsymbol{\Sigma}_i} = \boldsymbol{\epsilon}_i^T \mathbf{L}_i^T \mathbf{L}_i^{-T} \mathbf{L}_i^{-1} \mathbf{L}_i \boldsymbol{\epsilon}_i = \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i$. Therefore the last equation can be approximated as

$$q(\boldsymbol{\mu}_i + \mathbf{L}_i \boldsymbol{\epsilon}_i) \approx p_i (2\pi)^{-K/2} |\boldsymbol{\Sigma}_i|^{-1/2} \exp\Big\{ -\frac{1}{2} \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i \Big\}.$$

I.e., in high dimensions the mixture components will not overlap. This gives us

$$\begin{aligned}\mathcal{H}(q(\mathbf{x})) &\approx -\sum_{i=1}^{L} p_i \int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \mathbf{I}) \log \left(p_i (2\pi)^{-K/2} |\boldsymbol{\Sigma}_i|^{-1/2} \exp\Big\{ -\frac{1}{2} \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i \Big\} \right) \mathrm{d}\boldsymbol{\epsilon}_i \\ &= \sum_{i=1}^{L} \frac{p_i}{2} \left(\log |\boldsymbol{\Sigma}_i| + \int \mathcal{N}(\boldsymbol{\epsilon}_i; 0, \mathbf{I}) \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i \mathrm{d}\boldsymbol{\epsilon}_i + K \log 2\pi \right) + \mathcal{H}(\mathbf{p})\end{aligned}$$

[2] With mean zero and variance $\mathrm{Var}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i - \mathbf{L}_i \boldsymbol{\epsilon}_i) = 2I + \boldsymbol{\Sigma}_i$.

[3] To be exact, for diagonal matrices Λ, Δ and $\mathbf{v} \sim \mathcal{N}(0, \Lambda)$, we have $\mathbb{E}[||\mathbf{v}||_\Delta] = \mathbb{E}[\mathbf{v}^T \Delta^{-1} \mathbf{v}] = \sum_{k=1}^{K} \mathbb{E}[\Delta_k^{-1} v_k^2] = \sum_{k=1}^{K} \Delta_k^{-1} \Lambda_k$.

where $\mathcal{H}(\mathbf{p}) := -\sum_{i=1}^{L} p_i \log p_i$. Since $\boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i$ distributes according to a χ^2 distribution, its expectation is K, and the entropy can be approximated as

$$\mathcal{H}(q(\mathbf{x})) \approx \sum_{i=1}^{L} \frac{p_i}{2}\Big(\log|\boldsymbol{\Sigma}_i| + K(1+\log 2\pi)\Big) + \mathcal{H}(\mathbf{p}). \tag{A.3}$$

Next, evaluating the expected log probability term of the KL divergence we get

$$\int q(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x} = \sum_{i=1}^{L} p_i \int \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}$$

for $p(\mathbf{x}) = \mathcal{N}(0, \mathbf{I}_K)$ it is easy to show that

$$\int q(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x} = -\frac{1}{2} \sum_{i=1}^{L} p_i \Big(\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + \mathrm{tr}(\boldsymbol{\Sigma}_i)\Big). \tag{A.4}$$

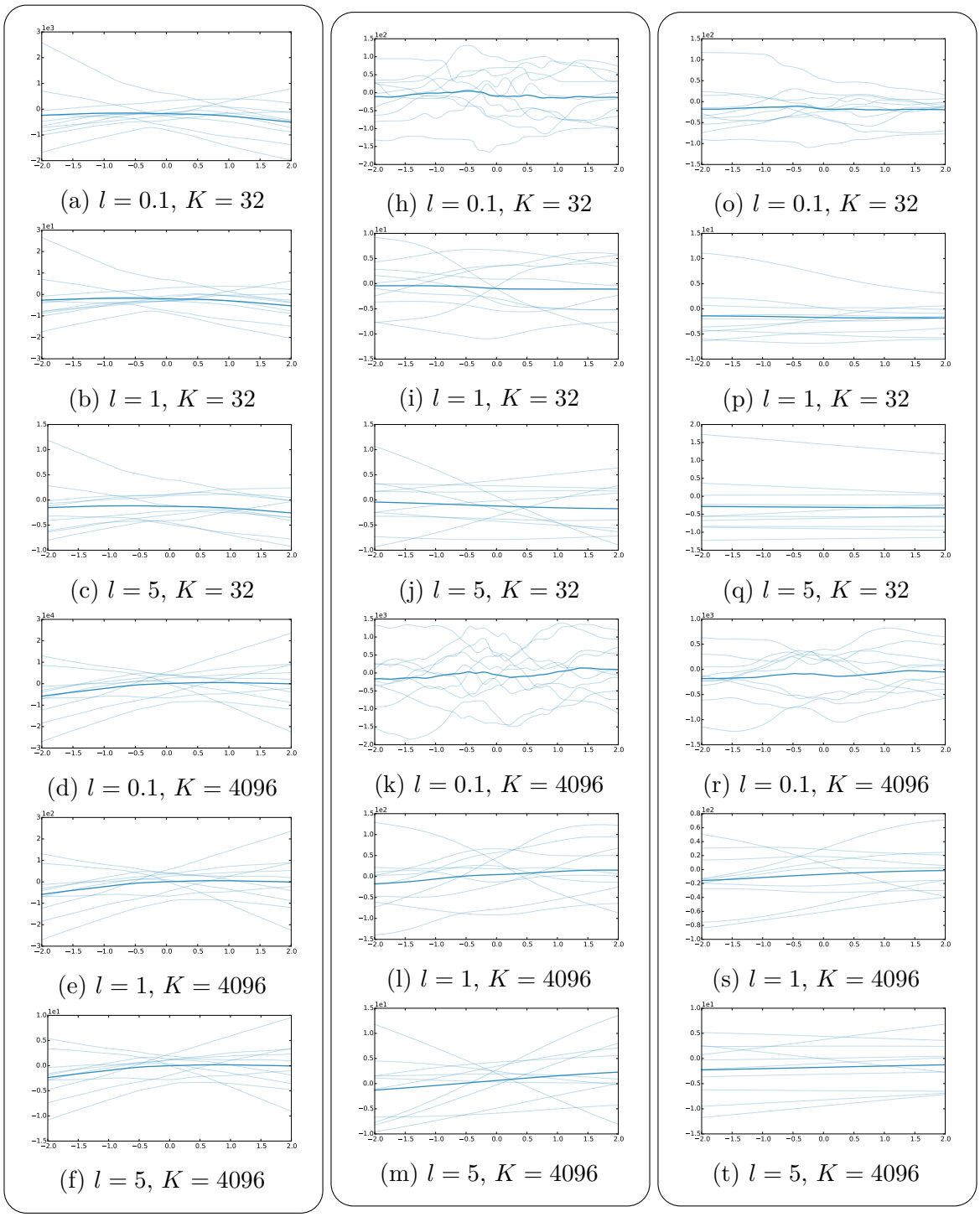
Finally, combining eq. (A.3) and eq. (A.4) as in (A.2) we get:

$$\mathrm{KL}(q(\mathbf{x})||p(\mathbf{x})) \approx \sum_{i=1}^{L} \frac{p_i}{2}\Big(\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + \mathrm{tr}(\boldsymbol{\Sigma}_i) - K(1+\log 2\pi) - \log|\boldsymbol{\Sigma}_i|\Big) - \mathcal{H}(\mathbf{p}),$$

as required to show. □

Appendix B

Figures



(a) $l = 0.1$, $K = 32$ (h) $l = 0.1$, $K = 32$ (o) $l = 0.1$, $K = 32$

(b) $l = 1$, $K = 32$ (i) $l = 1$, $K = 32$ (p) $l = 1$, $K = 32$

(c) $l = 5$, $K = 32$ (j) $l = 5$, $K = 32$ (q) $l = 5$, $K = 32$

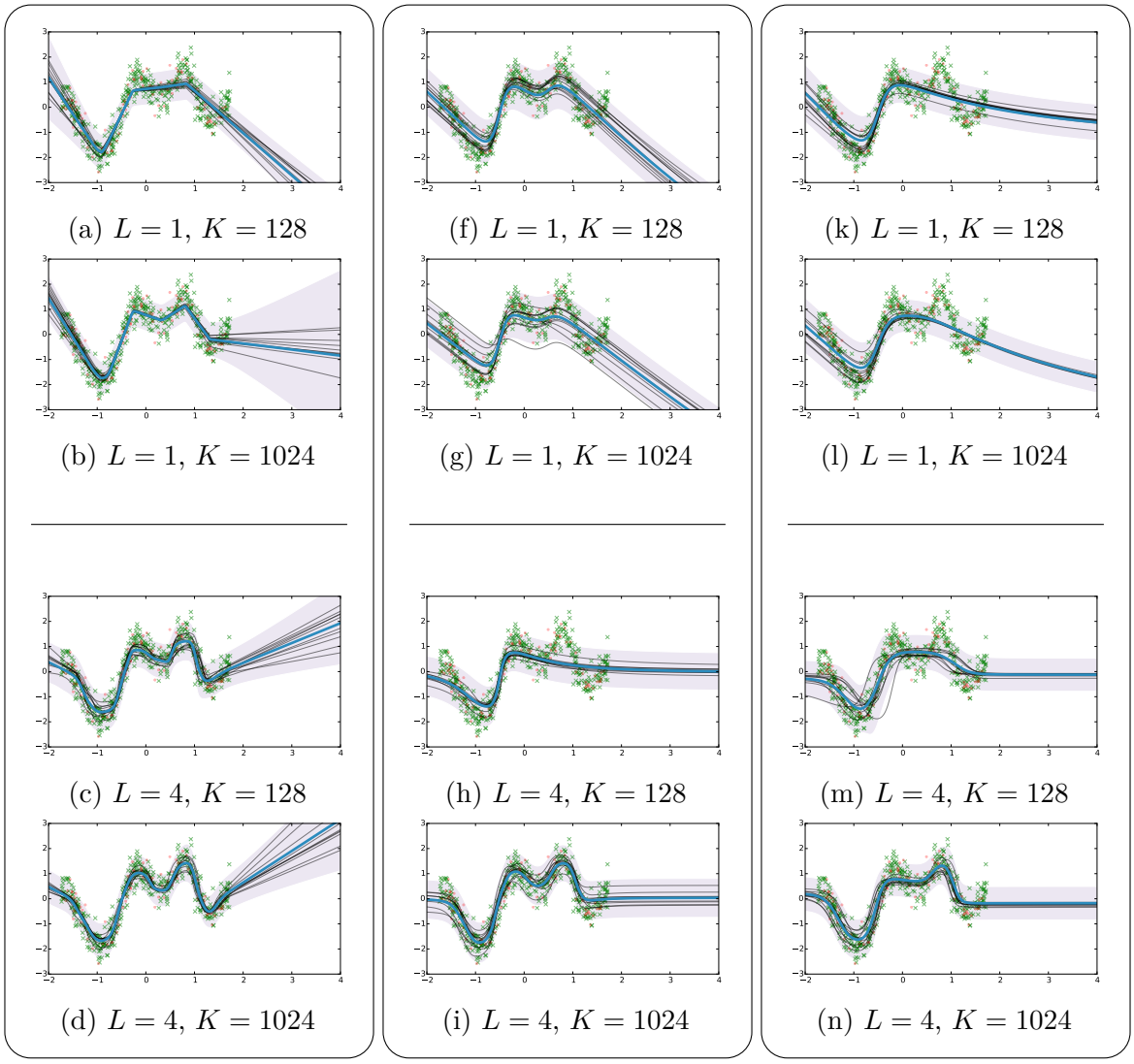
(d) $l = 0.1$, $K = 4096$ (k) $l = 0.1$, $K = 4096$ (r) $l = 0.1$, $K = 4096$

(e) $l = 1$, $K = 4096$ (l) $l = 1$, $K = 4096$ (s) $l = 1$, $K = 4096$

(f) $l = 5$, $K = 4096$ (m) $l = 5$, $K = 4096$ (t) $l = 5$, $K = 4096$

(g) ReLU (n) TanH (u) Sigmoid

Fig. B.1 Draws from Bayesian neural network prior with $L = 1$ hidden layers. Here l is the prior length-scale and K is the number of units.



(a) $L = 1$, $K = 128$ (f) $L = 1$, $K = 128$ (k) $L = 1$, $K = 128$

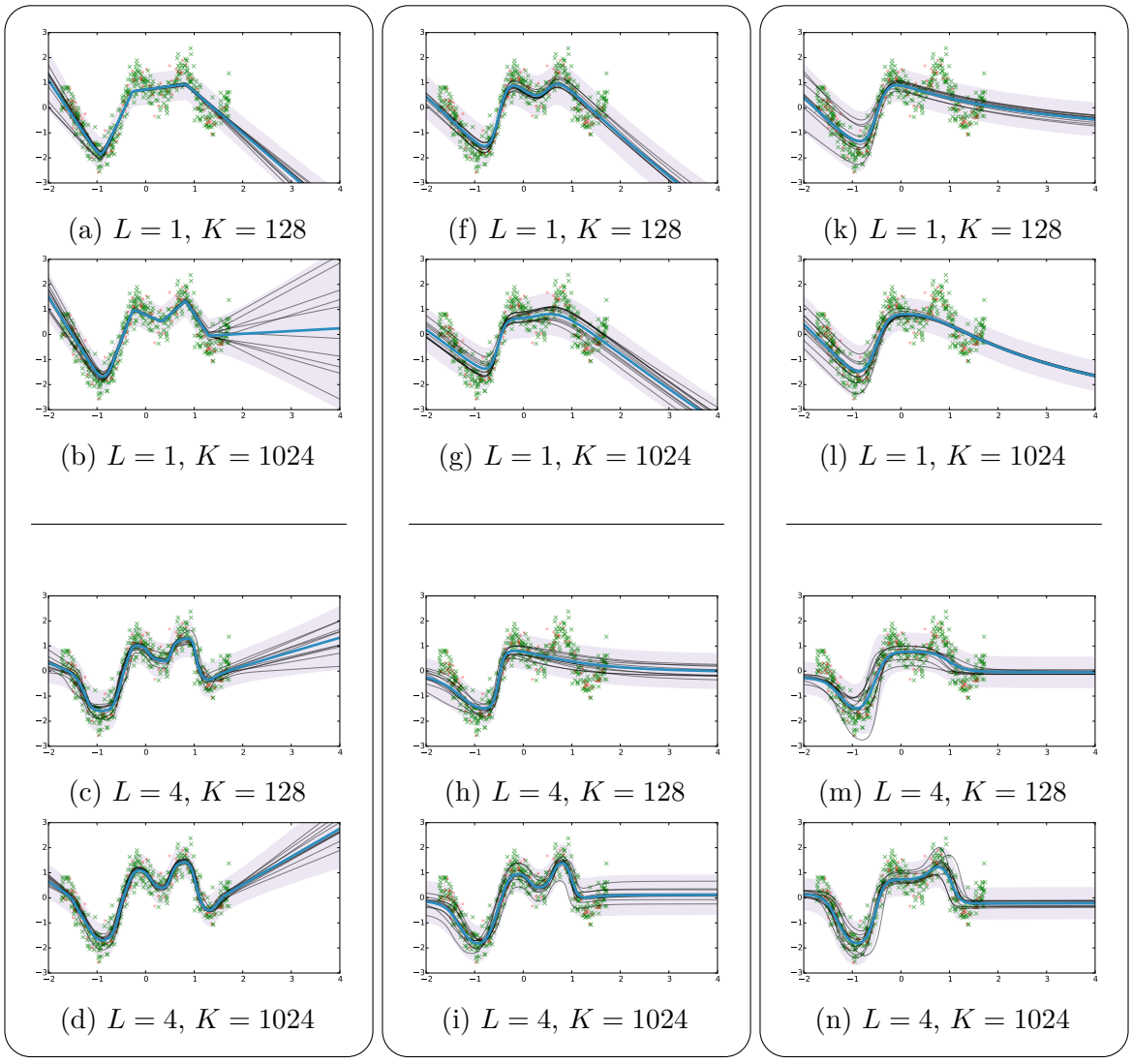
(b) $L = 1$, $K = 1024$ (g) $L = 1$, $K = 1024$ (l) $L = 1$, $K = 1024$

(c) $L = 4$, $K = 128$ (h) $L = 4$, $K = 128$ (m) $L = 4$, $K = 128$

(d) $L = 4$, $K = 1024$ (i) $L = 4$, $K = 1024$ (n) $L = 4$, $K = 1024$

(e) ReLU (j) TanH (o) Sigmoid

Fig. B.2 Draws from a Bayesian neural network posterior with **dropout approximating distribution**; shown are predictive mean (thick blue line), predictive uncertainty (shaded area, showing 2 standard deviations), and draws from the posterior (thin black lines). Scattered are training points. Best viewed on a computer screen. Here L is the number of network hidden layers and K is the number of units in each hidden layer.



(a) $L = 1$, $K = 128$ (f) $L = 1$, $K = 128$ (k) $L = 1$, $K = 128$

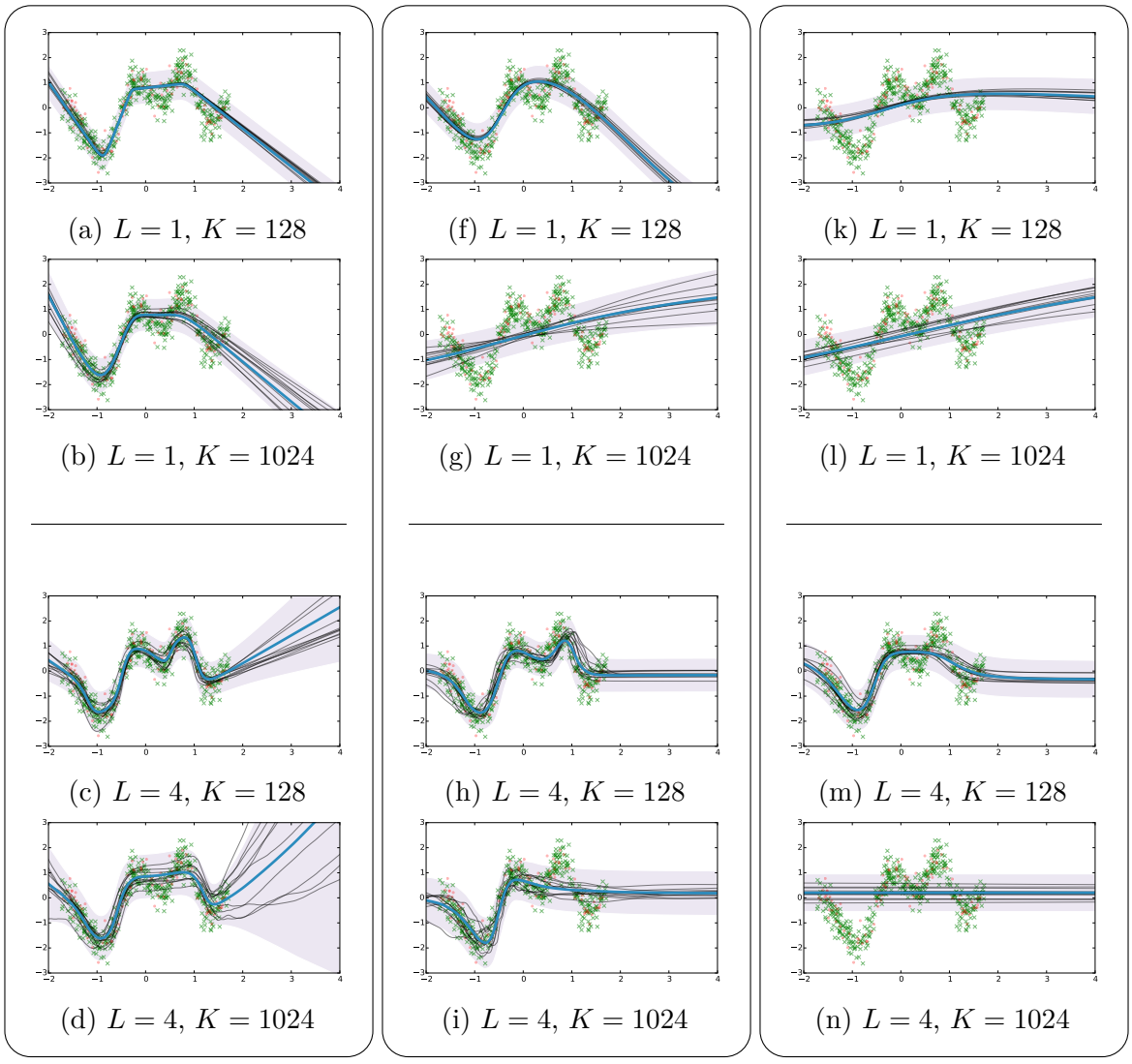
(b) $L = 1$, $K = 1024$ (g) $L = 1$, $K = 1024$ (l) $L = 1$, $K = 1024$

(c) $L = 4$, $K = 128$ (h) $L = 4$, $K = 128$ (m) $L = 4$, $K = 128$

(d) $L = 4$, $K = 1024$ (i) $L = 4$, $K = 1024$ (n) $L = 4$, $K = 1024$

(e) ReLU (j) TanH (o) Sigmoid

Fig. B.3 Draws from a Bayesian neural network posterior with **multiplicative Gaussian noise (MGN) approximating distribution**; shown are predictive mean (thick blue line), predictive uncertainty (shaded area, showing 2 standard deviations), and draws from the posterior (thin black lines). Scattered are training points. Best viewed on a computer screen. Here L is the number of network hidden layers and K is the number of units in each hidden layer.



(a) $L = 1$, $K = 128$ (f) $L = 1$, $K = 128$ (k) $L = 1$, $K = 128$

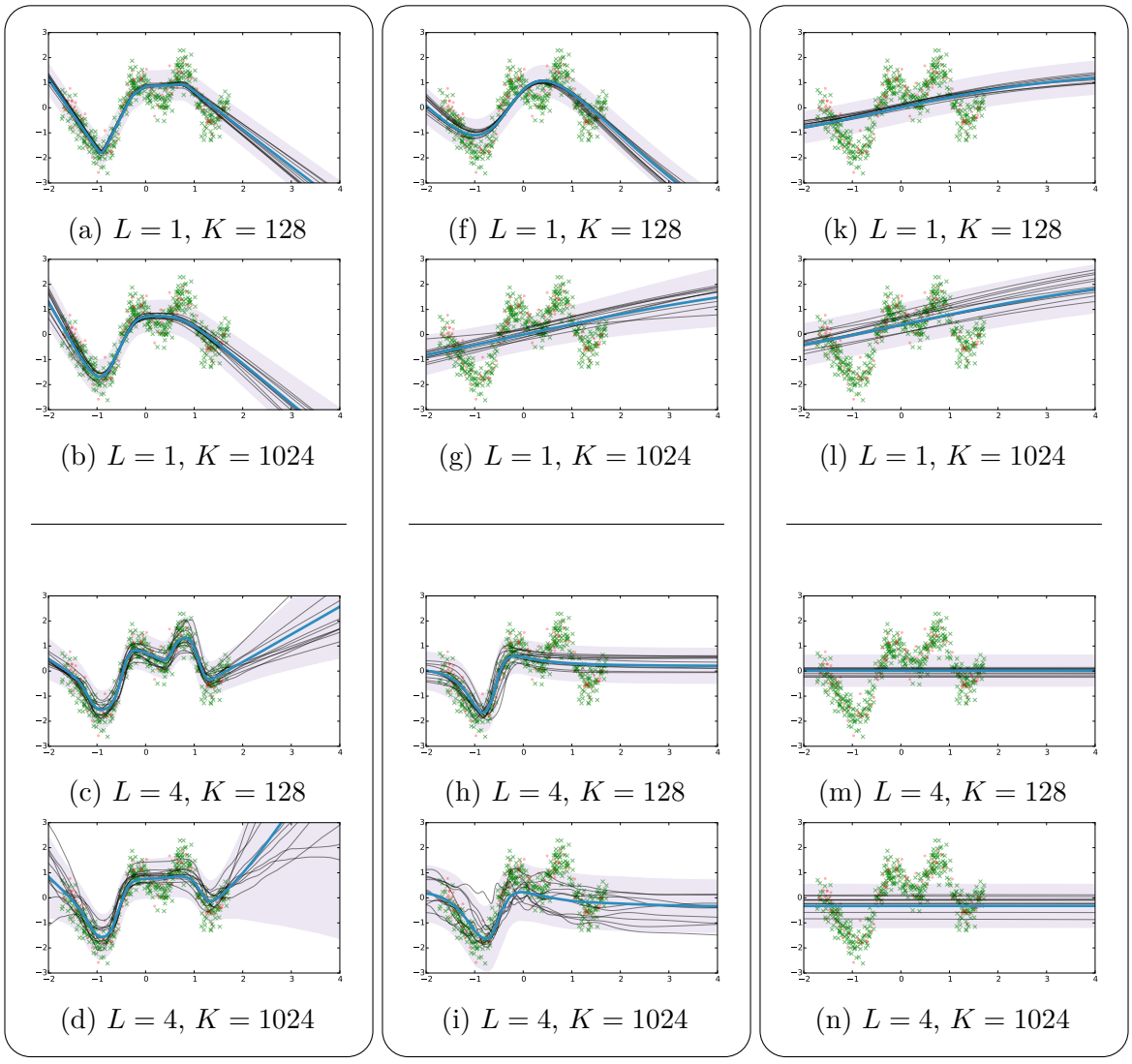
(b) $L = 1$, $K = 1024$ (g) $L = 1$, $K = 1024$ (l) $L = 1$, $K = 1024$

(c) $L = 4$, $K = 128$ (h) $L = 4$, $K = 128$ (m) $L = 4$, $K = 128$

(d) $L = 4$, $K = 1024$ (i) $L = 4$, $K = 1024$ (n) $L = 4$, $K = 1024$

(e) ReLU (j) TanH (o) Sigmoid

Fig. B.4 Draws from a Bayesian neural network posterior with a **factorised Gaussian approximating distribution**; shown are predictive mean (thick blue line), predictive uncertainty (shaded area, showing 2 standard deviations), and draws from the posterior (thin black lines). Scattered are training points. Best viewed on a computer screen. Here L is the number of network hidden layers and K is the number of units in each hidden layer.



(a) $L = 1$, $K = 128$ (f) $L = 1$, $K = 128$ (k) $L = 1$, $K = 128$

(b) $L = 1$, $K = 1024$ (g) $L = 1$, $K = 1024$ (l) $L = 1$, $K = 1024$

(c) $L = 4$, $K = 128$ (h) $L = 4$, $K = 128$ (m) $L = 4$, $K = 128$

(d) $L = 4$, $K = 1024$ (i) $L = 4$, $K = 1024$ (n) $L = 4$, $K = 1024$

(e) ReLU (j) TanH (o) Sigmoid

Fig. B.5 Draws from a Bayesian neural network posterior with a **mixture of Gaussians (MoG, row-wise) approximating distribution**; shown are predictive mean (thick blue line), predictive uncertainty (shaded area, showing 2 standard deviations), and draws from the posterior (thin black lines). Scattered in are training points. Best viewed on a computer screen. Here L is the number of network hidden layers and K is the number of units in each hidden layer.

Appendix C

Spike and slab prior KL

We can evaluate the KL divergence between the approximating distribution of section §6.6.5 and the spike and slab prior analytically:

$$\mathrm{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega})) = \sum_{ik} \mathrm{KL}(q(\mathbf{w}_{ik})||p(\mathbf{w}_{ik}))$$

with

$$\begin{aligned}
\mathrm{KL}(q(\mathbf{w}_{ik})||p(\mathbf{w}_{ik})) &= \int q(\mathbf{w}_{ik}) \log \frac{q(\mathbf{w}_{ik})}{p(\mathbf{w}_{ik})} \mathrm{d}\mathbf{w}_{ik} \\
&= \int q(\mathbf{w}_{ik}) \log \frac{\mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) p(\mathbf{w}_{ik})/Z_q}{p(\mathbf{w}_{ik})} \mathrm{d}\mathbf{w}_{ik} \\
&= \int q(\mathbf{w}_{ik}) \log \mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) \mathrm{d}\mathbf{w}_{ik} - \log Z_q \\
&= \int \left(\frac{\alpha}{\alpha+1} \delta_{\mathbf{0}} + \frac{1}{\alpha+1} \mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I \right) \right) \\
&\qquad \cdot \log \mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) \mathrm{d}\mathbf{w}_{ik} - \log Z_q \\
&= \frac{\alpha}{\alpha+1} \log \mathcal{N}(\mathbf{0}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) + \frac{1}{\alpha+1} \int \mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I \right) \\
&\qquad \cdot \log \mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) \mathrm{d}\mathbf{w}_{ik} - \log Z_q
\end{aligned}$$

Note that the KL is properly defined since for every measurable set $q(\cdot)$ has mass on, $p(\cdot)$ has mass on as well (including the singleton set $\{0\}$!).

We evaluate the last integral as follows:

$$\int \mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I \right) \log \mathcal{N}(\mathbf{w}_{ik}; \mathbf{m}_{ik}, \sigma_{ik}^2 I) \mathrm{d}\mathbf{w}_{ik}$$

$$
\begin{aligned}
&= \int \mathcal{N}\left(\mathbf{w}_{ik}; \frac{\mathbf{m}_{ik}}{1+l^2\sigma_{ik}^2}, \frac{\sigma_{ik}^2}{1+l^2\sigma_{ik}^2} I\right)\left(-\frac{1}{2\sigma_{ik}^2}(\mathbf{w}_{ik}^T\mathbf{w}_{ik} - 2\mathbf{w}_{ik}^T\mathbf{m}_{ik} + \mathbf{m}_{ik}^T\mathbf{m}_{ik})\right.\\
&\qquad \left. - \frac{K}{2}\log(2\pi\sigma_{ik}^2)\right)\mathrm{d}\mathbf{w}_{ik}\\
&= -\frac{l^4\sigma_{ik}^2}{(1+l^2\sigma_{ik}^2)^2}\frac{\mathbf{m}_{ik}^T\mathbf{m}_{ik}}{2} - \frac{K}{2(1+l^2\sigma_{ik}^2)} - \frac{K}{2}\log(2\pi\sigma_{ik}^2)
\end{aligned}
$$

leading to an analytical solution. Note that Z_q depends on the variational parameters, and thus was not omitted. This derivation results in an L_2 like regularisation, depending on the magnitude of $\mathbf{m}$, but through α as well.