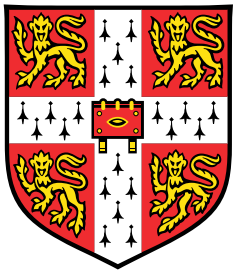
Probabilistic Model Compression



Kristopher Miltiadou

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

St. Edmund's College August 2021

I dedicate this thesis to my grandparents, whilst not all of you are with us today, I will never forget the opportunity that your hard work has brought me.

Declaration

I, Kristopher Miltiadou of St. Edmund's College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

All the software used in this thesis was written from scratch in Python using the PyTorch library.

This dissertation contains no more than 14,978 words including the appendices.

Kristopher Miltiadou
August 2021

Acknowledgements

First and foremost, I would like to acknowledge my supervisor, José Miguel Hernández-Lobato for nudging me in the right directions and answering my incoherent questions with great efficiency.

Second, I must acknowledge my co-supervisor Gergely Flamich. Words cannot describe how grateful I am for your council, advice and friendship throughout the thesis. You have been a star and I wish you the best in the future.

I want to thank Laurence for his friendship throughout the course. It was so much fun running, jumping, swimming and climbing around Cambridge this year.

My parents and sister, thank you for always supporting me and having immense patience driving up and down to meet me whenever I asked.

Finally, thank you to my girlfriend Maria. I loved exploring Cambridge with you, you've made this a year I will never forget. You were my rock this whole year when I had bad times, I can't thank you enough.

Abstract

With advances in technology, data is being transmitted at growing rates. The efficiency by which we handle this data is fundamental to continuing the productivity of the technology industry. Traditional compression methods rely upon eradicating redundancies in data and mapping frequent patterns to shorter messages. The key question of compression is how to find such patterns. For specific tasks, compression methods have been very successful (e.g. JPEG for images), but these methods are often hand-crafted and medium-specific.

This work extends a probabilistic compression method called Index Relative Entropy Coding (iREC), proposed by Flamich et al. [2021]. This method compresses the data by sending a random sample from a posterior distribution that models the data, using the prior distribution that is shared with the receiver. It can be applied to any data modelled with a generative distribution, so has wide applications unlike medium-specific methods. In this work, we conduct investigations and give new insights into the hyper-parameter settings of iREC.

Sometimes the posterior distributions modelling the data are intractable and approximations are necessary which introduce a performance gap. Previous work with iREC used variational inference to approximate the posterior with a factorised Gaussian distribution. We propose two new target distributions to approximate the posterior; a mixture of deltas and a mixture of Gaussians. These approximations are more expressive than a factorised Gaussian, and better suited to approximate complex posterior distributions with multi-modality and correlations. The posteriors of probabilistic models are often complex, so we consider the task of communicating multiple weights from the posterior of a regression model to perform inference on unseen data. We demonstrate that our proposed target distributions lead to better approximations of the posterior. The resulting models have better uncertainty calibration on the test data and smaller compression sizes when compressed with iREC. Lastly, we propose two new additions to the iREC method: (1) a method for training the posterior distribution of the data to yield a specific compression size and, (2) a method to update the prior distribution, based on the sequence of transmitted samples, that improves both the compression size and quality.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

To sustain the rapid advancements in technology, our methods for handling data efficiently must stand up against the growing rates of data creation and transmission. This is increasingly important within Machine Learning, due to the strong trend of increasing models sizes [Brown et al., 2020]. To enable widespread adoption of machine learning models on less powerful devices like smartphones, models need to be communicated efficiently and take on smaller memory footprints. Therefore, a memory efficient representation of the model is key for its practicality in everyday use.

Standard model compression methods typically rely on a 3 step process: weight pruning, quantization and coding [Han et al., 2015]. Pruning and quantization can effectively reduce the footprint of the network. By having fewer weights, and lower precision for each weight, the empirical distribution over the weights has smaller entropy. Shannon [1948] showed that this is directly linked to the optimal compression efficiency. These standard compression methods rely heavily on hyper-parameter tuning to yield good performance and are not trivial to apply to arbitrary architecture.

In this work, we examine and extend an alternative approach proposed by Flamich et al. [2021], called Index Relative Entropy Coding (iREC). With this method, instead of communicating a deterministic setting of weights, we model the weights in a probabilistic fashion and communicate a random sample of the weights. That is, after observing a data-set $\mathscr{D}$ we learn the posterior of the weights and send a sample $\mathbf{w} \sim p(\mathbf{w} \mid \mathscr{D})$. To do this, we draw samples from a shared prior distribution and transmit a sample that has high probability under the posterior. Since we can take any model and place a prior over its weights, this method is easily applied unlike the standard methods which require more engineering. The compression cost of this method is the relative entropy between the posterior and the prior. Hinton and

Van Camp [1993], in their 'Bits-back' argument, showed that this is the optimal compression size for communicating samples from a posterior when drawing samples from the prior. The previous work with iREC, from Flamich et al. [2021], used variational inference to learn the posterior distribution, we show that using exact samples from the true posterior improves performance and compression.

Furthermore, by using this compression method we can transmit multiple samples from the posterior distribution and perform inference. In situations where the wrong decision can be extremely costly (e.g. autonomous driving), having poor uncertainty calibration can be catastrophic. This necessitates the need for models which capture uncertainty to be stored and transmitted efficiently. Our method provides an easy way to compress Bayesian models. As such, we focus on the task of transmitting weights from a posterior distribution of a regression model. A key determinant of success is ensuring that the weight samples are representative of the full posterior distribution, rather than only faithful to local regions (e.g. specific modes) and so this is focused upon in this work.

1.1 Main Contributions

The key contributions of this work are:

- Extending the current repertoire of target distributions[1] that can be used with iREC. We derive two novel schemes: (1) a mixture of deltas where each mixture element is an exact sample from the posterior, and (2) a Gaussian kernel density estimate of the true posterior.
- Applying the iREC in the setting of probabilistic model compression. We discuss the caveats of this problem setting compared to previous applications since we consider the transmission of multiple samples from the posterior.
- A novel method for optimising the auxiliary variables' prior variances, and discovering scenarios in which the optimised variances are not suitable.
- A comprehensive dive into the limitations of iREC, and demonstration of how the performance is sensitive to the hyper-parameter settings.
- Proposing methods to improve iREC; including explicit compression sizes and dynamic proposals. Thorough experimentation is left to future work.

[1] The target distribution are typically approximations to the posterior distribution which is often intractable.

1.2 Thesis Outline

The thesis is organised as follows:

- Chapter 2 outlines the relevant source coding theory and describes how to encode integers efficiently - which is fundamental to our compression method. We link the Minimal Description Length Principle to Bayesian modelling and provide intuition for why using exact samples should improve the performance of our compressed models.
- Chapter 3 presents Index Relative Entropy Coding and our novel target distributions. We give a detailed description and examine the hyper-parameter choices. We also raise concerns over the proposed method, in particular, compression speed and mode bias.
- Chapter 4 explains the methodology for our experiments; the choice of data-sets, models and training procedures. We examine the results from each experiment, linking our findings to the investigations conducted in Chapter 3.
- Chapter 5 begins with a high-level overview of the current methods used for model compression. Subsequently, we draw inspiration from these to propose improvements to iREC.
- Chapter 6 concludes our work by discussing the significance and limitations of our method - specifically the next steps required to make it fully functional in a production setting.

Chapter 2

Background

2.1 Information Theory Foundations

This section lays out the information theory necessary to digest the thesis. First we explain two results: Shannon's Source Coding Theorem and the Kraft-McMillan Inequality. These results tell us what the optimal average codelength is, and how long the codelengths for each symbol should be to achieve this. Afterwards, we detail a practical method for encoding integers. The method involves the use of entropy coding methods with an appropriate Zeta distribution; an efficient encoding of integers is vital for our proposed model compression method.

2.1.1 Foundations of Source Coding

We define the information source as a random variable X governed by a generating distribution $P(X)$. Imagine a sequence of symbols (data) $S_1, ..., S_N$ that are the outcomes of the random variable X. To communicate this data, the sender must map the sequence of symbols to a sequence of binary digits (bits) such that the receiver can decode and recreate the original symbols. With compression, the aim is to perform this mapping by minimising the number of bits transmitted to the receiver. Naively, one might index each symbol and map the index to a unique binary sequence, the following example shows that this is sub-optimal:

Example:
Suppose an 8-sided die has probabilities for each number given by $\{\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16},\frac{1}{64},\frac{1}{64},\frac{1}{64},\frac{1}{64}\}$. How should you use binary digits to communicate the outcome of a throw of this die?
There are many ways to index the outcomes, for example $\mathscr{C}(1) = 000, \mathscr{C}(2) = 001, \mathscr{C}(3) = 010, \mathscr{C}(4) = 100, ...$ etc. Regardless of the dice throw, this method results in 3 bits need to communicate the outcome. Due to the non-uniformity in probabilities, we can do better:
Instead consider the following messages for each outcome: $1, 01, 001, 0001, 000011, 000010, 000001, 000000$. The average codelength is 2 bits compared to the 3 bits for the naive method - a 33% reduction in size. In fact, the codelength is equal to the entropy of the distribution of the die and we will see later this is optimal.

Intuitively, more frequent events will be transmitted more often, so should be given shorter codelengths. A keen-eyed reader might ask: why not encode the 2 most probable events with codes $0, 1$ then the 4 next most probable with codes $00, 01, 10, 11$ and so on? If we send multiple outcomes using one long string of binary digits this idea breaks down, since we cannot uniquely decode each outcome. A code is *uniquely decodable* if any sequence of codes can be decoded into one unique sequence of messages. When discussing optimal codelength, we do so in the setting of transmitting a (potentially) infinite stream of messages. Given this, Shannon [1948] proved tight bounds on how well we can do:

Theorem 1. *(Shannon's Source Coding Theorem as in MacKay [2003])*
N i.i.d. random variables each with entropy $H[X]$ can be compressed into more than $N \times H[X]$ bits with negligible risk of information loss, as $N \longrightarrow \infty$; conversely if they are compressed into fewer than $N \times H[X]$ bits it is virtually certain that information will be lost.

This theorem declares that, when transmitting an infinite stream of data, it is impossible to compress the data such that the codelength is less than the entropy of the source X without losing information. On the other hand, we can achieve a codelength arbitrarily close to the entropy with negligible information loss. Therefore, the optimal codelength we can achieve is given by the entropy and it is possible to get arbitrarily close to this. Methods achieving close to this bound are referred to as entropy coding methods - notable examples are Huffman Coding [Huffman, 1952] and Arithmetic coding [Witten et al., 1987].

Whilst this provides a target for the optimal codelength, how should we go about achieving it? The Kraft-McMillan Inequality states how long the codelength of each symbol needs to be depending on how often they occur. We present the theorem as stated in Cover [1999], but for the case where we encode our messages with binary digits.

Theorem 2. *For any set of symbols $S_1, ..., S_N$ that can be encoded with binary digits into a uniquely decodable code, the codelengths of each symbol, $l_1, ..., l_N$, must satisfy the inequality:*

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1. \tag{2.1}$$

Conversely, given a set of codeword lengths satisfying this inequality, it is possible to construct a uniquely decodable code.

This theorem tells us how to decide the codelengths for each symbol but we still have a wide array of possible codes to choose from. A convenient set of uniquely decodable codes are known as *prefix codes*. A symbol code is a prefix code when no codeword is a prefix of any other codeword [MacKay, 2003].

Example:
$\{00, 01, 10, 11\}$ is a prefix code as none of the codewords are prefixes for any other.
$\{0, 010\}$ is not a prefix code since 0 is the prefix for 010.

Kraft proved that if the inequality in Theorem 2 is satisfied, there exists a prefix code with the given lengths. This means that for any problem, there exists an optimal set of codewords that are also prefix codes. Huffman coding [Huffman, 1952] is an entropy coding method that is an optimal prefix code, it provides an algorithm for how to construct the code based on the probability distribution generating the data. For a detailed explanation see Cover [1999].

2.1.2 Encoding Integers with the Zeta Distribution

The main method we propose in this thesis involves the transmission of integers. In general, the distribution of these integers is unknown, so we must find a general distribution for integers that gives good coding efficiency, regardless of the true distribution of the integers. Such a distribution can be used with an entropy coding method like Huffman coding to send integers efficiently.

The Zeta distribution is a discrete probability distribution over integers $k \in \mathbb{N}$ with probability mass function given by

$$f_s(k) = k^{-s}/\zeta(s). \tag{2.2}$$

Here s is a parameter of the distribution and ζ is the Riemann-zeta function defined

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s} = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \cdots, \text{ if } \mathscr{R}e(s) > 1 \tag{2.3}$$

where $\mathscr{R}e(s)$ denotes the real-part of the complex number $s \in \mathbb{C}$. We require $\mathscr{R}e(s) > 1$ to ensure the summation in Equation (2.3) converges (for $s = 1$ we get the infamous divergent Harmonic Series). The Zeta distribution concerns $s \in (1, \infty)$, so the mass function decays as k grows and the rate of this decay is given by s. To see that Equation (2.2) defines a valid probability mass function note that

$$\sum_{k=1}^{\infty} k^{-s}/\zeta(s) = \frac{\sum_{k=1}^{\infty} k^{-s}}{\sum_{n=1}^{\infty} n^{-s}} = 1. \tag{2.4}$$

Furthermore, the decaying properties of the Zeta distribution mean we expect larger integers to occur less frequently. A plot of the distribution is given in Figure 2.1.

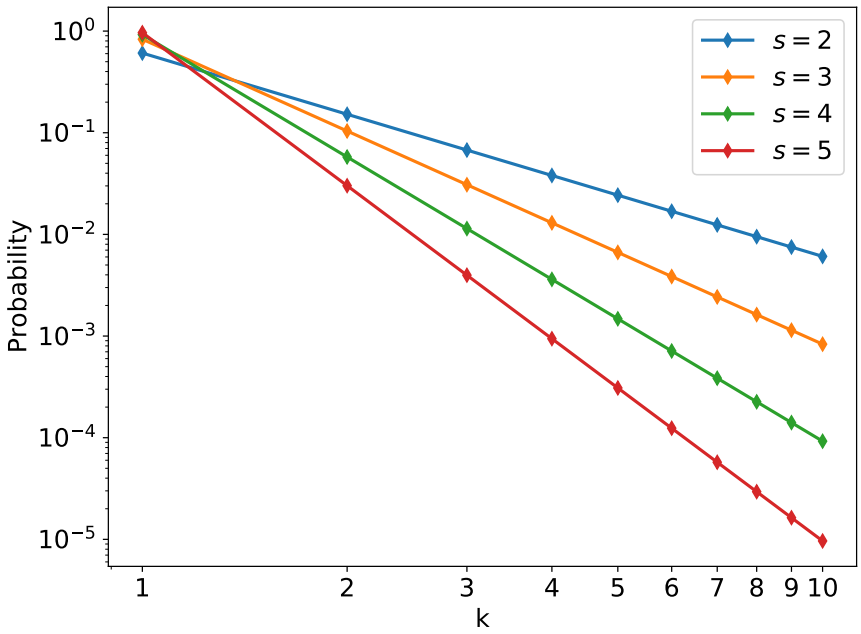


Fig. 2.1 Plot of the Zeta distribution for a variety of s values. Here we find that as s gets larger the decay in probability mass becomes more extreme and so larger integers are deemed less probable.

Suppose the sender wishes to transmit a potentially infinite stream of integers $N_1, N_2, ...$ from source Θ. We assume the only shared knowledge between the sender and receiver is $\mathbb{E}[\log\Theta]$ (this information can be transmitted along with the integers with minimal cost). In this case, we want to encode the integers with a method that gives us some assurances of the expected codelength. The following proposition shows that we can bound the entropy of the distribution of Θ:

Proposition 1. *(Taken from Appendix B Li and El Gamal [2018]) Let $\Theta = \{1,2,3,...\}$ be a random variable, then*

$$H(\Theta) \leq \mathbb{E}[\log\Theta] + \log(\mathbb{E}[\log\Theta] + 1) + 1. \tag{2.5}$$

Furthermore, the proof of the proposition gives a recipe for achieving an expected coding rate, which is given by the upper bound in Equation (2.5):

Proof. (Taken from Appendix B Li and El Gamal [2018]) Let $q(\theta) = c\theta^{-\lambda}$ where $\lambda = 1 + 1/\mathbb{E}[\log\Theta]$, and $c > 0$ such that $\sum_{\theta=1}^{\infty} q(\theta) = 1$. Note that

$$\sum_{\theta=1}^{\infty} \theta^{-\lambda} \leq 1 + \int_1^{\infty} \theta^{-\lambda} d\theta = 1 + \frac{1}{\lambda - 1}$$

and therefore

$$\begin{aligned}
H(\Theta) &\leq \sum_{\theta=1}^{\infty} p_\Theta(\theta) \log \frac{1}{q(\theta)} \\
&= \sum_{\theta=1}^{\infty} p_\Theta(\theta)(\lambda \log\theta - \log c) \\
&= \lambda\mathbb{E}[\log\Theta] + \log\left(\sum_{\theta=1}^{\infty} \theta^{-\lambda}\right) \\
&\leq \lambda\mathbb{E}[\log\Theta] + \log\left(1 + \frac{1}{\lambda - 1}\right) \\
&= \mathbb{E}[\log\Theta] + \log(\mathbb{E}[\log\Theta] + 1) + 1.
\end{aligned}$$

□

The distribution in the proof used to encode Θ, $q(\theta)$, is precisely the Zeta distribution with parameter $s = 1 + \frac{1}{\mathbb{E}[\log\Theta]}$. Therefore, using the correct Zeta distribution with an

entropy coding method, like Huffman coding, yields a prefix code over the integers that is asymptotically optimal[1].

2.1.3 Minimum Description Length Principle

Occam's Razor is a familiar heuristic that tells us, out of all possible models, we should prefer the model that provides the simplest explanation of the data. The MDL principle [Barron et al., 1998] formalises this and states that the best model $\mathbf{H}_i$ in some set of models $\mathscr{H}$ is the one that minimises the quantity

$$\mathscr{L}(\mathbf{H}) + \mathscr{L}(\mathscr{D} \mid \mathbf{H}), \tag{2.6}$$

where $\mathscr{L}(\mathbf{H})$ is the length in bits required to describe $\mathbf{H}$ and $\mathscr{L}(\mathscr{D} \mid \mathbf{H})$ is the length in bits of the description of the data when encoded using the model $\mathbf{H}$. The first term can be thought of as a model complexity term, with the second a data-fit term. Later, we will see this is related to Bayesian modelling.

2.2 Probabilistic Modelling

In this section, we introduce Variational Inference, Markov chain Monte Carlo, and link the MDL Principle to Bayesian modelling. Later, we demonstrate how to trade-off model fit and compressibility by using a β-objective function that directly relates to the MDL principle.

2.2.1 Modelling Uncertainty

Given a data-set $\mathscr{D} = \{X, Y\}$ composed of inputs $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ and targets $Y = \{\mathbf{y}_1, ..., \mathbf{y}_n\}$, we want to train a model $\mathscr{M}$ with weights $\mathbf{w}$ to correctly predict the target at each input. Rather than learning a specific deterministic value for the weights, we place a prior $p(\mathbf{w})$ over the weights to factor in our initial beliefs on their values. After observing the data $\mathscr{D}$, we update our belief in the weights using Bayes' rule:

$$p(\mathbf{w} \mid \mathscr{D}) = \frac{p(\mathscr{D} \mid \mathbf{w}) p(w)}{p(\mathscr{D})}. \tag{2.7}$$

[1] This means that the ratio of the expected codelength and the optimal codelength is bounded by a constant and converges to 1 as the entropy tends to infinity (see Elias [1975] for more details)

This distribution describes the probability of each weight setting having generated the dataset $\mathscr{D}$, given how likely we thought their values were initially. Given this, we can make inferences on unseen data $X^\star, Y^\star$ by marginalising over all possible weight values:

$$p(Y^\star \mid X^\star, \mathscr{D}) = \int p(Y^\star \mid X^\star, \mathbf{w}) p(\mathbf{w} \mid \mathscr{D}) \mathrm{d}\mathbf{w}. \tag{2.8}$$

This gives a distribution over the potential targets, meaning our uncertainty in the model parameters is represented in uncertainty in predictions.

For many models, performing the marginalisation in Equation (2.8) is intractable, so other methods are needed to help make these inferences. We can write Equation (2.8) as the following expectation:

$$p(Y^\star \mid X^\star, \mathscr{D}) = \mathbb{E}_{p(\mathbf{w}\mid\mathscr{D})}\left[p(Y^\star \mid X^\star, \mathbf{w})\right]. \tag{2.9}$$

Therefore, we can use a Monte Carlo estimate

$$p\left(Y^\star \mid X^\star, \mathscr{D}\right) \approx \frac{1}{K} \sum_{k=1}^{K} p\left(Y^\star \mid X^\star, \mathbf{w}_k\right), \quad \text{where } \mathbf{w}_k \sim p(\mathbf{w} \mid \mathscr{D}). \tag{2.10}$$

With enough samples, this approximation converges to the true predictive posterior. The problem is we often cannot trivially sample from $p(\mathbf{w} \mid \mathscr{D})$. We consider two different methods for dealing with this: (1) Variational Inference (VI) which approximates the true posterior, and (2) Markov chain Monte Carlo which aims to sample from the true posterior.

2.2.2 Variational Inference

Variational Inference (VI) [Hinton and Van Camp, 1993] approximates the posterior distribution of the weights of a model $p(\mathbf{w} \mid \mathscr{D})$ using a distribution $q_\phi(\mathbf{w})$ parametrised by ϕ. The Kullback–Leibler divergence (KL) (also referred to as the relative entropy) between the two distributions, $\mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w} \mid \mathscr{D})\right]$, measures the similarity between q_ϕ and p. Thus, given some parametrised distribution $q_\phi(\mathbf{w})$, we wish to find the parameters ϕ minimising the quantity

$$\mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w} \mid \mathscr{D})\right] = \int q_\phi(\mathbf{w}) \log \frac{q_\phi(\mathbf{w})}{p(\mathbf{w} \mid \mathscr{D})} \, \mathrm{d}\mathbf{w} \tag{2.11}$$

$$= \log p(\mathscr{D}) + \int q_\phi(\mathbf{w}) \log \frac{q_\phi(\mathbf{w})}{p(\mathbf{w})} \, \mathrm{d}\mathbf{w} - \int q_\phi(\mathbf{w}) \log p(\mathbf{w} \mid \mathscr{D}) \, \mathrm{d}\mathbf{w}. \tag{2.12}$$

The first term in Equation (2.12) is the model evidence which is often intractable. It is straightforward to show that the KL is non-negative using Jensen's inequality[2], so we can find a lower bound to the model evidence, named the Evidence Lower Bound (ELBO),

$$\log p(\mathscr{D}) \geq \int q_\phi(\mathbf{w}) \log p(\mathbf{w} \mid \mathscr{D}) \, \mathrm{d}\mathbf{w} - \int q_\phi(\mathbf{w}) \log \frac{q_\phi(\mathbf{w})}{p(\mathbf{w})} \, \mathrm{d}\mathbf{w} \tag{2.13}$$

$$= \mathbb{E}_{q_\phi(\mathbf{w})} \left[\log p(\mathscr{D} \mid \mathbf{w}) \right] - \mathrm{KL} \left[q_\phi(\mathbf{w}) \| p(\mathbf{w}) \right] = \mathrm{ELBO}. \tag{2.14}$$

By maximising the ELBO, we minimise the KL between $p(\mathbf{w} \mid \mathscr{D})$ and $q_\phi(\mathbf{w})$ given in Equation (2.11). The first term in the ELBO is the log-likelihood of the data conditioned on the model parameters, which measures how well the model explains the data, and the second term is the KL between the variational posterior and the prior, which can be interpreted as the complexity in the model.

Regardless of how we select q_ϕ, models like neural networks cause Equation (2.14) to be intractable, and Monte Carlo estimates coupled with gradient-based optimisation are used to optimise ϕ. We can draw weight samples $\mathbf{w}^{(i)} \sim q_\phi(\mathbf{w})$ and approximate the ELBO as

$$\mathrm{ELBO} \approx \sum_{i=1}^{I} \log p(\mathscr{D} \mid \mathbf{w}^{(i)}) - \log q_\phi(\mathbf{w}^{(i)}) + \log p(\mathbf{w}^{(i)}). \tag{2.15}$$

To differentiate this quantity, we need to propagate gradients through the random sample $\mathbf{w}^{(i)} \sim q_\phi(\mathbf{w})$. The *reparameterisation trick* introduced by Kingma and Welling [2013], allows for gradients to be easily computed with respect to parameters ϕ, by reparameterising the random variable $\mathbf{w}$ as the output of a differentiable transformation of some auxiliary noise,

$$\mathbf{w} = g_\phi(\varepsilon) \quad \text{with} \quad \varepsilon \sim p(\varepsilon). \tag{2.16}$$

For example, a diagonal Gaussian with parameters $\phi = \{\mu, \sigma\}$ can be sampled as follows:

$$\mathbf{w}^{(i)} = \mu + \sigma \odot \varepsilon \tag{2.17}$$

where $\varepsilon \sim \mathscr{N}(\mathbf{0}, \mathbb{I})$ and $\odot$ is the element-wise product.

For Bayesian Neural Networks (BNNs), the typical training method is *Bayes-By-Backprop* [Blundell et al., 2015], which formulates the ELBO for use with minibaches of the data.

[2] Jensen's inequality states that $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$ if f is a convex function.

Later, Kingma et al. [2015] discovered that sampling the activations rather than the weights results in a lower variance gradient estimator for the ELBO.

Linking VI to MDL Principle:
Hinton and Van Camp [1993] with their 'Bits-back Argument' established a link between VI and the MDL principle. One can view the negative ELBO as the description length:

$$\text{NEGATIVE ELBO} = \underbrace{-\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathscr{D} \mid \mathbf{w})]}_{\mathscr{L}(\mathscr{D}|\mathbf{H})} + \underbrace{\text{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right]}_{\mathscr{L}(\mathbf{H})}. \tag{2.18}$$

The ELBO relates to the objective used in Bayesian model comparison, so we can make an equivalence between being Bayesian about our model selection and choosing models that will compress well. Further, Equation (2.18) shows the optimal compression size of the model $\mathscr{L}(\mathbf{H})$, is given by $\text{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right]$. Since this fact was derived by Hinton and Van Camp [1993] in their Bits-back Argument, methods that (asymptotically) achieve this compression size are refereed to as 'Bits-back' efficient. This however is a misnomer, in many cases the methods do not actually utilise or follow the bits-back argument; they simply approach the postulated coding efficiency. True Bits-back methods require the transmission of negative information which is not convenient for algorithmic implementation.

We have shown how to train probabilistic models with VI and linked the ELBO to the MDL principle. Whilst these methods are common in the Machine Learning literature, the quality of the results depend heavily on the variational approximation. Getting closer to the true posterior should yield models with larger ELBOs, improving the data-fit and compressibility.

2.2.3 Markov Chain Monte Carlo

Instead of a variational approximation, we can target the exact posterior by constructing a Markov chain with $p(\mathbf{w} \mid \mathscr{D})$ as its stationary distribution. To construct these Markov chains we consider the class of sampling methods called Markov chain Monte Carlo (MCMC). A Markov chain will yield samples from $p(\mathbf{w} \mid \mathscr{D})$, if it is ergodic and leaves the target distribution invariant. At a high level, ergodicity means that, independent to our starting point, our chain eventually converges to the target, and invariance means once we reach the target, further transitions in our Markov chain don't lead to it escaping, i.e. our Markov

chain's stationary distribution is the target distribution. If our chain satisfies these conditions, we can use the accepted samples to evaluate the estimate in Equation (2.10) - with enough samples it should converge to the true predictive.

Typical machine learning applications involve large models evaluated on large data-sets of high dimensions. Consequently, many of the customary MCMC methods cannot be practically applied to current deep learning architectures since they can take long to converge and scale badly with the size of the data-set.

Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) draws inspiration from physical systems to explore complex high-dimensional distributions efficiently. In this section, we assume the target distribution is the posterior over the weights of a model, $p(\mathbf{w} \mid \mathscr{D})$.

Hamiltonian Monte Carlo (HMC) uses the Hamiltonian dynamics of a fictitious physical system, in which a ball moves around a friction-less surface, to sample from $p(\mathbf{w} \mid \mathscr{D})$. If we imagine this surface to live in $\mathbb{R}^3$ with coordinates $\{x, y, z\}$, then the **position** of the ball is given by the x, y co-ordinates, and its **height** is given by z. The weights $\mathbf{w}$ are defined to be the position of the ball, and as we move around the weight space, the ball will move around the surface. Assume the ball has mass M and momentum $\mathbf{r}$, then its kinetic energy is given by $K(\mathbf{r}) = \frac{1}{2M}\mathbf{r}^T\mathbf{r}$. The potential energy depends on the ball's height h which relates to its position on the surface, $\mathbf{w}$. Therefore it is given by $U(\mathbf{w}) \propto h$. The core idea in HMC is that we can write the posterior on the weights as

$$p(\mathbf{w} \mid \mathscr{D}) \propto p(\mathbf{w}, \mathscr{D}) \tag{2.19}$$

$$= \frac{1}{Z_p} \exp(-U(\mathbf{w})). \tag{2.20}$$

So the potential energy of the ball can be expressed as

$$U(\mathbf{w}) = -\log p(\mathscr{D} \mid \mathbf{w}) - \log p(\mathbf{w}). \tag{2.21}$$

By inversely relating the height of the ball to the posterior probability of its position (weights) $p(\mathbf{w} \mid \mathscr{D})$, we can make an equivalence between the motion of the ball and exploring the posterior distribution of the weights. Furthermore, we can consider the momentum of the ball (with mass 1) given by the rate of change of the ball's position,

$$\mathbf{r} = \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\tau}. \tag{2.22}$$

The acceleration of the ball is the rate of change of momentum. This is given by the force exerted on the ball, which is the negative gradient of the potential energy,

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\tau} = -\frac{\mathrm{d}U(\mathbf{w})}{\mathrm{d}\mathbf{w}}. \tag{2.23}$$

We can define the joint distribution $p(\mathbf{w},\mathbf{r})$ in terms of the Hamiltonian, $H(\mathbf{w},\mathbf{r})$, which is the total energy of the system

$$H(\mathbf{w},\mathbf{r}) = K(\mathbf{r}) + U(\mathbf{w}) \tag{2.24}$$

$$\propto -\log p(\mathbf{w},\mathbf{r}). \tag{2.25}$$

With these equations, we can simulate the ball rolling around the physical system by considering its Hamiltonian dynamics,

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\tau} = \frac{\partial H}{\partial \mathbf{r}} \tag{2.26}$$

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\tau} = -\frac{\partial H}{\partial \mathbf{w}}. \tag{2.27}$$

By the conservation of energy, H stays constant throughout time, and it can be shown using *Liouville's Theorem*[3] that Hamiltonian systems preserve volume in phase space. These properties imply that Hamiltonian dynamics will leave the distribution $p(\mathbf{w},\mathbf{r})$ invariant. Since the value of the Hamiltonian H remains constant, the Markov chain will not be ergodic, it will stick to the same probability contours that correspond to a single value of the total energy in the system. Therefore, to sample ergodically we need to change the value of the Hamiltonian. This can be done by making movements in the phase space whilst ensuring we leave $p(\mathbf{w},\mathbf{r})$ invariant. Gibbs sampling the momentum term $\mathbf{r}$ accomplishes this and is trivial to do,§ since the conditional distribution $p(\mathbf{r} \mid \mathbf{w})$ is a Gaussian due to the independence of $\mathbf{r}$ and $\mathbf{w}$. So we can sample from $p(\mathbf{w},\mathbf{r})$, discarding the sampled momentum $\mathbf{r}$, to obtain a sample from $p(\mathbf{w} \mid \mathscr{D})$.

In continuous time this method gives us a way to make large movements in the sample space without needing to reject any samples. To practically apply this method however, we need to discretize the time domain - the typical method being *leapfrog* discretization. These discretizations introduce residual errors, and a *Metropolis* step is used to account for this.

[3] For an explanation of the theorem see Bishop [2006].

2.2.4 Trading Model Fit and Compressibility

Referring to the link between the ELBO and the MDL principle, we see that by weighting the KL divergence term, we can impose a trade-off between the model's data fit and its complexity - directly relating to its compressibility. To do this we adopt the β-ELBO loss akin to the β-VAE proposed by Higgins et al. [2017]. This is given explicitly as

$$\beta\text{-ELBO} = \mathbb{E}_{q_\phi(\mathbf{w})}\left[\log p(\mathscr{D} \mid \mathbf{w})\right] - \beta \mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right]. \tag{2.28}$$

It can be convenient to instead train with an alternative objective, whereby we raise the likelihood of the data given the model by the exponent $1/\beta$

$$\mathbb{E}_{q_\phi(\mathbf{w})}\left[\log p(\mathscr{D} \mid \mathbf{w})^{1/\beta}\right] - \mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right]. \tag{2.29}$$

This expression has the same functional derivative as the standard β-ELBO; (proof in Appendix A). With these two training objectives, we can implicitly control the compressibility of our variational models by placing more or less emphasis on the model complexity.

When using MCMC to perform inference, we would like to impose an analogous trade-off. Fortunately, we can augment the potential energy term in Equation (2.21) used by HMC in a similar fashion to the alternative training loss in Equation (2.29). We set the potential energy

$$\tilde{U}(\mathbf{w}) = -\log p(\mathscr{D} \mid \mathbf{w})^{1/\beta} - \log p(\mathbf{w}). \tag{2.30}$$

If the likelihood is Gaussian, there is a more practical way to write the potential energy that is less obtrusive for established HMC implementations. Suppose we have a data-set $\mathscr{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$ and model $\mathscr{M}$, whose output at point $\mathbf{x}$ using weights $\mathbf{w}$ is given by $\mathscr{F}_\mathbf{w}(\mathbf{x})$. Then for a known noise term σ, we define the Gaussian likelihood as

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathscr{N}(y \mid \mathscr{F}_\mathbf{w}(\mathbf{x}), \sigma^2). \tag{2.31}$$

We can write the log-likelihood term raised to $1/\beta$ as

$$\begin{aligned}
\log p(y \mid \mathbf{w})^{1/\beta} &= \frac{1}{\beta} \log p(y \mid \mathbf{w}) \\
&= \frac{1}{\beta} \log \mathscr{N}(y_n \mid \mathscr{F}_\mathbf{w}(\mathbf{x}), \sigma^2) \\
&\propto -\frac{1}{2\beta\sigma^2} \left(\mathscr{F}_\mathbf{w}(x) - y\right)^2 \\
&\propto \log \mathscr{N}(y; \mathscr{F}_\mathbf{w}(x), \beta\sigma^2).
\end{aligned} \tag{2.32}$$

Since the final quantity is proportional to the log-likelihood raised to $1/\beta$, their derivatives will coincide. Therefore, to make the trade-off between data-fit and compressibility, we can implement HMC with the potential energy function

$$\hat{U}(\mathbf{w}) = -\log \hat{p}(\mathscr{D} \mid \mathbf{w}) - \log p(\mathbf{w}) \tag{2.33}$$

where $\hat{p}(y \mid \mathbf{x}, \mathbf{w}) = \mathscr{N}(y; \mathscr{F}_{\mathbf{w}}(x), \beta\sigma^2)$.

In conclusion, we have shown how to use both VI and HMC to perform inference over probabilistic models. Specifically, by using alternative β training objectives, we can implicitly control their compression size.

Chapter 3

Relative Entropy Coding

In this section, we focus on the Index Relative Entropy Coding (iREC) compression method from Flamich et al. [2021]. We propose new target distributions for iREC that utilise exact samples from the true posteriors. Our proposed target distributions are much more flexible than the factorised Gaussian distribution that is currently used. This allows them to better approximate complex posterior distributions which should result in improved compresssion efficiency. After, we analyse the iREC hyper-parameters by conducting small demonstrative tests and highlight some limitations of the method. All the algorithms presented in this chapter were either taken or adapted from Flamich et al. [2021].

3.1 Introduction

Relative Entropy Coding (REC), introduced by Flamich et al. [2021], is a class of compression algorithms that communicate a random sample from a target distribution q using a proposal distribution p. Explicitly, the sender wishes to send a sample $\mathbf{z}$ from a target distribution denoted $q(\mathbf{z})$ (unknown to the receiver), using a shared coding distribution $p(\mathbf{z})$ and a shared source of randomness (e.g. a random number generator). Algorithms that output a random variable $\mathbf{z} \sim q(\mathbf{z})$ with communication cost close to the relative entropy between q and p, $\mathrm{KL}[q(\mathbf{z})||p(\mathbf{z})]$, are termed REC algorithms.

Previous work with REC algorithms focused on sending single samples from the latent space of a VAE to compress images. In this thesis, we focus on sending sets of compressed samples $\mathbf{w}_1, ..., \mathbf{w}_N$ from a trained Bayesian model $\mathscr{M}$ and using these to make predictions. To fully capture the uncertainty modelled by $\mathscr{M}$, our compressed samples must be representative of the posterior distribution over the weights. The caveats of sending multiple samples rather

than a single sample are discussed later in this chapter, and it is useful for the reader to keep this difference in mind. Furthermore, since these compression schemes work for any target and coding distribution, we choose to denote our sample with $\mathbf{z}$.

The question beckons: why is such a class of algorithms useful? Suppose we want to send a sample from a prior over the weights of a model, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \mathbb{I})$. With the standard entropy coding methods described in Section 2.1.1, we know the expected codelength is lower bounded by the Shannon entropy. Since $\mathbf{w}$ is a Gaussian, its Shannon entropy[1] is infinite and so the expected codelength would be infinite. Instead, if we used a REC algorithm with $q(\mathbf{w}) = p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \mathbb{I})$ then to send a sample of the weights, we simply need to communicate to the receiver that they have to draw a sample from p, this incurs an $\mathcal{O}(1)$ cost. In both cases, the receiver needs to know $p(\mathbf{w})$, but using a REC algorithm makes compressing a sample from $q(\mathbf{w})$ much cheaper. This shows that encoding a random sample from the target distribution can be far cheaper than communicating a specific sample with entropy coding and motivates the need for REC algorithms. This benefit is even stronger in our case of sending multiple weight samples $\mathbf{w}_1, \mathbf{w}_2, \ldots$ to perform inference. Since we are using the compressed samples to make a Monte Carlo estimate of the predictive distribution, refer back to Equation (2.10), it is unnecessary to send specific samples. Rather, we simply want the set of compressed samples to be representative of the posterior distribution over the weights, ensuring sensible uncertainty calibration in our predictions. This further motivates the use of REC algorithms for compressing probabilistic models.

3.2 Index Coding

Index coding (iREC) [Flamich et al., 2021] is a REC algorithm that leverages pseudo-random number generators to provide the shared source of randomness between the sender and receiver. The sender and receiver agree upon a prior (coding) distribution $p(\mathbf{z})$ and random number generator to draw equivalent samples from $p(\mathbf{z})$. To communicate $\mathbf{z} \sim q(\mathbf{z} \mid \mathbf{x})$ the sender draws a sequence of samples from the coding (prior) distribution $\mathbf{z}_1, \mathbf{z}_2, \ldots \sim p(\mathbf{z})$ and chooses a sample $\mathbf{z}_i$ to communicate to the receiver. Since the sender and receiver share the pseudo-random number generator, the sender simply has to transmit the index i and the receiver draws i samples $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_i \sim p(\mathbf{z})$ to decode $\mathbf{z}_i$.

[1] Note that this is different to its differential entropy.

3.2.1 Choosing the sample

The iREC method uses an importance sampling procedure proposed by Havasi et al. [2018] to draw samples from $q(\mathbf{z})$. The procedure is as follows: using the shared pseudo-number generator draw $M = \lceil \exp(\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]) \rceil$ samples from $p(\mathbf{z})$. Create a categorical distribution $\tilde{Q}$ over these samples by setting the probability mass proportional to the importance weights $\tilde{Q}(\mathbf{z}_i) \propto \frac{q(\mathbf{z}_i)}{p(\mathbf{z}_i)}$, and draw a sample from $\tilde{Q}$. This sample corresponds to a specific $\mathbf{z}_i$ and the sender can easily transmit its index i to the receiver (Algorithm 1). The receiver then simply draws the i^{th} sample $\mathbf{z}_i$ from $p(\mathbf{z}_i)$ (Algorithm 2). Since we can encode any number $0 \leq i < M$ with $\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]$ nats, this method satisfies the coding efficiency to be classed a REC algorithm.

Algorithm 1: iREC Encoder. Here we use $\overset{R}{\sim}$ to denote samples generated using the pseudo random number with shared seed R.

Data: $p(\mathbf{z}), q(\mathbf{z})$
Result: $\mathbf{z}_i, i$
$M \leftarrow \lceil \exp(\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]) \rceil$;
$\mathbf{z}_1, \ldots, \mathbf{z}_M \overset{R}{\sim} p(\mathbf{z})$;
for $m = 1 : M$ **do**
  $w_m \leftarrow \frac{q(\mathbf{z}_m)}{p(\mathbf{z}_m)}$
end
for $m = 1 : M$ **do**
  $\tilde{Q}(\mathbf{z}_m) \leftarrow \frac{w_m}{\sum_{k=1}^{K} w_k}$
end
$\mathbf{z}_i \sim Q(\mathbf{z})$;
return $\mathbf{z}_i, i$;

Algorithm 2: iREC Decoder. The receiver simply draws samples until the i-th index.

Data: $p(\mathbf{z}), i$
Result: $\mathbf{z}_i$
$\mathbf{z}_1, \ldots, \mathbf{z}_i \overset{R}{\sim} p(\mathbf{z})$;
return $\mathbf{z}_i$

3.2.2 Why should this work?

If we draw infinitely many samples from $p(\mathbf{z})$, under reasonable conditions[2], the categorical distribution $\tilde{Q}(\mathbf{z})$ should converge to $q(\mathbf{z})$. To achieve our desired codelength we only draw $M = \lceil \exp(\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]) \rceil$ samples. Therefore the categorical distribution $\tilde{Q}$ will be biased. The amount of bias determines the efficacy of the method - if $\tilde{Q}$ is too biased then our samples won't be representative of $q(\mathbf{z})$ and the method breaks down. Theorem 1.2 from Chatterjee and Diaconis [2017], specifically the special case defined Havasi et al. [2018], determines the extent of the bias:

Theorem 3. *(Taken from Havasi et al. [2018])*
Let p, q, be distributions over $\mathbf{z}$*. Let* $t \geq 0$ *and* $\tilde{Q}$ *be a discrete distribution constructed by drawing* $M = \lceil \exp(\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]) \rceil$ *samples* $\{\mathbf{z}_i\}_{m=1}^{M}$ *from p and defining* $\tilde{Q}(\mathbf{z}_i) := \frac{q(\mathbf{z}_i)}{p(\mathbf{z}_i)} / \sum_{m=1}^{M} \frac{q(\mathbf{z}_m)}{p(\mathbf{z}_m)}$*. Furthermore, let* $f(\mathbf{z})$ *be a measurable function and* $\|f\|_q = \sqrt{\mathbb{E}_q[f^2]}$ *be its 2-norm under q. Then it holds that*

$$\mathbb{P}\left(\left|\mathbb{E}_{\tilde{Q}}[f] - \mathbb{E}_q[f]\right| \geq \frac{2\|f\|_q \rho}{1-\rho}\right) \leq 2\rho \tag{3.1}$$

where

$$\rho = \left(e^{-t/4} + 2\sqrt{\mathbb{P}\left(\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right) > \mathrm{KL}\left[q(\mathbf{z})\|p(\mathbf{z})\right] + \frac{t}{2}\right)}\right)^{1/2}. \tag{3.2}$$

The quantity $\left|\mathbb{E}_{\tilde{Q}}[f] - \mathbb{E}_{q_\phi}[f]\right|$ in Equation (3.1) is the bias of estimating the expectation of function f with proxy distribution $\tilde{Q}$. Therefore, this theorem tells us that, with high probability, the bias from using our method is bounded since $e^{-t/4}$ decays with t and $\log \frac{q(\mathbf{z})}{p(\mathbf{z})}$ is concentrated around its expectation $\mathbb{E}_q[\log \frac{q(\mathbf{z})}{p(\mathbf{z})}] = \mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]$. In practice we choose to draw $\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})](1+\varepsilon)$ samples where $\varepsilon \geq 0$. In this case, ρ grows with the quantity

$$\mathbb{P}\left(\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right) > \mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})](1+\varepsilon) + \frac{t}{2}\right), \tag{3.3}$$

so drawing more samples makes the bound tighter[3] - reducing the bias.

[2] By this we mean that both q and p are continuous, and their importance weights are infinite for all values of $\mathbf{z}$.

[3] Drawing more samples reduces the probability in Equation (3.3) making ρ and therefore the bias smaller.

3.2.3 Greedy Selection

Another way of dealing with this bias is to greedily pick the sample $\mathbf{z}_i$ with the largest importance sampling weight, rather that creating and sampling from $\tilde{Q}$ (this method is detailed in Algorithm 3). This invokes a bias-variance trade-off, where the selected samples are biased to the modes of the target distribution - reducing their variance as a result. It is important to distinguish between the two biases: (1) the bias described in Theorem 3 and (2) the mode bias from greedily selecting the samples. To measure the first bias, note that we can write the relative entropy as the expectation

$$\mathrm{KL}\left[q(\mathbf{z})\|p(\mathbf{q})\right] = \mathbb{E}_q\left[\log\frac{q(\mathbf{z})}{p(\mathbf{z})}\right]. \tag{3.4}$$

Then we can approximate this quantity using samples compressed with our iREC scheme and analyse the bias-variance of the estimates. In Figure 3.1, we compress samples from a 1D Gaussian by sampling from a standard Gaussian using the importance sample method and greedy selection. Here, the Greedy selection method decreases the bias for the estimate of Equation (3.4) but also has smaller variance - even smaller than when using true samples. Whilst the Greedy selection method biases our samples to the mode, there are far fewer samples that would be highly unlikely under the true distribution q (see the samples of the left tail). Overall, as the difference between q and p grows, using the Greedy selection method is worth this trade-off, and so we chose to use this method in our investigations.

Algorithm 3: iREC Encoder w/ Greedy Selection.

Data: $p(\mathbf{z}), q(\mathbf{z})$
Result: $\mathbf{z}_i, i$
$M \leftarrow \lceil \exp\left(\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]\right) \rceil$;
$\mathbf{z}_1, ..., \mathbf{z}_M \overset{R}{\sim} p(\mathbf{z})$;
for $m = 1 : M$ **do**
  $w_m \leftarrow \frac{q(\mathbf{z}_m)}{p(\mathbf{z}_m)}$
end
$\mathbf{z}_i \leftarrow \arg\max_m w_m$;
return $\mathbf{z}_i, i$

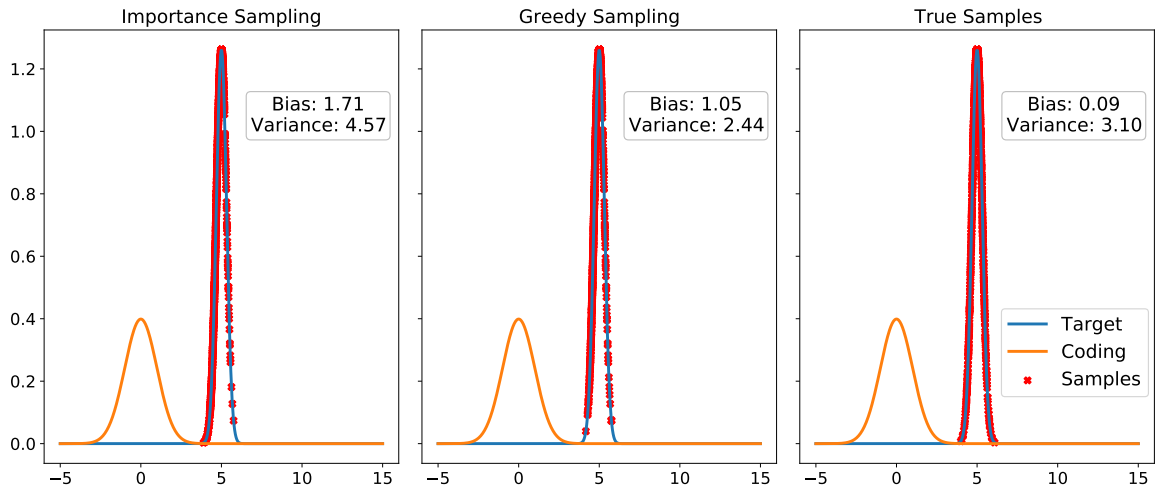


Fig. 3.1 Example comparing Importance Sampling to Greedy Selection. Here we can see that Greedy Selection improves the heuristic measure for bias, but in doing so reduces the sample variance.

3.3 Auxiliary Variable Method

For most problems, sampling $M = \lceil \exp(\mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]) \rceil$ random variables is infeasible since M grows exponentially with the relative entropy. To deal with this Flamich et al. [2021] propose breaking up the sample $\mathbf{z}$ into auxiliary variables, $\mathbf{a}_1, ..., \mathbf{a}_K$, such that $\mathbf{z} = f(\mathbf{a}_{1:K})$. Since the joint distribution of the auxiliary variables factorises, $q(\mathbf{a}_{1:k}) = \prod_{i=1}^{k} q(\mathbf{a}_i \mid \mathbf{a}_{1:i-1})$, each auxiliary variable $\mathbf{a}_k \sim q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$ can be sequentially transmitted using the iREC method with shared coding distribution $p(\mathbf{a}_k)$. This method incurs a communication cost of $\mathrm{KL}[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})\|p(\mathbf{a}_k)]$ nats to transmit the auxiliary variables $\mathbf{a}_k$. Now the sender and receiver must agree to K shared sources of randomness and prior distributions $p(\mathbf{a}_k)$, and a function f such that $\mathbf{z} = f(\mathbf{a}_{1:K})$. The sender transmits an index for each auxiliary variable and the receiver decodes each auxiliary variable to compute the desired sample $\mathbf{z} = f(\mathbf{a}_{1:K})$. So long as[4]

$$\sum_{k=1}^{K} \mathrm{KL}\left[q\left(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}\right) \| p\left(\mathbf{a}_k\right)\right] = \mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})], \tag{3.5}$$

the expected codelength of using the auxiliary variable method will coincide with that of transmitting a random sample $\mathbf{z} \sim q(\mathbf{z})$ with the standard iREC method. Flamich et al. [2021] prove that the auxiliary target distributions which satisfy Equation (3.5) must be of the

[4]Here we set $\mathrm{KL}\left[q\left(x \mid \mathbf{y}\right) \| p(x)\right] = \mathbb{E}_{x,y\sim q(x,y)}\left[\log \frac{q(x|y)}{p(x)}\right]$ as in Cover [1999].

form

$$q(\mathbf{a}_{1:k}) := \int p(\mathbf{a}_{1:k} \mid \mathbf{z})\, q(\mathbf{z}) \mathrm{d}\mathbf{z}, \quad \text{for } k \in \{1 \dots K\} \tag{3.6}$$

where

$$p(\mathbf{a}_{1:k} \mid \mathbf{z}) = \frac{p(\mathbf{z} \mid \mathbf{a}_{1:k})\, p(\mathbf{a}_{1:k})}{p(\mathbf{z})}. \tag{3.7}$$

This gives us a recipe for 'building a sample' of $\mathbf{z} \sim q(\mathbf{z})$ from the auxiliary variables $\mathbf{a}_{1:K}$.

Encoding the first auxiliary variable is simple: first compute $q(\mathbf{a}_1)$ and then perform the iREC method using shared coding distribution $p(\mathbf{a}_1)$. Encoding the subsequent auxiliary variables is more involved since there are dependencies due to the constraint that $\mathbf{z} = f(\mathbf{a}_{1:k})$. Therefore we need to find the conditional distribution for each auxiliary variable, $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$, and use these as the target distributions. Each auxiliary variable target distribution should adjust based on the previously sampled trajectory of auxiliary variables $\mathbf{a}_{1:k-1}$. The adjustment needs to ensure that final trajectory can reach a sample $\mathbf{z} \sim q(\mathbf{z})$. We can recursively find the necessary conditional distributions as follows[5]:

$$\begin{aligned} q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) &= \int q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z})\, q(\mathbf{z} \mid \mathbf{a}_{1:k-1})\, \mathrm{d}\mathbf{z} \\ &= \int p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z})\, q(\mathbf{z} \mid \mathbf{a}_{1:k-1})\, \mathrm{d}\mathbf{z} \\ q(\mathbf{z} \mid \mathbf{a}_{1:k}) &= \frac{q(\mathbf{z}, \mathbf{a}_k \mid \mathbf{a}_{1:k-1})}{q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})} = \frac{p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z})\, q(\mathbf{z} \mid \mathbf{a}_{1:k-1})}{q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})}. \end{aligned} \tag{3.8}$$

With these equations we can implement iREC for each auxiliary variable and the decoder can recreate a random sample $\mathbf{z} \sim q(\mathbf{z})$ by decoding each $\mathbf{a}_k$ and computing $\mathbf{z} = f(\mathbf{a}_{1:K})$. Like previously, there is a bias from our importance sampling scheme. Now the bias occurs for each auxiliary variable $\mathbf{a}_k$, thus accumulates across the trajectory. Consequently, we choose again to greedily select the trial sample with the largest log importance weight, that is, $\mathbf{a}_k = \arg\max_{\mathbf{a}_k^{(i)}} \frac{q(\mathbf{a}_k^{(i)} \mid \mathbf{a}_{1:k-1}^{(i)})}{p(\mathbf{a}_k^{(i)})}$ (this method is detailed in Algorithm 4).

3.4 Target Distributions

This thesis concerns sending samples[6] from a posterior distribution $p(\mathbf{z} \mid \mathbf{x})$. Previous work with iREC used a factorised Gaussian approximation as the target. In this thesis, we propose

[5] The second equality for $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$ is implied by Equation (3.6).

[6] Specifically we focus on sending weights of a model, but since these compression methods are not medium specific we choose to present them in general terms.

Algorithm 4: iREC Encoder w/ Greedy Selection using auxiliary variables. The iREC$(\cdot,\cdot)$ function performs the greedy selection described in Algorithm 3.

Data: $q(\mathbf{z}), p(\mathbf{a}_{1:K})$
Result: $S = (i_1, ..., i_K)$
$S \leftarrow \{()\}$ $q_0(\mathbf{z}) \leftarrow q(\mathbf{z})$; **for** $k = 1 : K$ **do**
  $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \leftarrow \int p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z}) q_{k-1}(\mathbf{z})\, \mathrm{d}\mathbf{z}$;
  $\mathbf{a}_k, i_k \leftarrow$ iREC$(q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}), p(\mathbf{a}_k))$;
  S.append(i_k)
end
return S

two new target distributions that use exact samples from the true posterior: a mixture of deltas and a mixture of Gaussians. These distributions can capture more complex structures like multimodality and correlation. Below we define the equations necessary to implement iREC with the difference schemes, with full derivations for the novel schemes given in Appendix A.2.

3.4.1 Factored Gaussian Scheme

Like we discussed in Section 2.2, often the posterior over the weights is intractable and so we approximate it with Variational Inference (VI). Commonly in VI, $p(\mathbf{z} \mid \mathbf{x})$ is approximated using a factorised Gaussian $q(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \nu, \rho)$ with prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma^2\mathbf{I})$.

To compress samples $\mathbf{z} \sim q(\mathbf{z})$ using the auxiliary variable iREC scheme, we can choose auxiliary variables such that $\mathbf{z} = \sum_{k=1}^{K} \mathbf{a}_k$ with $\mathbf{a}_k \sim \mathcal{N}(\mathbf{a}_k \mid \mathbf{0}, \sigma_k^2\mathbf{I})$ where $\sigma = \sum_{k=1}^{K} \sigma_k^2$. Due to the stability of Gaussians under addition, we already know that the auxiliary variable targets will be Gaussians, and so the factorised Gaussian makes for a convenient target distribution.

Let $s_k = \sum_{i=k+1}^{K} \sigma_k^2$ and $\mathbf{b}_k = \sum_{i=1}^{k} \mathbf{a}_i$, since $\mathbf{z} = \sum_{k=1}^{K} \mathbf{a}_k$ it follows from the formulae for products of Gaussians (see Appendix A.2 in Rasmussen [2003]) that

$$p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z}) = \mathcal{N}\left(\mathbf{a}_k \,\middle|\, (\mathbf{z} - \mathbf{b}_{k-1}) \frac{\sigma_k^2}{s_{k-1}^2}, \frac{s_k^2 \sigma_k^2}{s_{k-1}^2}\mathbf{I}\right). \tag{3.9}$$

Assume that $q(\mathbf{z} \mid \mathbf{a}_{1:k-1}) = \mathcal{N}(\mathbf{z} \mid \nu_{k-1}, \rho_{k-1})$, since Equation (3.8) concerns products and integrals of Gaussians, it follows that

$$q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) = \mathcal{N}\left(\mathbf{a}_k \,\middle|\, (\nu_{k-1} - \mathbf{b}_{k-1}) \frac{\sigma_k^2}{s_{k-1}^2}, \frac{s_k^2 \sigma_k^2}{s_{k-1}^2}\mathbf{I} + \frac{\sigma_k^4}{s_{k-1}^4}\rho_{k-1}^2\right) \tag{3.10}$$

and

$$q(\mathbf{z} \mid \mathbf{a}_{1:k}) = \mathcal{N}\left(\mathbf{z} \mid \mathbf{c}^{(k)}, \mathbf{C}^{(k)}\right), \tag{3.11}$$

where

$$\mathbf{C}^{(k)} = \left(\frac{\sigma_k^2}{s_{k-1}^2 s_k^2}\mathbf{I} + \rho_{k-1}^{-1}\right), \text{ and } \mathbf{c}^{(k)} = \mathbf{C}^{(k)}\left(\frac{\sigma_k^2}{s_{k-1}^2 s_k^2}\left(\mathbf{b}_{k-1} + s_k^2 \mathbf{a}_k\right)\right). \tag{3.12}$$

The Gaussian in Equation (3.11) adjusts the distribution over $\mathbf{z}$ to take into account the current trajectory $\mathbf{a}_{1:k-1}$. The mean of the Gaussian in Equation (3.10) directs the sampling scheme to choose the $\mathbf{a}_k$ that best makes up the remaining distance the trajectory needs to travel to reach $q(\mathbf{z} \mid \mathbf{a}_{1:k-1})$, $\nu_{k-1} - \mathbf{b}_{k-1}$, weighted according to how much variance $\mathbf{a}_k$ contributes, σ_k^2 / s_{k-1}^2. The variance of the target is additive, taking into account the variance of $\mathbf{z}$ and the variance of $\mathbf{a}_k$ relative to the other auxiliary variables. We refer to using iREC with a factorised Gaussian target as the `FG-Scheme`.

Since the factorised Gaussian approximation minimises $\mathrm{KL}[q(\mathbf{z}) \| p(\mathbf{z} \mid \mathbf{x})]$ through the ELBO, it is mode-seeking [Bishop, 2006]. This means it is unable to capture multimodality which can occur with complex posteriors of BNNs. In addition, by assuming the factorisation, the approximation cannot model correlations. We will see later in the results that these issues cause poor uncertainty calibration in a model's predictions.

3.4.2 Mixture of Deltas

Instead of using a variational approximation, we propose sampling directly from the true posterior $p(\mathbf{z} \mid \mathbf{x})$ and forming a mixture of deltas. This approximation is much more flexible, and with enough samples directly converges to the true posterior.

Formally, we draw samples[7] from $z_1, ..., z_D \sim p(\mathbf{z} \mid \mathbf{x})$ and form the mixture of deltas

$$q(z) = \frac{1}{D}\sum_{i=1}^{D} \delta_{z_i}(z). \tag{3.13}$$

[7] If we cannot sample from $p(\mathbf{z} \mid \mathbf{x})$, we can use HMC as described in Section 2.2.

The recursive relations corresponding to Equation (3.8) are given by

$$q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) = \sum_{i=1}^{D} \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)}{\sum_{j=1}^{D} p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_j)} p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z}_i) \tag{3.14}$$

$$q(\mathbf{z} \mid \mathbf{a}_{1:k-1}) = \sum_{i=1}^{D} \delta_{\mathbf{z}_i}(\mathbf{z}) \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)}{\sum_{j=1}^{D} p(\mathbf{a}_{1:k-1} \mid z_j)}. \tag{3.15}$$

Set $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma^2\mathbf{I})$ and assume that $\mathbf{z} = \sum_{k=1}^{K} \mathbf{a}_k$ where $\mathbf{a}_k \sim \mathcal{N}(\mathbf{a}_k \mid \mathbf{0}, \sigma_k^2\mathbf{I})$. In this case, the auxiliary variables take on different targets than the distribution for $\mathbf{z}$ which is a mixture of deltas. Equation (3.9) implies that $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$ is a mixture of Gaussians. These update equations mean that the auxiliary variables target a mixture of Gaussians where each mixing weight $p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)$ determines how probable sample $\mathbf{z}_i$ is under the current trajectory $\mathbf{a}_{1:k-1}$. Further, each component is a Gaussian which directs the auxiliary variables toward each sample $\mathbf{z}_i$, again weighted by how much variance the auxiliary variable $\mathbf{a}_k$ contributes to the trajectory $\mathbf{a}_{1:K}$. Note, the formulae above are undefined for the final auxiliary variable $\mathbf{a}_K$, later in Section 3.4.4 we discuss how to deal with this.

This approximation is able to capture multimodality by having clusters of delta masses at each mode. Furthermore, it can model correlations as there is no restriction on where the delta masses can lie, they are simply drawn from $p(\mathbf{z} \mid \mathbf{x})$. Altogether, this target should be a much more faithful approximation of $p(\mathbf{z} \mid \mathbf{x})$. We refer to using iREC with a mixture of deltas target as the `MOD-Scheme`.

3.4.3 Mixture of Gaussians

For complex posterior distributions, a mixture of deltas may not be a sufficient approximation. As such, we propose using a kernel density estimate. Specifically, we form a mixture of Gaussians with equal mixing weights and isotropic noise denoted σ^2_{KDE},

$$q(\mathbf{z}) = \frac{1}{D} \sum_{d=1}^{D} \mathcal{N}(\mathbf{z} \mid \mathbf{z}_d, \sigma^2_{\text{KDE}}\mathbf{I}). \tag{3.16}$$

The component means $\mathbf{z}_d$ are exact samples from the true posterior[8]. It can be shown that that

$$q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) = \sum_{d=1}^{D} \frac{\lambda_d^{-1(k-1)}}{\sum_{j=1}^{D} \lambda_j^{-1(k-1)}} \mathcal{N}\left(\mathbf{a}_k \;\middle|\; \frac{\sigma_k^2}{s_{k-1}^2}(\mathbf{c}_d^{(k-1)} - \mathbf{b}_{k-1}), \frac{s_k^2 \sigma_k^2}{s_{k-1}^2}\mathbf{I} + \frac{\sigma_k^4}{s_{k-1}^4}\mathbf{C}_d^{(k-1)}\right), \tag{3.17}$$

$$q(\mathbf{z} \mid \mathbf{a}_{1:k}) = \sum_{d=1}^{D} \frac{\lambda_d^{-1(k)}}{\sum_{j=1}^{D} \lambda_j^{-1(k)}} \mathcal{N}(\mathbf{z} \mid \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}), \tag{3.18}$$

where

$$\lambda_d^{-1(k)} = \mathcal{N}(\mathbf{b}_k; \mathbf{z}_d, (s_k^2 + \sigma_{\text{KDE}}^2)\mathbf{I}) \tag{3.19}$$

$$\mathbf{C}_d^{(k)} = \left(\frac{1}{s_k^2} + \frac{1}{\sigma_{\text{KDE}}^2}\right)^{-1} \mathbf{I} \tag{3.20}$$

$$\mathbf{c}_d^{(k)} = \mathbf{C}_d^{(k)} \left(\frac{\mathbf{b}_k}{s_k^2} + \frac{\mathbf{z}_d}{\sigma_{\text{KDE}}^2}\right). \tag{3.21}$$

Looking closely, these equations are a combination of those derived for the `FG-Scheme` and the `MOD-Scheme`. Each auxiliary variable targets a mixture of Gaussians, where the mixing weights relate to how likely the trajectory is to reach the specific sample $\mathbf{z}_d$. Unlike the `MOD-Scheme`, where the only update to the distribution of $\mathbf{z}$ was to alter the mixing weights, in this case, both the mixing weights and the Gaussian components themselves update upon observing the trajectory $\mathbf{a}_{1:k-1}$. So rather than simply targeting each $\mathbf{z}_d$, they target a mixture with component means $\mathbf{c}_d^{(k)}$ which are updated depending on $\mathbf{a}_{1:k-1}$. This demonstrates that the mixture of Gaussians is a more flexible target than the mixture of deltas, but also has many more parameters to maintain. We refer to using iREC with a Gaussian kernal density estimate as the `KDE-Scheme`.

Note: consider drawing only one sample $\mathbf{z}_d$ equal to the mode of $p(\mathbf{z} \mid \mathbf{x})$. In this case, the equations for the `KDE-Scheme` collapse down to those for the `FG-Scheme`, whereby the only difference is our Gaussian is more constrained in that it is isotropic rather simply factorised. This is a good sanity check to ensure that the update equations are sensible.

[8]These are the same samples we would be using to form the mixture of deltas for the `MOD-Scheme`.

3.4.4 Choosing the Final Auxiliary

For the final auxiliary variable, the update equations given above may be undefined. Therefore, for all schemes we choose to select the final sample by completing the sum $\mathbf{z} = \sum_{k=1}^{K-1} \mathbf{a}_k + \mathbf{a}_K$ and selecting the $\mathbf{a}_K$ that maximises the importance weights $p(\mathbf{z} \mid \mathbf{x})/p(\mathbf{z}) = p(\mathbf{x} \mid x)$. Since we can always compute the likelihood this method is simple to implement even in cases where the posterior is intractable. However, doing this increases bias to the modes. Alternatively, we could simply not send the final auxiliary variable, which you can interpret as adding small noise onto $\mathbf{z}$, but this would cause issues when we have few auxiliary variables as the noise would be large relative to the variance of $\mathbf{z}$.

3.5 Visualising the Trajectories

To see the differences using iREC with a factorised target and a more flexible target, example trajectories are plotted for both the `FG-Scheme` and the `MOD-Scheme` in Figure 3.2. Here, $p(\mathbf{a}_k) = \mathcal{N}(\mathbf{a}_k \mid \mathbf{0}, \frac{1}{K}\mathbf{I})$ where K is the total number of auxiliary variables. Later we discuss how the sender should choose the auxiliary variable variances. The `FG-Scheme` targets a factorised Gaussian approximation that minimises $\mathrm{KL}\left[q(\mathbf{z}) \| p(\mathbf{z} \mid \mathbf{x})\right]$. Due to correlations in the true posterior, the approximation is a small target contained within the posterior. This has 2 effects: (1) the smaller target means the relative entropy between the $q(\mathbf{z})$ and $p(\mathbf{z})$ is larger, so we require more auxiliary variables - increasing the codelength to compress a sample. And (2) the scheme is unable to target the correlated part of the distribution which is easier to reach with samples from p. The trajectories in this plot have trouble reaching higher density regions of the target, but by using the parameter ε we could draw more samples to improve the quality of the compressed sample. However, this introduces an overhead in the codelength. Altogether, the `MOD-Scheme` requires fewer auxiliary variables which corresponds to a more efficient encoding, but also reaches the true target more effectively. Whilst this is only one run (using the same random seed) it is illustrative of the main differences between the schemes, and demonstrates some benefits of using exact samples from the posterior.

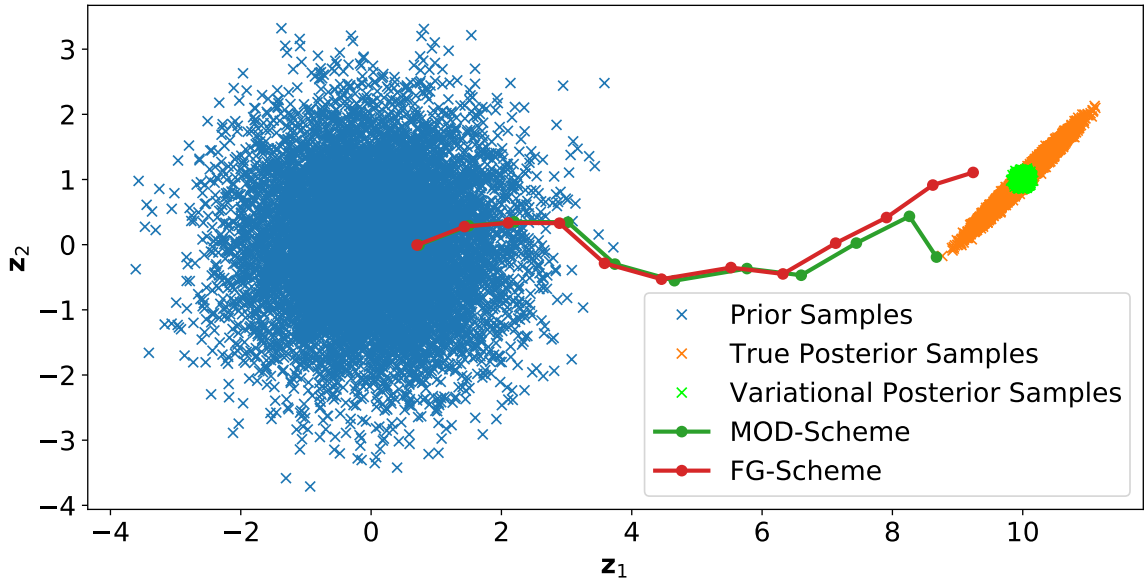


Fig. 3.2 Example comparing the `FG-Scheme` and `MOD-Scheme`. Since the factorised Gaussian variational approximation cannot capture correlations, the `FG-Scheme` fails to reach the target distribution and in turn the trajectory lands in a place with very low density. In contrast the `MOD-Scheme` has a trajectory that takes it close to the tip of the correlated edge and so the final compressed sample is more representative of the true target.

3.6 Practical Considerations

Each of the 3 proposed schemes has its own advantages and disadvantages. From a computational standpoint, the methods using exact samples (`MOG-Scheme` and `KDE-Scheme`) may require HMC to get samples from $p(\mathbf{z} \mid \mathbf{x})$ which can be expensive compared to simply training a factorised Gaussian with VI using the ELBO. Furthermore, they involve storing many more objects in the computer's memory: for each empirical sample, $\mathbf{z}_d$, we must store the Gaussian distribution it induces into the mixture density of $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$. In contrast, just one Gaussian needs to be maintained for the `FG-Scheme`. Whilst not a key aspect of this thesis, the computational cost is significant enough to deserve consideration, and could result in only the `FG-Scheme` being feasible for extremely high dimensional problems. We compare the runtimes of the `FG-Scheme` and `MOD-Scheme` when compressing samples. To ensure that the test is fair, we consider the targets $q_d(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \not{4}_d, \mathbf{I}_d)$ and priors $p_d(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}_d, \mathbb{I}_d)$ for dimension $d \in \{1,5,10,25,50,75,100\}$, this means both schemes target the same distribution. The relative entropies for each dimension are given by $\mathrm{KL}[q_d(\mathbf{z}) \| p_d(\mathbf{z})] = d/2$, so this test shows how both methods scale with the compression size. Finally, for the `MOD-Scheme`, the receiver must determine how many samples from

the true target they must use - each additional sample increases the memory usage and contributes to the computational complexity but gives a more refined approximation of the target. Therefore we compare how using $1, 10, 100$ and 1000 empirical samples to form our mixture of deltas affects the runtime. Figure 3.3 shows with 1 exact sample the `MOD-Scheme` is the fastest. A single delta distribution results with $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$ being a single isotropic Gaussian which has fewer parameters than the factorised Gaussian used in the `FG-Scheme`. A single delta is a very poor approximation of the true posterior, so we would typically use many more samples to form the approximation. With 10 or more samples, the runtime is much slower for the `MOD-Scheme` since now a mixture of Gaussian need to be stored and maintained for $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$, which has many more parameters than a single factorised Gaussian. Finally, both methods have significantly longer runtimes as the dimension and relative entropy increases.

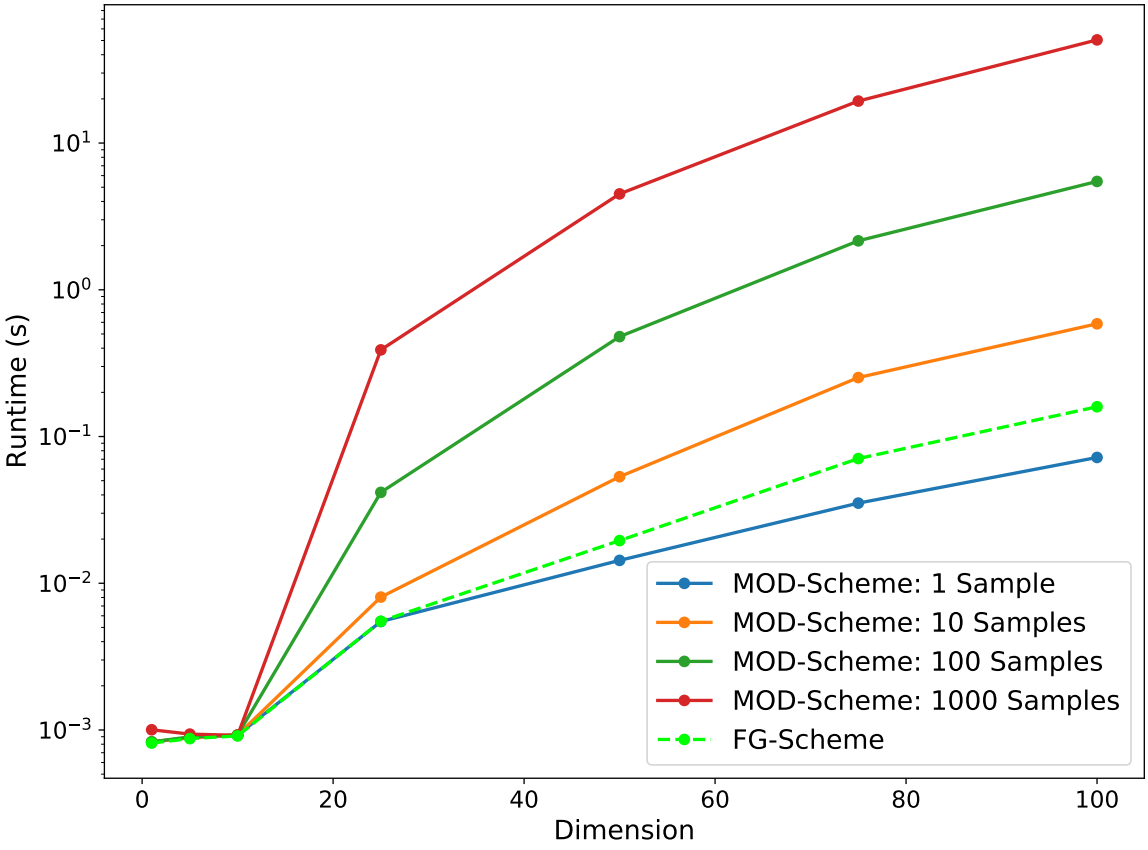


Fig. 3.3 Example comparing Importance Sampling to Greedy Selection. Here we can see that Greedy Selection improves the heuristic measure for bias, but in doing so reduces the sample variance.

3.7 Optimising the Prior Variances

For each auxiliary variable we draw $\exp(\mathrm{KL}\left[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \| p(\mathbf{a}_k)\right])$ samples from the prior, if this value is too large then implementing the scheme is impractical. To this end, we fix a hyper-parameter Ω and aim to keep the relative entropies of each auxiliary variable close to this value, i.e. $\mathrm{KL}\left[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \| p(\mathbf{a}_k)\right] \approx \Omega$. Since we require the sum of the relative entropies to equal the total relative entropy over $\mathbf{z}$, as to not increase the codelength (Equation (3.5)), we use $K = \lceil \mathrm{KL}\left[q(\mathbf{z}) \| p(\mathbf{z})\right] / \Omega \rceil$ auxiliary variables in total. To keep the relative entropies close to Ω, one can optimise the variance terms for the priors of each auxiliary variable.

In previous work, Flamich et al. [2021] optimised the variance according to the objective

$$\begin{aligned} \underset{\sigma_k^2}{\arg\min} \quad & \Big[\mathrm{KL}\left[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \| p(\mathbf{a}_k)\right] - \Omega\Big]^2 + \\ & \Big[\big(\hat{\mathrm{KL}} - \mathrm{KL}\left[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \| p(\mathbf{a}_k)\right]\big) - (K-k-1)\Omega\Big]^2. \end{aligned} \tag{3.22}$$

where, $\hat{\mathrm{KL}} = \mathrm{KL}\left[q\left(\mathbf{z}\right) \| p\left(\mathbf{z}\right)\right] - \sum_{i=1}^{k} \mathrm{KL}\left[q\left(\mathbf{a}_i \mid \mathbf{a}_{1:i-1}\right) \| p\left(\mathbf{a}_i\right)\right]$, is the remaining KL.

The first term in Equation (3.22) ensures the relative entropy of $\mathbf{a}_k$ stays close to Ω and the second term ensures the remaining KL is large enough to ensure the auxiliary varibles have relative entropies close to Ω.

3.7.1 Our proposed optimisation method

Rather than optimise the auxiliary variances independently and in sequence according to Equation (3.22), as in Flamich et al. [2021], we propose a joint optimisation over all the auxiliary variances. We optimise the objective

$$\begin{aligned} \underset{\sigma_{1:K}}{\arg\min} \quad & \sum_{k=1}^{K} \Big[\mathrm{KL}\left[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \| p(\mathbf{a}_k)\right] - \Omega\Big]^2 + \\ & \Big[\big(\hat{\mathrm{KL}} - \mathrm{KL}\left[q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \| p(\mathbf{a}_k)\right]\big) - (K-k-1)\Omega\Big]^2. \end{aligned} \tag{3.23}$$

To ensure the variances sum to the desired amount, we optimise logits and pass them through a softmax function to ensure they sum to 1. Then we multiply the post softmax values by σ^2 where $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma^2 \mathbf{I})$. We found this method of optimisation to be more stable and yield smoother variances plots, likely due to the softmax in the optimisation process.

3.7.2 Caveats of Optimising the Variances

Flamich et al. [2021] found, on image compression problems using VAEs, a power law approximating the remaining variable $\sigma^2 - \sum_{j=1}^{k-1} \sigma_j^2$ by $(K+1-k)^{-0.79}$ was a good estimate of the optimal variances. The power law intuitively makes a lot of sense: by allocating more variance in the first few auxiliary variables, the trajectory can point in the general direction at the start, with smaller steps at the end to make micro-adjustments. We find however, the best settings for the variances are problem-specific; with different target and coding distributions, the optimisation gave markedly different results and sometimes wasn't even optimal.

To demonstrate this, we compress samples with the `MOD-Scheme` on two problems with very different characteristics: the first involves a large target far from the prior and the second with a small target well contained within the prior. We compress 1000 samples for each problem, with optimised variances and uniform variances[9] as a benchmark. We then compute the mean log probability of each compressed sample under the target for the different problems.

One crucial finding, shown in Table 3.1, is that without sub-sampling the compressed samples, the optimised variance performance catastrophically fails on the problem with a small target. The optimised variances place much more variance in the first few auxiliary variables (following an exponential decay as was reported by Flamich et al. [2021]), which means if the initial samples to choose from are poor, there is not enough variance in the final few auxiliary variables to reach the target. However, if the first few auxiliary variables were high quality, taking us close to the target, having smaller variance in the final samples allows for the trajectory to make micro-adjustments, reaching a high-density area within the target. For distant targets, this is almost a non-issue, since each auxiliary variable will be chosen to lie close to the straight line from the prior to the mode, and the probability of overshooting the target is virtually zero. When sub-sampling half of the samples, the optimised variances begin to perform much better on the small target problem. In Figure 3.4, we plot trajectories for the compressed samples. This plot shows the optimised variances can make finer adjustments, whereas the uniform variances take similar size steps. When they overshoot the target, it is harder to make small corrective steps. For the distant target, the uniform variances outperform the optimised ones. In Figure 3.5, we plot the mean trajectory and find that the uniform variances get much closer to the target. The reason for this is that uniform variances maximise the expected euclidean distance of the trajectory - we prove this in Appendix A.3. Ideally, the receiver would use a correctly calibrated prior, so that the samples can easily reach the target. This may be impractical as the sender and receiver must agree to the priors a priori. Therefore, the users of our compression method, should

[9] By uniform, we simply divide the variance of $p(\mathbf{z})$ amongst each auxiliary variable equally.

analyse the relationship between the target and prior to determine how to set the auxiliary prior variances. In our subsequent experiments, the target distributions will be posteriors on weights of Bayesian models, and close to the prior. Thus, we can safely use optimised variances without concern about reaching the target modes.

Table 3.1 Table showing the mean log probabilities of the compressed samples on two problems; one with a small target well contained in the prior, and the other with the target distribution place further away from the prior. In this table, 'Optimised' refers to optimising the auxiliary prior variances according to Equation (3.23), and 'Uniform' simply refers to selecting each auxiliary prior variance to be equal across all auxiliary variables. The 'Uniform' variances serve as a baseline since it is the simplest choice one could make.

Problem	Optimised	Optimised w/ Sub-sampling	Uniform	Uniform w/ Sub-sampling
Small Target within Prior	-10076	6.5	-941.3	-105.4
Distant Target	-399.7	-236.0	-358.2	-209.0

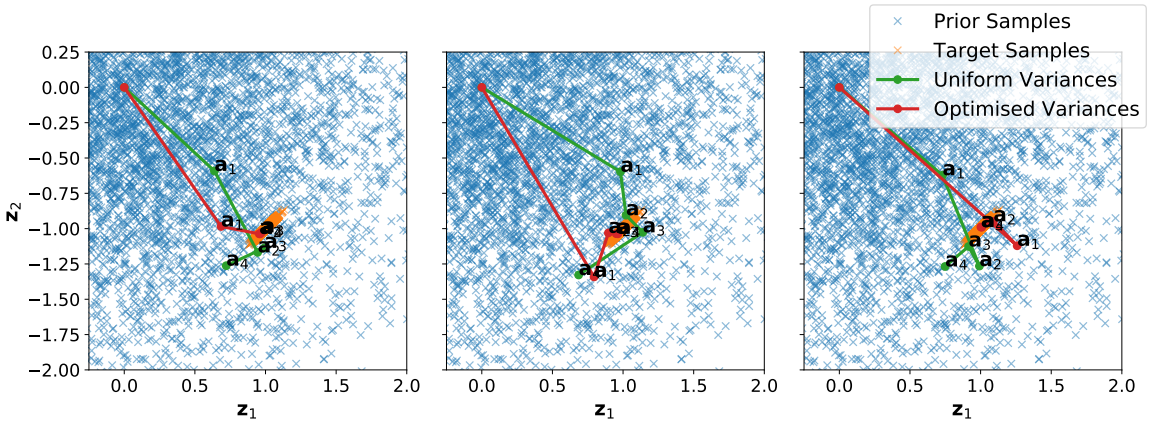


Fig. 3.4 Plot of trajectories for both the 'optimised' and 'uniform' variances of auxiliary variables. It's clear in this plot that the uniform variances often overshoot the target and are unable to refine onto the target, compared to the 'optimised' variances which can, due to the decaying variances. We also indicate the mean of the prior distribution with an unlabelled point to help visualise how the first auxiliary variable is chosen.

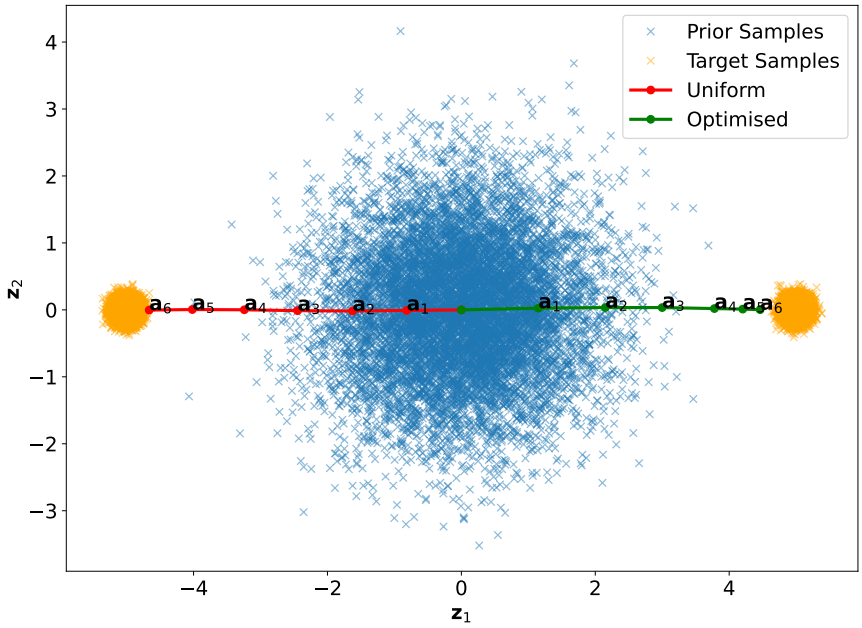


Fig. 3.5 Here we show the trajectory when using both 'optimised' and 'uniform' variances for the auxiliary variables after averaging over 1000 compressed samples. The 'uniform' variances on average reach further than the 'optimised' variances which is to be expected as we prove they maximise the expected euclidean distance for the compressed sample. We use identical targets placed equidistant from the prior distribution to help visualise the difference in distance travelled by the auxiliary variables. We also label the mean of the prior distribution to help see that the uniform variances result in taking equal-sized steps, compared to the optimised variances which take large steps initially and subsequently reduce the step size.

Note: Since the optimal variances are problem-specific, we cannot apply a specific rule like in Flamich et al. [2021]. Hence, the sender and receiver must communicate the variances. To send the variance for each auxiliary variable is far too expensive, and so we chose to approximate the variances with an exponential function of the form $f(x) = a\exp(-bx) + c$. We can then optimise the parameters $a, b, c \in \mathbb{R}$ and communicate these to the receiver. To see that the approximation is reasonable, consider the plot in Figure 3.6. As the relative entropy of the problem grows, the overhead in communicating these parameters becomes negligible, but it does increase the cost of communication considerably for simple problems. In these cases, it may be more reasonable to use uniform variances and draw more samples by increasing ε or use sub-sampling.

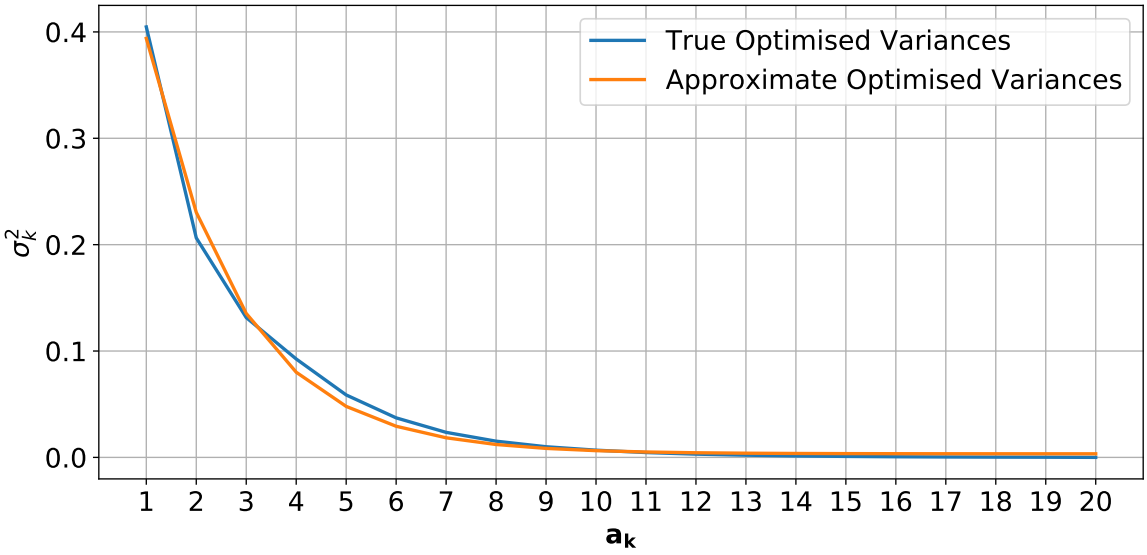


Fig. 3.6 Plot showing the fit of our approximate optimised variances versus the exact optimised variances. This plot shows that the approximation is very close, and requires us to send 3 additional parameters rather than sending all the σ_k^2s.

3.8 Beam-search

At selection time, we cannot ensure the greedily selected auxiliary variable will pair well with the subsequent choices, and throwing away all other candidate samples is wasteful. To remedy this, we implement a beam-search over the potential auxiliary variable trajectory. Formally, we aim to maintain the set of B trajectories $\mathbf{a}_{1:k}$ with the lowest bias. At each step, we combine the current top B trajectories with each of the M trial samples for $\mathbf{a}_k$, and select the top B according to their joint importance weights $\frac{q(\mathbf{a}_1,\ldots,\mathbf{a}_k)}{p(\mathbf{a}_1,\ldots,\mathbf{a}_k)}$. The algorithm for the encoder is shown in Algorithm 5 and decoder in Algorithm 6.

Using a beam-search should result in the final compressed sample $\mathbf{z}$ having a larger value for $\frac{q(\mathbf{z})}{p(\mathbf{z})}$ than the simple greedy scheme. Whilst the beam-search reduces the bias incurred by the importance sampling scheme, there are several downsides. First, the mode bias issues we encountered previously are exaggerated, meaning if we want the samples to accurately represent uncertainty, beam-search may be detrimental since the variance in the final samples will decrease. Furthermore, there is a significant increase in the computational demands when running the beam-search, see Figure 3.7, where we consider the same problem set up as in Section 3.6 and test the `FG-Scheme` with numerous beam-widths. Note that the `MOD-Scheme` and `KDE-Scheme` would fare even worse in this comparison since they have more computational demands themselves. Overall, the sender must be careful in their

deployment of the beam-search not to under represent uncertainty in their compressed samples and also to balance the computational costs incurred by this method.

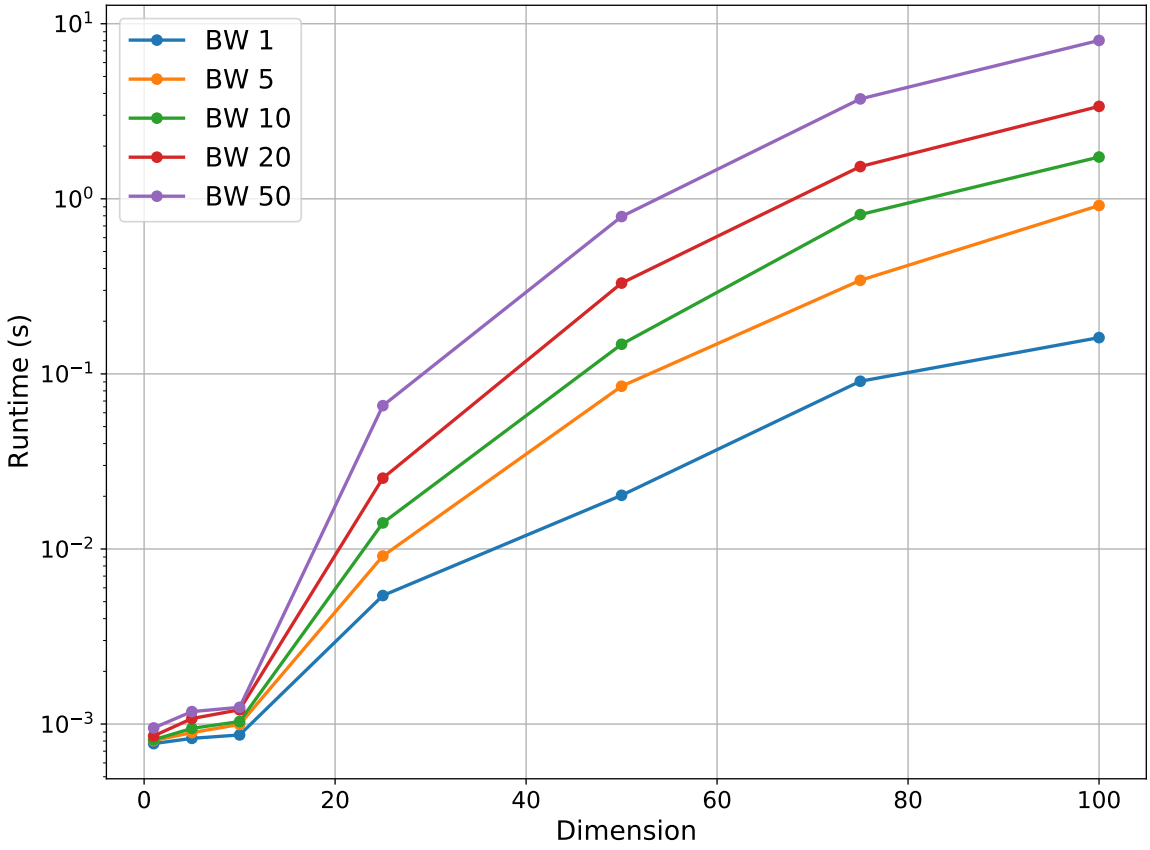


Fig. 3.7 Plot of runtimes using the `FG-Scheme` with various beam-searches. The computationally complexity grows significantly with each additional beam-width due to many more distributions must be maintained at each step of the algorithm.

Algorithm 5: iREC Encoder w/ Beam-search.

Data: $q(\mathbf{z})$

Result: $(i_1, ..., i_K)$

$S_0 \leftarrow \{()\}$

$K \leftarrow \left\lceil \frac{\mathrm{KL}[q(\mathbf{z}) \| p(\mathbf{z})]}{\Omega} \right\rceil$

$M \leftarrow \lceil \exp(\Omega(1+\varepsilon)) \rceil$

for $k = 1 : K$ **do**

$\mathbf{a}_{k_1}, ..., \mathbf{a}_{k_M} \overset{R}{\sim} p(\mathbf{a}_k)$

$\hat{S}_k \leftarrow S_{k-1} \times \{1, ..., M\}$

$S_k \leftarrow \underset{(j_1,...,j_k) \in \hat{S}_k}{\mathrm{argtop}_B} \left\{ \frac{q\left(a_{1_{j_1}}, ..., a_{k_{j_k}}\right)}{p\left(a_{1_{j_1}}, ..., a_{k_{j_k}}\right)} \right\}$

end

$(i_1, ..., i_K) \leftarrow \underset{(j_1,...,j_K) \in \hat{S}_K}{\arg\max} \left\{ \frac{q\left(f\left(a_{1_{j_1}}, ..., a_{K_{j_K}}\right)\right)}{\left(f\left(a_{1_{j_1}}, ..., a_{K_{j_K}}\right)\right)} \right\}$

return $(i_1, ..., i_K)$

Algorithm 6: iREC Decoder w/ Beam-search.

Data: $(i_1, ..., i_K)$

Result: $\mathbf{z}$

$M \leftarrow \lceil \exp(\Omega(1+\varepsilon)) \rceil$

for $k = 1 : K$ **do**

$\mathbf{a}_{k_1}, ..., \mathbf{a}_{k_M} \overset{R}{\sim} p(\mathbf{a}_k)$

end

$\mathbf{z} \leftarrow f(\mathbf{a}_{1_{i_1}}, ..., \mathbf{a}_{K_{i_K}})$

return $\mathbf{z}$

3.9 Choosing Ω

The sender must determine the hyper-parameter Ω, which determines how much of the relative entropy is divided between each auxiliary variable. A small value of Ω leads to more biased samples, but using a large value for Ω causes the computational issues which motivated the need for the auxiliary variable method in Section 3.3 - drawing the required number of samples is infeasible. In Figure 3.8, the trajectory using $\Omega = 1$ is very noisy as the scheme has very few samples to pick from for each auxiliary variable. With $\Omega = 10$ there are $\lceil \exp(10) \rceil$ many samples from $p(\mathbf{a}_k)$ to select from, so choosing samples that form a trajectory straight toward the target mode is more probable; yielding a smoother line. This

simple example suggests using larger values of Ω is preferred, but as the dimension and the relative entropy of the problem grows, this becomes computationally infeasible due to memory constraints. As such, we strike a balance by using $\Omega = 5$ similarly to Flamich et al. [2021].

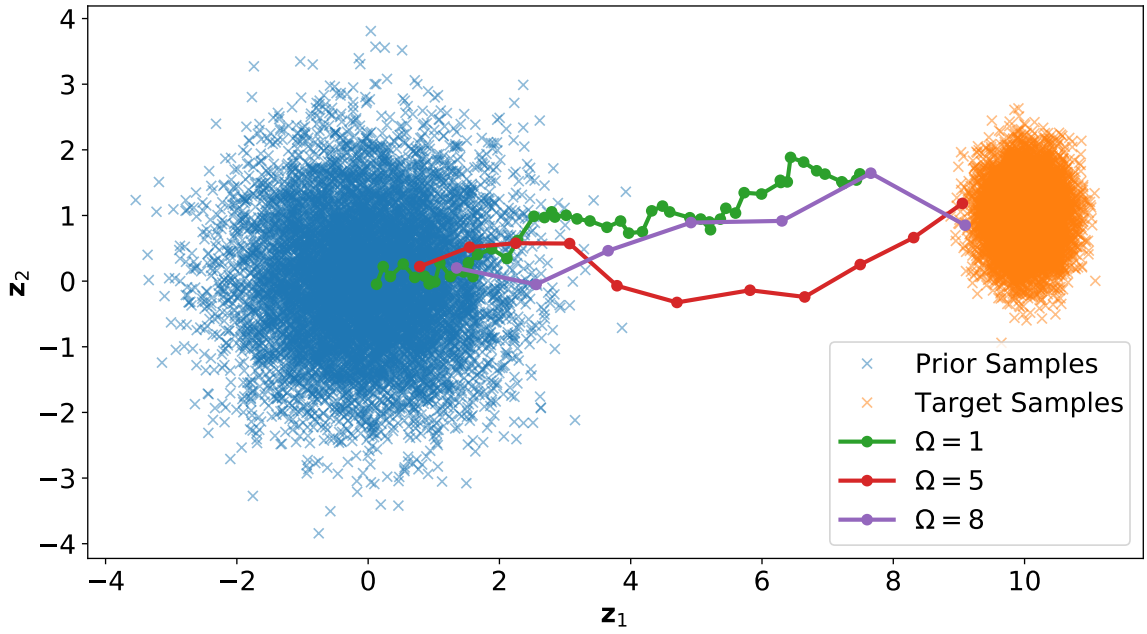


Fig. 3.8 Example comparing differences in the choice of Ω, with larger values the trajectory is smoother and the samples can get closer to the mode - the trade-off is too much memory requirements if the relative entropy is large.

Chapter 4

Experiments

In the previous chapters, we discussed why compressing samples from the true posterior should yield better performance than using a variational approximation. In this section we confirm this in practice on a variety of problems. At a high level, our problem setting is a regression task, where the sender has observed data $\mathscr{D}$ and trained a model $\mathscr{M}$. Once the model is trained, the sender will implement the iREC compression algorithm and send multiple compressed samples $\mathbf{w}_1, ..., \mathbf{w}_N \sim q(\mathbf{w} \mid \mathscr{D})$ to the receiver. The receiver subsequently decodes the samples and uses them to make predictions on the regression data. First, we explain how we make predictions and measure performance. Then we discuss the problem-specific methodology and analyse the results. Specific hyper-parameters for training, such as the learning rates or number of samples for HMC can be found in Appendix B.

4.1 Making Predictions and Measuring the Data-fit

Bayesian models make predictions by marginalising over all possible settings for the model weights. For many posterior distributions this is infeasible, and so MCMC estimates are used. As such, we compress multiple samples from our posterior and combine the predictions, using each sample to approximate the marginalisation. To form the predictive distribution, we can define the model as a function of the weights $\mathscr{M}[\mathbf{w}]$. Then for a set of weights $\mathbf{w}_1, ..., \mathbf{w}_N$, and test points $\{\mathbf{x}_t, y_t\}_{t=1}^T$, we compute the set of predictions $\{\mathscr{M}[\mathbf{w}_i](\mathbf{x}_t)\}_{t=1}^T$ for each weight setting $\mathbf{w}_i$. For each input-output pair $\{\mathbf{x}_t, y_t\}$, we consider the set of all predictions $\{\mathscr{M}[\mathbf{w}_i](\mathbf{x}_t)\}_{i=1}^N$ and form the following mixture of Gaussians:

$$g(y_t) = \frac{1}{N} \sum_{i=1}^{N} \mathscr{N}(y_t \mid \mathscr{M}[\mathbf{w}_i](\mathbf{x}_t), \sigma^2), \tag{4.1}$$

where σ^2 is assumed to be the known additive noise in the data. To evaluate the performance, we consider the density of the target y_t under its corresponding GMM, i.e. $g(y_t)$. For each target we get a likelihood, and our predictive likelihood over the whole test-set is simply the product of each likelihood. In practice, log probabilities were considered so we simply sum the log-likelihoods,

$$\text{log-likelihood} = \sum_{t=1}^{T} \log g_t(y_t), \tag{4.2}$$

and this is the metric we use to compare the performance of our transmitted samples unless stated otherwise.

4.2 Measuring the Compression

In Section 2.1.2, we described how to use the Zeta distribution to compress integers in the case where their underlying distribution is unknown. With iREC, we send integers whose underlying distribution is unknown, and the only information we have is that the largest value they can take is $M = \lceil \exp(\mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})]\rceil$. Therefore, we can apply the method described in Section 2.1.2 to encode the integers for our setting. Recall that the sender and receiver needed access to $\mathbb{E}[\log\Theta]$ to create the correct Zeta distribution for encoding and decoding. In our case, we instead require that the sender and receiver both know the value of M so they can set up the corresponding Zeta distribution - communicating M can be done[1] with $\mathscr{O}(1)$ cost. Given the decoder knows M, it follows that our integer to transmit Θ lies in the set $\{1,...,M\}$, so

$$\log\Theta \leq \log M \leq \mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})] + 1. \tag{4.3}$$

This means that

$$\mathbb{E}[\log\Theta] \leq \mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})] + 1, \tag{4.4}$$

and so using the Zeta distribution with parameter $s = 1 + \frac{1}{\mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})]}$ yields an average coding efficiency for transmitting Θ of

$$\mathbb{E}[\log\Theta] + \log(\mathbb{E}[\log\Theta] + 1) + 1 \leq \mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})] + 1 + \log(\mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})] + 2) + 1 \tag{4.5}$$

$$= \mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})] + \log(\mathrm{KL}[q(\mathbf{w})\|p(\mathbf{w})] + 2) + \mathscr{O}(1). \tag{4.6}$$

[1] We can send this value with a lower precision (8 or 16 bits) and so this is a cheap piece of additional information to transmit as opposed to sending a specific length of our codewords which would be much more expensive.

Therefore, combining this with sending the $\mathscr{O}(1)$ message containing the value for M, we get an expected codelength bounded by[2],

$$\text{Avg. Codelength} \leq \text{KL}\left[q(\mathbf{w}) \| p(\mathbf{w})\right] + \log(\text{KL}\left[q(\mathbf{w}) \| p(\mathbf{w})\right] + 2) + \mathscr{O}(1). \tag{4.7}$$

Implementing the auxiliary scheme incurs a small overhead introduced when dividing the relative entropy between $K = M/\Omega$ many auxiliaries and so $K \times \Omega$ may not be exactly equal to M. As the relative entropy gets larger this discrepancy becomes negligible, and we can use Equation (4.7) as a measure of our average compression efficiency to send a sample $\mathbf{w} \sim q(\mathbf{w})$ with iREC. For the subsequent results, we report the expected codelength per sample to make all the compression sizes across the problems comparable.

4.3 Bayesian Linear Regression

4.3.1 Problem Setting

Consider the problem of compressing samples from the posterior distribution of Bayesian linear regression model. In this model, we assume the regression output $y_n \in \mathbb{R}$ is generated given the feature input $\mathbf{x}_n \in \mathbb{R}^D$ by

$$y_n = \mathbf{w}^T \mathbf{x}_n + \varepsilon_n, \tag{4.8}$$

where $\varepsilon_n \sim \mathscr{N}(0, \sigma^2)$ with known noise σ^2 and $\mathbf{w}$ is a vector of regression coefficients. By placing a Gaussian prior on the weights such that $\mathbf{w} \sim \mathscr{N}(\mathbf{w} \mid \mathbf{0}, \alpha^{-1}\mathbf{I})$, the posterior distribution becomes a Gaussian $p(\mathbf{w} \mid \mathscr{D}) = \mathscr{N}(\mathbf{m}, \mathbf{V})$ where

$$\begin{aligned} \mathbf{V} &= \left(\mathbf{X}^T \mathbf{X} \sigma^{-2} + \alpha \mathbf{I}\right)^{-1} \\ \mathbf{m} &= \mathbf{V} \mathbf{X}^T \mathbf{y} \sigma^{-2}, \end{aligned} \tag{4.9}$$

here $\mathbf{X}$ is a matrix with rows $\mathbf{X}_n = [\mathbf{x}_n \quad 1]$ and $\mathbf{y} = (y_1, ..., y_N)^T$. Using this posterior we can draw samples and form both a mixture of deltas and a kernel density estimate to compress samples with the `MOD-Scheme` and `KDE-Scheme` respectively. The `FG-Scheme` targets a factorised Gaussian, so we approximate the correlated Gaussian posterior by

$$q(\mathbf{w}) = \prod_{d=1}^{D} \mathscr{N}(w_d \mid \nu_d, \rho_d) = \mathscr{N}(\mathbf{w} \mid \nu, \rho). \tag{4.10}$$

[2] In the case where we use the de-biasing parameter ε we can simply replace the relative entropy in Equation (4.7) by $\text{KL}\left[q(\mathbf{w}) \| p(\mathbf{w})\right](1+\varepsilon)$.

We choose the parameters to minimise the relative entropy given by

$$\mathrm{KL}\left[q(\mathbf{w})\|p(\mathbf{w}\mid\mathscr{D})\right]=\frac{1}{2}\left[\log\frac{|\mathbf{V}|}{|\operatorname{diag}(\rho)|}-D\operatorname{trace}(\mathbf{V}^{-1}\operatorname{diag}(\rho))+(\mathbf{m}-\nu)^{T}\mathbf{V}^{-1}(\mathbf{m}-\nu)\right] \tag{4.11}$$

where $\operatorname{diag}(\rho)$ is the diagonal matrix with entries given by ρ, and $\operatorname{trace}(A)$ is the sum of the diagonal entries of matrix A. From Equation (4.11) it's clear the value for ν which minimises the expression is simply $\mathbf{m}$. For ρ, we differentiate the expression to obtain

$$\frac{\mathrm{dKL}\left[q(\mathbf{w})\|p(\mathbf{w}\mid\mathscr{D})\right]}{\mathrm{d}\operatorname{diag}(\rho)}=-\operatorname{diag}(\rho)^{-1}+\operatorname{diag}(\mathbf{V}^{-1}), \tag{4.12}$$

which is at zero for $\rho=\operatorname{diag}(\mathbf{V}^{-1})$.

With these posterior distributions, the sender can compress samples from both $p(\mathbf{w}\mid\mathscr{D})$ and $q(\mathbf{w})$ using the corresponding iREC compression schemes.

4.3.2 Generating the Data-set

Our data-set is generated by sampling a setting of the weights $\hat{\mathbf{w}}\sim p(\mathbf{w})=\mathscr{N}(\mathbf{w}\mid\mathbf{0},\mathbb{I})$ and input targets $\mathbf{x}\sim\mathscr{N}(\mathbf{x}\mid\mathbf{0},\mathbb{I})$. We then compute the regression target

$$y=\hat{\mathbf{w}}^{T}\mathbf{x}+\sigma\varepsilon, \tag{4.13}$$

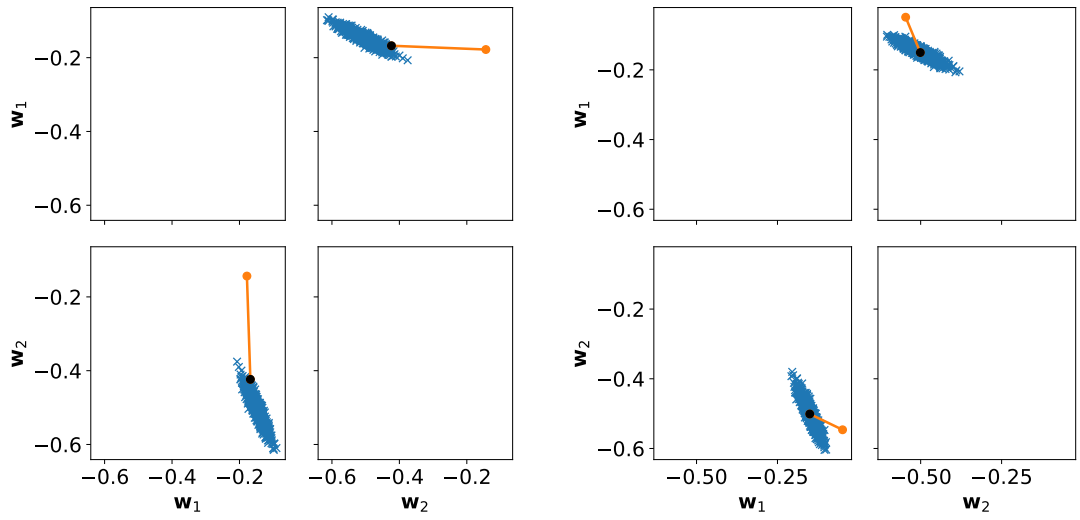
where $\varepsilon=\mathscr{N}(0,1)$ and σ is a specific noise term that we vary depending on the dimension of the problem we consider. Rather than viewing the dimensionality in terms of the dimension of $\mathbf{x}\in\mathbb{R}^{N}$, we report the dimension of the weight-space, since we are concerned with compressing samples from $p(\mathbf{w}\mid\mathscr{D})$. The dimension is therefore given by $N+1$ since we have a weight for each component of $\mathbf{x}$, plus a bias term. For 2D, we chose $\sigma=0.01$, and for all others $\sigma=0.1$. This is because linear regression is a very easy task in lower dimensions and so we need to set up the problem in such a way to still preserve uncertainty over predictions. We use a 50:50 training and test split, with the number of total data-points equal to $2\times D$ where $\mathbf{w}\in R^{D}$. This ensures the problem leads to the model having significant uncertainty in its predictions and correlations its posterior $p(\mathbf{w}\mid\mathscr{D})$. If we use too many training points or have too small signal noise, the posterior will collapse onto a delta mass and not make for an interesting compression problem. After computing $p(\mathbf{w}\mid\mathscr{D})$, we find the factorised Gaussian approximation to the posterior derived above, denoted $q(\mathbf{w})$, and use this to compress samples with the `FG-Scheme`. Then, we draw 50 empirical samples from $p(\mathbf{w}\mid\mathscr{D})$ and use these to form the mixture of deltas so that we can compress samples

with the `MOD-Scheme`. For each problem we consider $\varepsilon \in [-0.3, 0.4]$, this varies how many samples we draw per auxiliary variable which directly relates to the compression size of each sample. The decimal value of ε represents the overhead in compression size in comparison to the optimal value which is given the relative entropy[3]. Since our data was generated with a sample from the prior, the posterior is close-by. Consequently, we don't encounter issues reaching the mode as mentioned in Chapter 3, and can implement the optimised variances routine. Additionally, we use $\Omega = 5$ and a beam-search with $B = 5$. After compressing multiple samples, we construct the predictions detailed in Section 4.1 and measure the coding efficiency and log-likelihood.

4.3.3 Results

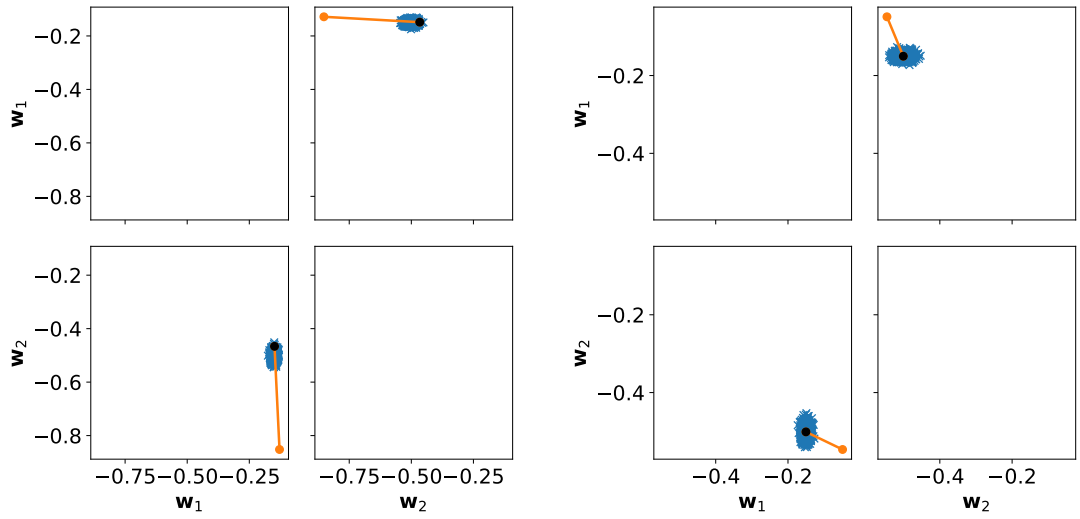
First, we present the results on the 2D problem, with the training and test-set results shown in Figure 4.3. For this problem, samples from the true posterior outperform the samples from the factored Gaussian variational approximation on both the training and test-sets. On the training-set, for every given compression rate, the `MOD-Scheme` samples outperform the `FG-Scheme` samples. For both schemes, the test-set performance begins to degrade as we relax the constraints on the compression size by increasing ε. This empirically demonstrates the concerns raised in Chapter 3, specifically that due to our choice of greedily picking the auxiliary samples and further implementing a beam-search, our samples are biased to the mode. In Figures 4.1 and 4.2, we plot example trajectories for both schemes on this problem with different compression budgets. With a large compression allowance, virtually all the compressed samples lie on the mode, meaning they under-represent the uncertainty in the posterior distribution. This results in degradation in the test-set performance but asymptotically increasing training-set performance as the models begin overfitting the training data. This demonstrates the extra care the sender must take on this task, since reaching the mode is sub-optimal when your predictions need to represent uncertainty.

[3] i.e. if $\varepsilon = 0.2$ then we have an overhead of 20%.



(a) Small compression allowance. (b) Large compression allowance.

Fig. 4.1 Plot of the trajectory of `MOD-Scheme` on 2D linear regression task with a large and small compression allowance. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.



(a) Small compression allowance. (b) Large compression allowance.

Fig. 4.2 Plot of the trajectory of `FG-Scheme` on 2D linear regression task with a large and small compression allowance. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.

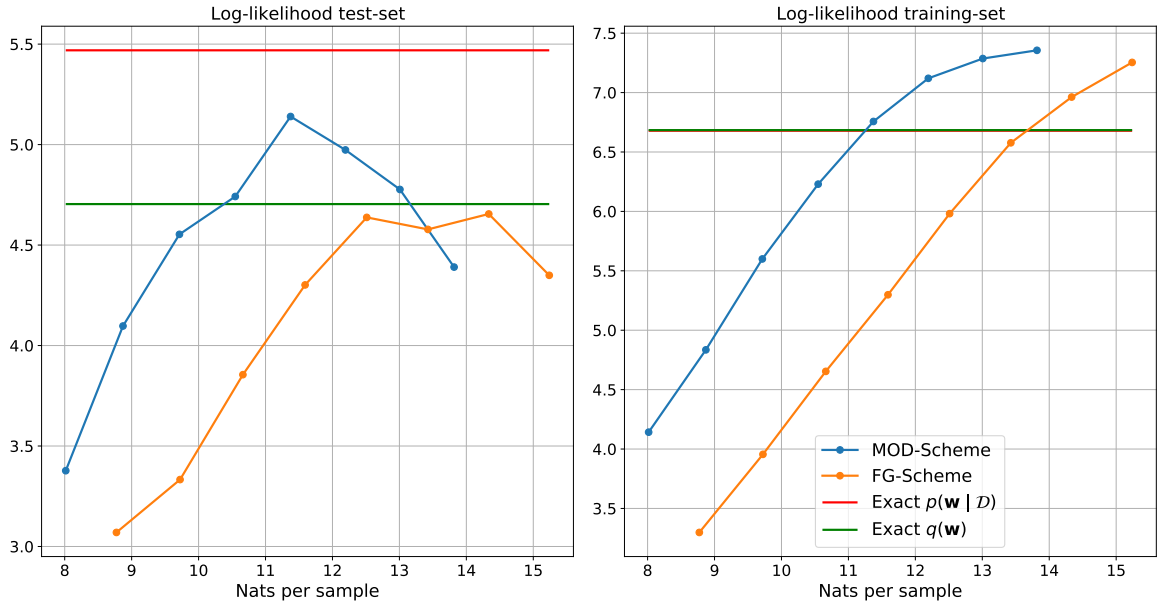


Fig. 4.3 Plot depicting the performance of Bayesian linear regression models on a 2D problem. We plot the performance of uncompressed samples from the true posterior and approximate posterior and compare these to compressed samples. Given enough overhead through the bias reducing parameter ε, our compressed samples experience mode bias which results in overfitting on the training set and worse test-set performance.

As we increase the dimensionality of the problem, two things occur: (1) more correlations appear in the resulting Gaussian posterior $p(\mathbf{w} \mid \mathscr{D})$ and (2) the relative entropies increase so the compression task becomes harder. We can examine the differences between the two schemes by considering a 10D problem and plotting some sample trajectories in Figures 4.6 and 4.7. From these plots, it's clear that the factorised Gaussian approximation is a poor representation of the true posterior, the targets are far too small and so our predictions will be overconfident. It even looks like the approximation is isotropic, but examining the posterior covariance matrix proves this to not be the case. These examples translate to performance on the data-set, in Figure 4.4, the drop-off in performance between samples from the true posterior and the variational approximation is much greater. Also, the `FG-Scheme` compressed samples perform better with fewer nats per sample (small values for ε). With few nats, the compressed samples have much more noise compared to true samples, and since the variational approximation is poor and underrepresents the uncertainty, adding noise to the samples is beneficial. This is further supported by a steep degradation in the performance as the nats per sample increases. The largest compression size yields performance similar to uncompressed samples from the factorised Gaussian.

Since the compression task gets harder as the dimension grows, the schemes need larger compression sizes to experience mode biasing. Despite this, on the 10D problem both schemes experience overfitting when the nats per sample is large enough. Increasing the dimension to 25D in Figure 4.5, the `MOD-Scheme` stops overfitting on the training set. The posterior distribution is higher dimensional and much harder to compress samples from, so landing directly on the mode is less probable. Furthermore, the performance gap between using true samples and the variational approximation widens compared to the 10D problem. Having more dimensions creates more correlations and so the approximation gets cruder.

Overall, these results demonstrate the `MOD-Scheme` is outperforming the `FG-Scheme` in the Pareto sense: for every fixed compression size, the test-set performance is better, and for every fixed test-set performance, the compression rate is higher. We give more examples of results in Appendix B, and note the trends discussed above were common amongst all our tested problems.

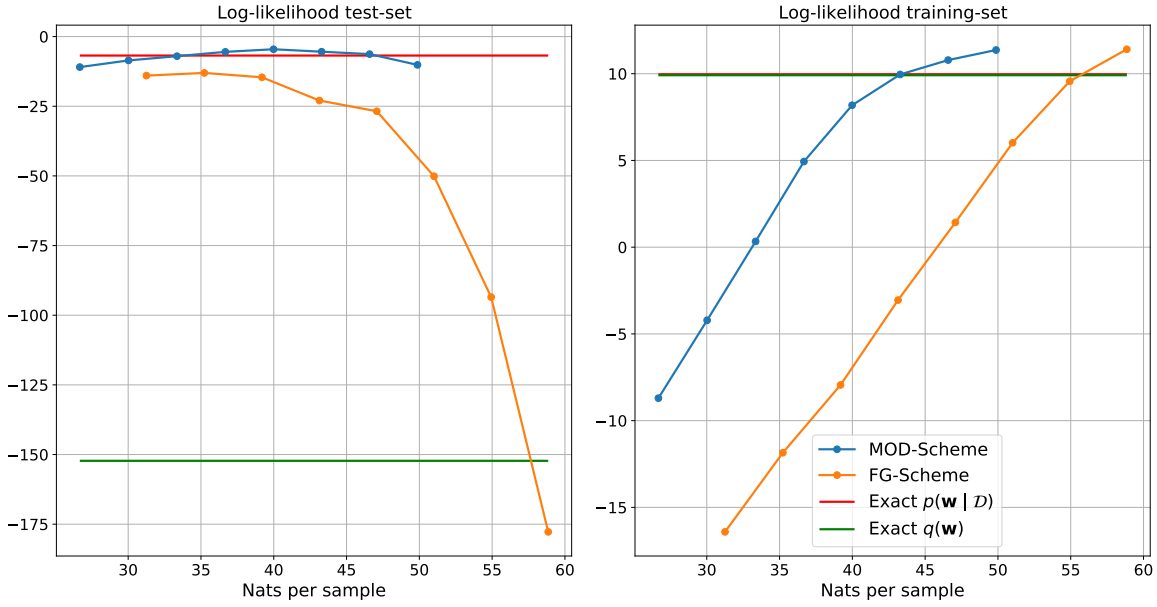


Fig. 4.4 Plot depicting the performance of Bayesian linear regression models on a 10D problem. We plot the performance of uncompressed samples from the true posterior and approximate posterior and compare these to compressed samples. Since this task is harder and the true posterior more correlated, the `MOD-Scheme` performs much better on the test-set. The `FG-Scheme` gets worse with more nats since it gets closer to the poor factorised Gaussian approximation.

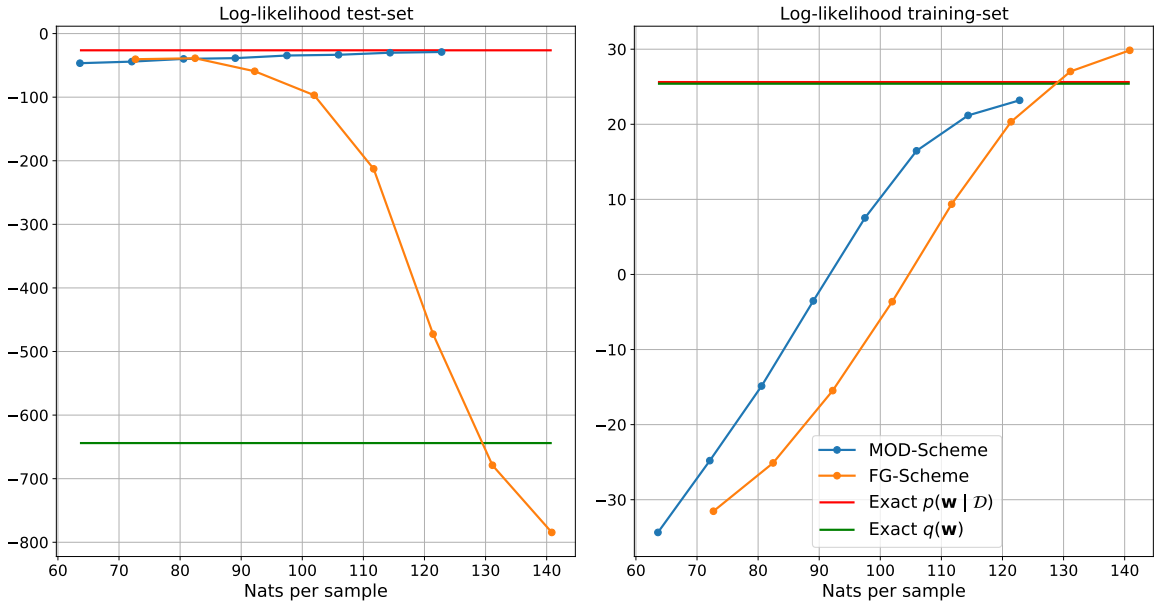


Fig. 4.5 Plot depicting the performance of Bayesian linear regression models on a 25D problem. We plot the performance of uncompressed samples from the true posterior and approximate posterior and compare these to compressed samples. Since this task is harder and the true posterior more correlated, the `MOD-Scheme` performs much better on the test-set. The `FG-Scheme` gets worse with more nats since it gets closer to the poor factorised Gaussian approximation.

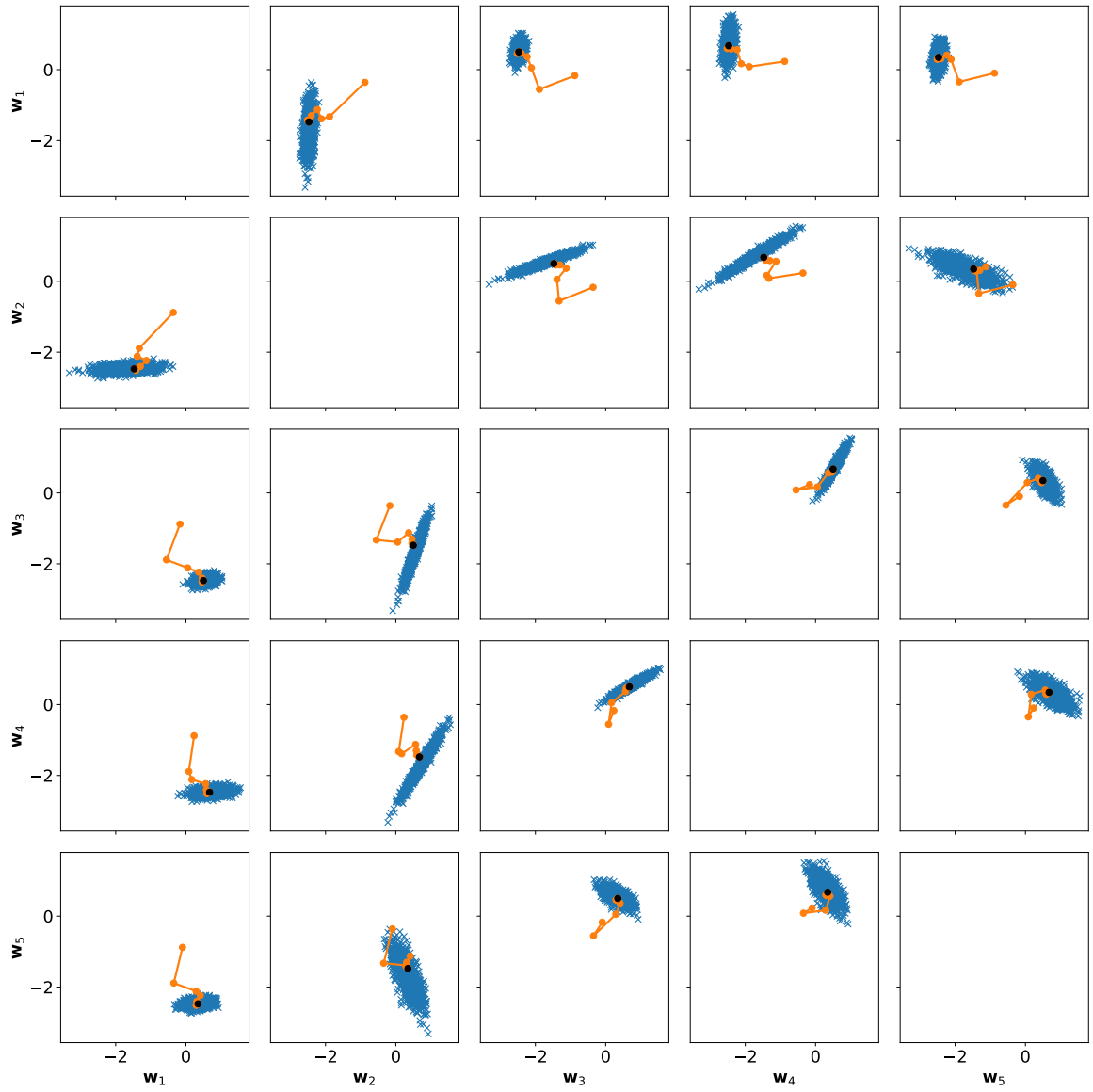


Fig. 4.6 Plot of the trajectory of a subset of the weights communicated using the `MOD-Scheme` for the 10D linear regression task. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.

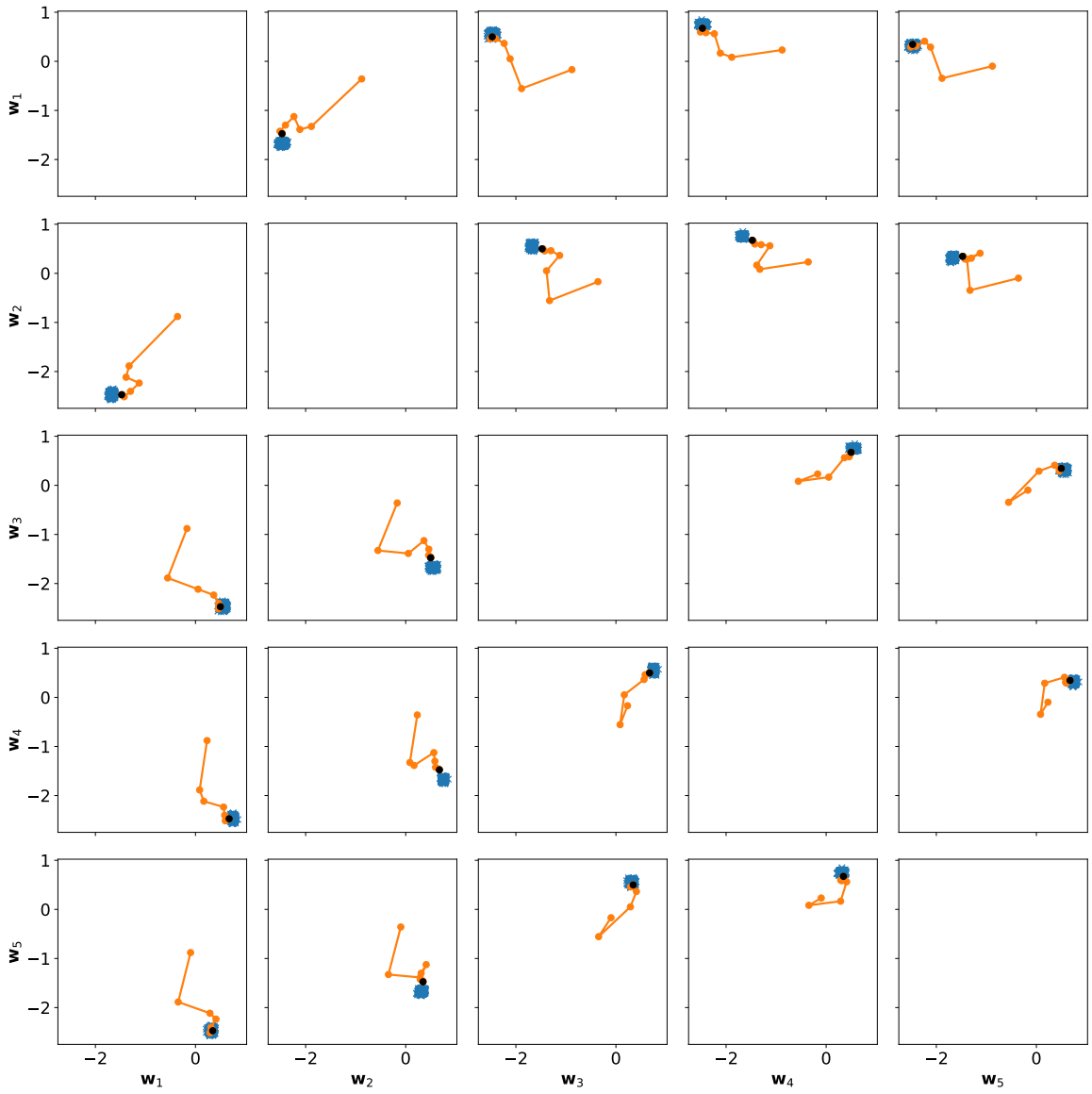


Fig. 4.7 Plot of the trajectory of a subset of the weights communicated using the `FG-Scheme` for the 10D linear regression task. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.

4.4 BNN Regression - Toy Problem

4.4.1 Defining the problem

Now consider using a Bayesian Neural Network on a regression task. Define the output of the neural network with weights $\mathbf{w}$ at input feature $\mathbf{x}$ by $\mathscr{F}_{\mathbf{w}}(\mathbf{x})$. Further, we make the assumption that the conditional distribution $p(y \mid \mathbf{x})$ is Gaussian, where the mean is given by the output of the Neural Network at $\mathbf{x}$ and that the noise is known to be σ^2, this results in the likelihood

$$p(y_n \mid \mathbf{x}_n, \mathbf{w}, \sigma^2) = \mathscr{N}(y_n \mid \mathscr{F}_{\mathbf{w}}(\mathbf{x}), \sigma^2). \tag{4.14}$$

We place the same Gaussian prior over the weights as for the Bayesian Linear Regression setting,

$$p(\mathbf{w} \mid \alpha) = \prod_{n=1}^{N} \mathscr{N}(\mathbf{w} \mid \mathbf{0}, \alpha^{-1}\mathbf{I}). \tag{4.15}$$

The likelihood over the full data-set is defined

$$p(\mathscr{D} \mid \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} \mathscr{N}(y_n \mid \mathscr{F}_{\mathbf{w}}(\mathbf{x}), \sigma^2). \tag{4.16}$$

The posterior distribution over the weights $\mathbf{w}$ is

$$p(\mathbf{w} \mid \mathscr{D}, \alpha, \sigma) \propto p(\mathscr{D} \mid \mathbf{w}, \sigma^2) p(\mathbf{w} \mid \alpha). \tag{4.17}$$

Due to non-linearities in the neural network, the function $\mathscr{F}_{\mathbf{w}}(\mathbf{x})$ depends non-linearly on $\mathbf{w}$, therefore the posterior distribution is intractable. As discussed in Section 2.2, we can perform inference using Variational Inference or with Markov chain Monte Carlo using Hamiltonian Monte Carlo. Moreover, the β training objectives allow us to trade-off data fit and model compressibility. We learn a factorised Gaussian approximation $q_\phi(\mathbf{w})$ with VI and apply the `FG-Scheme` to compress samples from it. Using HMC, we draw a set of samples from $p(\mathbf{w} \mid \mathscr{D})$ and use these to form our mixture of deltas; this then lets us implement the `MOD-Scheme` to compress samples of $\mathbf{w}$.

4.4.2 Approximating the Relative Entropy

The number of candidate samples $\mathbf{a}_k^{(1)}, \mathbf{a}_k^{(2)}, \ldots \sim p(\mathbf{a}_k)$ used per auxiliary variable for the `MOD-Scheme` is given by the relative entropy between the true posterior and the prior divided by the hyper-parameter Ω. Since the true posterior is intractable, we cannot compute the relative entropy in closed-form. Furthermore, MCMC estimates using HMC are high variance

and expensive to compute[4]. Instead, we fit a Gaussian KDE using a subset of 500 samples from HMC and compute the relative entropy between the KDE and the prior to determine how many samples to draw from $p(\mathbf{a}_k)$. To fit the KDE, we optimise the β-ELBO with respect to a shared isotropic noise parameter σ^2_{KDE} (ensuring to use the same β as the potential function in Equation (2.30) when running HMC). Given a subset $\{w_t\}_{t=1}^T$ of the HMC samples, we define the KDE

$$g(\mathbf{w}) = \frac{1}{T}\sum_{t=1}^{T} \mathcal{N}(\mathbf{w} \mid \mathbf{w}_t, \sigma^2_{\text{KDE}}\mathbf{I}), \tag{4.18}$$

and train it according to

$$\beta\text{-ELBO} = \sum_{n=1}^{N} \mathbb{E}_g\left[\log p\left(y_n \mid \mathbf{w}, \mathbf{x}_n\right)\right] - \beta \text{KL}\left(g(\mathbf{w}) \| p(\mathbf{w} \mid \alpha)\right). \tag{4.19}$$

In addition to using the KDE to determine how many samples to draw, we can also use it to compress samples using the `KDE-Scheme`. Since the KDE approximation is used to compute relative entropy for both schemes, their corresponding compression efficiency will be the same.

4.4.3 Data-set and Architecture

For this task, we use a BNN with 1 hidden layer and 3 hidden nodes. We use tanh activation functions and the input and output sizes are 1. Therefore, both the input data $\{x_n\}_{n=1}^N$ and the corresponding regression targets $\{y_n\}_{n=1}^N$ are one dimensional. Similarly to the previous task, we generate our data-set by sampling a setting of the weights $\hat{\mathbf{w}} \sim p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \mathbf{I})$, computing $\mathscr{F}_{\hat{\mathbf{w}}}(\mathbf{x})$ and adding the observational signal noise $\sigma^2 = \frac{1}{100}$. To create our training/test split, we sample two clumps of training points, leaving gaps in-between. The generated data-set is visualised in Figure 4.8; the ground truth function from the sampled BNN is showed with the training and test-sets over-layed. The rationale between having 'gaps' in our data is that Mean-Field Variational Inference methods are notoriously poor at dealing with 'in-between' uncertainty and this will help demonstrate why using exact samples from $p(\mathbf{w} \mid \mathscr{D})$ improves performance [Foong et al., 2019].

Finally, to compress the samples we set $\Omega = 5$, $\varepsilon = 0.2$ and used optimised variances. Despite our BNN having relatively few parameters, we were unable to use a beam-search. We choose the final sample by greedily selecting the candidate $\mathbf{a}_K$ which maximises the likelihood of the data under the compressed model. So for each candidate sample we have to load the weights

[4]Since we only know the density up to a normalisation constant, the MCMC estimates require the computation of $p(\mathscr{D})$ which yields a high variance estimator.

in a neural network and compute a forward pass. This computation gets extremely slow when combined with a beam-search as we have to compute $B \times M$ forward passes where B is the beam-width and $M = \lceil \exp(\Omega(1+\varepsilon)) \rceil$. This demonstrates that in its current state our method has trouble being applied to typical models which could have millions or more parameters.

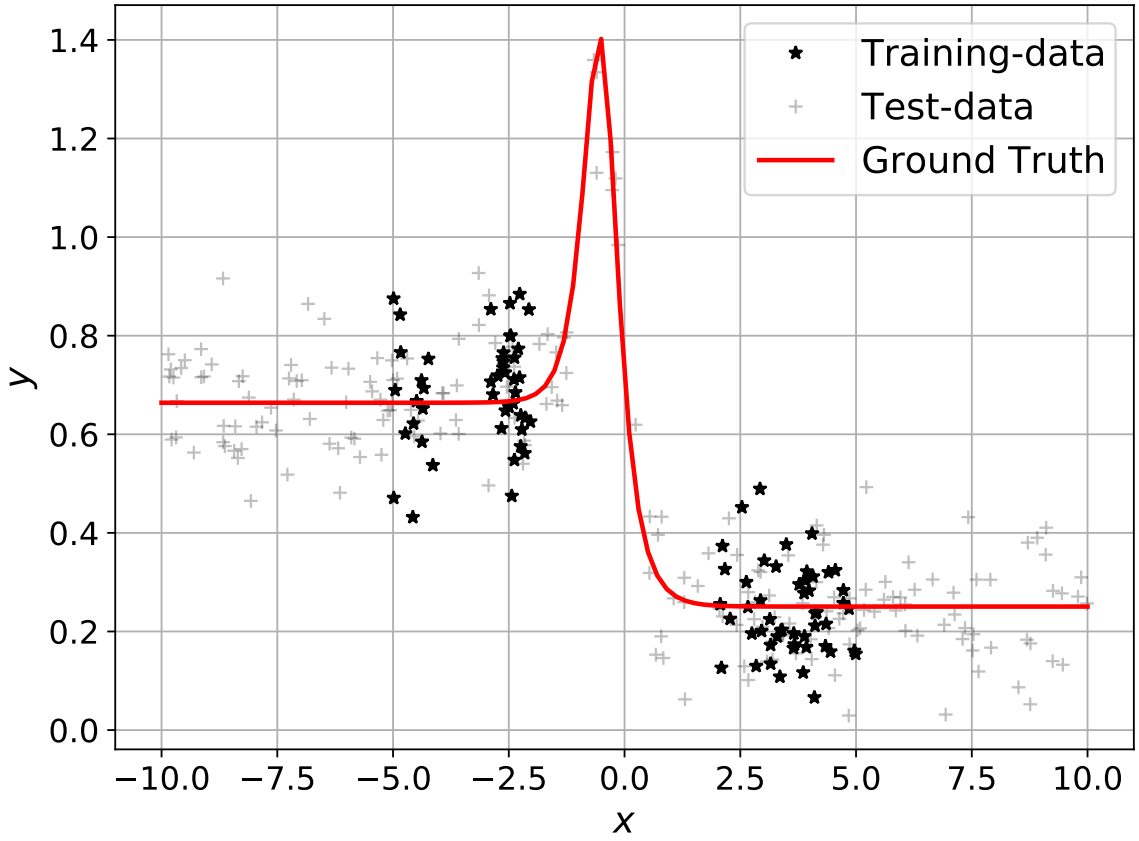


Fig. 4.8 Plot showing the training and test-set for our toy problems with BNNs. The ground truth is a sample from our weight prior, and the training data was created to include gaps to demonstrate both 'in-between' and extrapolation uncertainty.

4.4.4 Results

In Figure 4.9, we give predictive distributions for all three methods when using $\beta = 1$. Here, the uncompressed predictions from samples with HMC and KDE have much more uncertainty compared to the uncompressed factorised Gaussian samples; particularly in extrapolation. Furthermore, the predictive distribution using the `MOD-Scheme` compressed samples has more uncertainty compared to the `KDE-Scheme`. As the posterior distribution gets more complex and higher dimensional, more exact samples may be required for the

mixture of deltas to capture the shape of the posterior. The Gaussian KDE alleviates some of these issues by adding a small amount of noise to each delta, making the approximation less sparse. This results in the scheme being able to target areas 'in-between' the exact samples since they are given probability mass. The `MOD-Scheme` is unable to target these zones since they are given no mass whatsoever, resulting in the trajectories being constrained to directly target the empirical samples, which may be sub-optimal.

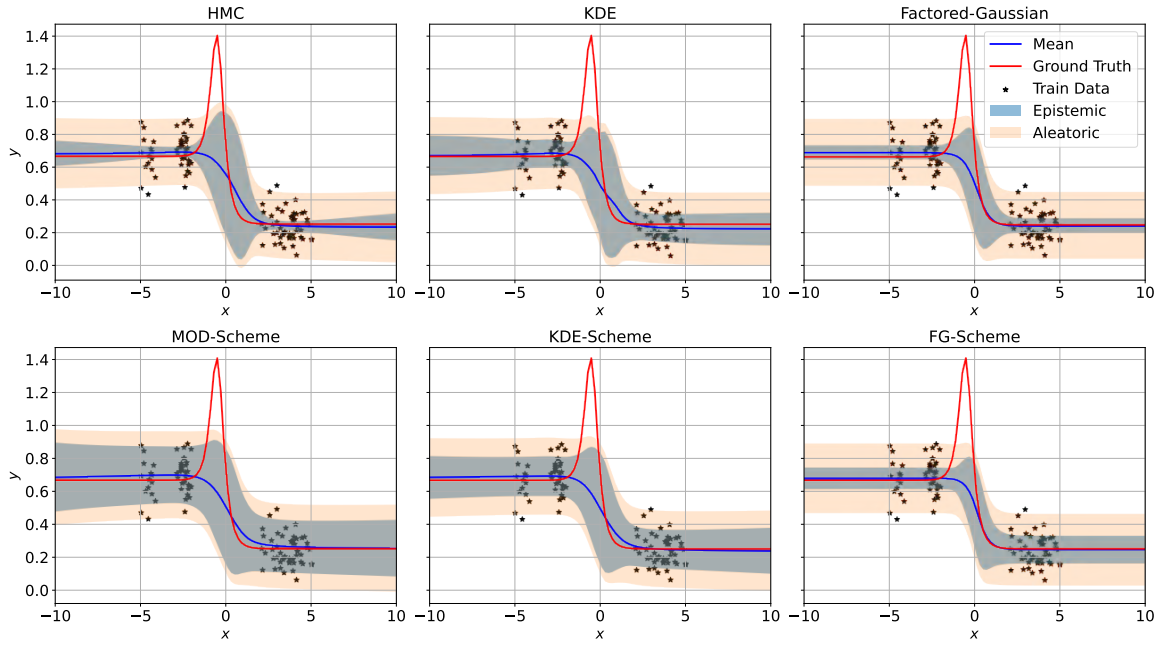


Fig. 4.9 Plot of the predictive distributions for samples using all the different compression schemes with $\beta = 1$. The first row shows the predictive distribution for the true samples, with the second row being the predictive distribution using samples compressed with the `MOD-Scheme`, `KDE-Scheme` and `FG-Scheme` respectively. Th reader is encouraged to zoom into these plots to see the predictive distributions in detail.

In Figures 4.12, 4.13 and 4.14 we plot example trajectories with their corresponding target distributions (when we trained with $\beta = 1$). These posterior distributions are more complex and potentially multi-modal, so the factorised Gaussian is an extremely poor approximation. Due to the KL term in the ELBO training objective, regions where q/p is large are heavily penalised, and so our variational posterior fits to a single mode of the true posterior. This means alternative explanations of the data are completely ignored, and therefore the model under-represents uncertainty in its predictions. This helps to explain the confident predictive distributions we saw in Figure 4.9 and suggests the test-set performance should be poorer.

By varying the β in our training objective we can analyse our compressed models' performances against their compression size with results shown in Figure 4.10. On the test-set the performance of all the methods improves as the compression size increases. In the Bayesian Linear Regression experiment, varying ε changed the compression size, but our samples were still targeting the same distributions. In this case, varying β instead results in different target distributions, so the mode bias issues are less prevalent since we keep ε fixed to a reasonable value of 0.2. This also shows that if we care about having good uncertainty calibration in our predictions we should prefer to use more nats through the β training objectives rather than simply increasing the overhead with ε.

Interestingly, the training set performance of the `FG-Scheme` is greater than the other methods for every compression size. This contrasts the linear regression results where the `MOD-Scheme` model scored better on both. The reason for this is that the uncompressed HMC samples perform worse than the factorised Gaussian samples on the training set, shown in Figure 4.11. We also find, similarly to the previous experiment, that the compressed samples with the `FG-Scheme` outperform uncompressed samples from the factorised Gaussian on the test-set - adding noise increases the model's uncertainty and so it over-fits less. Additionally, these results confirm that the `KDE-Scheme` is adding value compared to the simpler `MOD-Scheme`. Both results show an almost identical pattern, with asymptotically improving performance when using more nats, but the `KDE` scheme achieves better log-likelihoods at every tested value. Referring to the predictive distributions in Figure 4.9, the `KDE-Scheme` has a tighter fit where data is present but maintains the uncertainty in extrapolation and 'in-between' values. Figure 4.11 does indicate performance drop-off when using true samples from the KDE compared to the samples from HMC, suggesting that better performance could be achieved by using a more refined approximation.

Overall, these results demonstrate that as the true posterior distribution becomes more complex, the factorised Gaussian approximation struggles, and methods that target exact samples from $p(\mathbf{w} \mid \mathscr{D})$, like the `MOD-Scheme` and the `KDE-Scheme`, become more necessary.

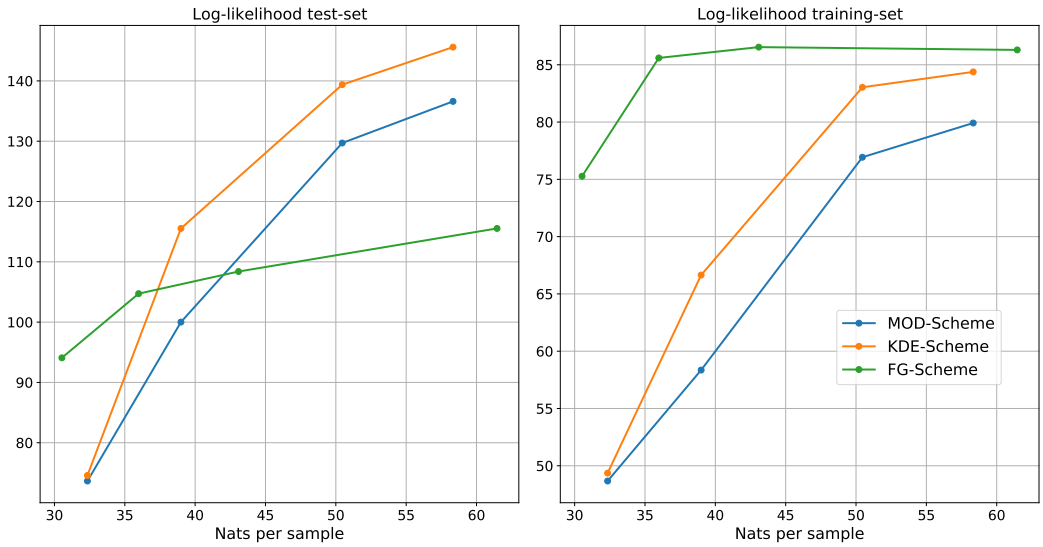


Fig. 4.10 Plot of the regression results for compressed models using `MOD-Scheme`, `KDE-Scheme` and `FG-Scheme`. As we weaken the compression the model performance increases for all schemes on both the training and test-sets.

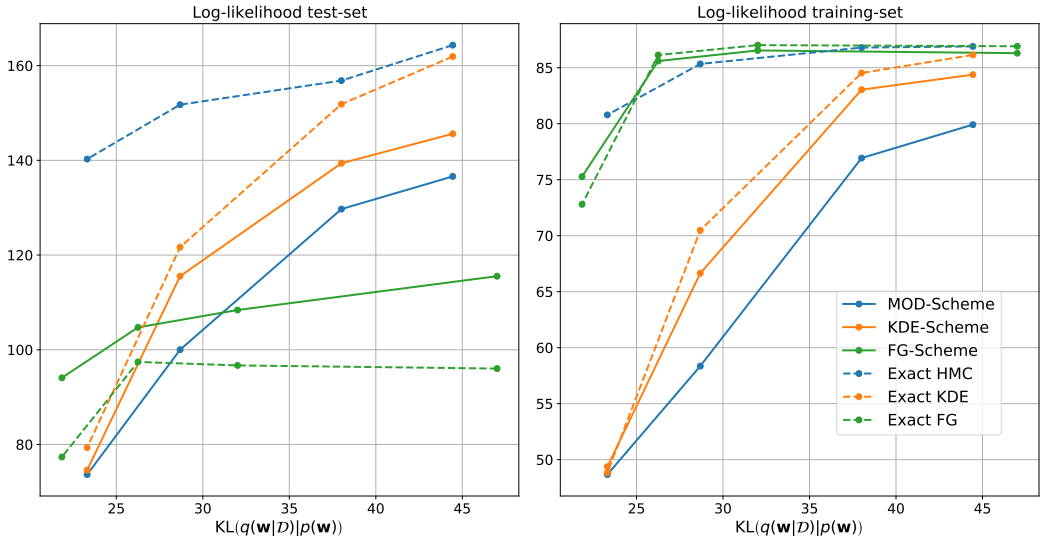


Fig. 4.11 Plot of the regression results for compressed models using `Empirical-Scheme`, `KDE-Scheme` and `FG-Scheme` compared to using uncompressed samples. Here we plot against the relative entropy using each method. This shows that as we decrease the value of β in our training objective, the relative entropy is able to increase and as a result the training set fit improves.

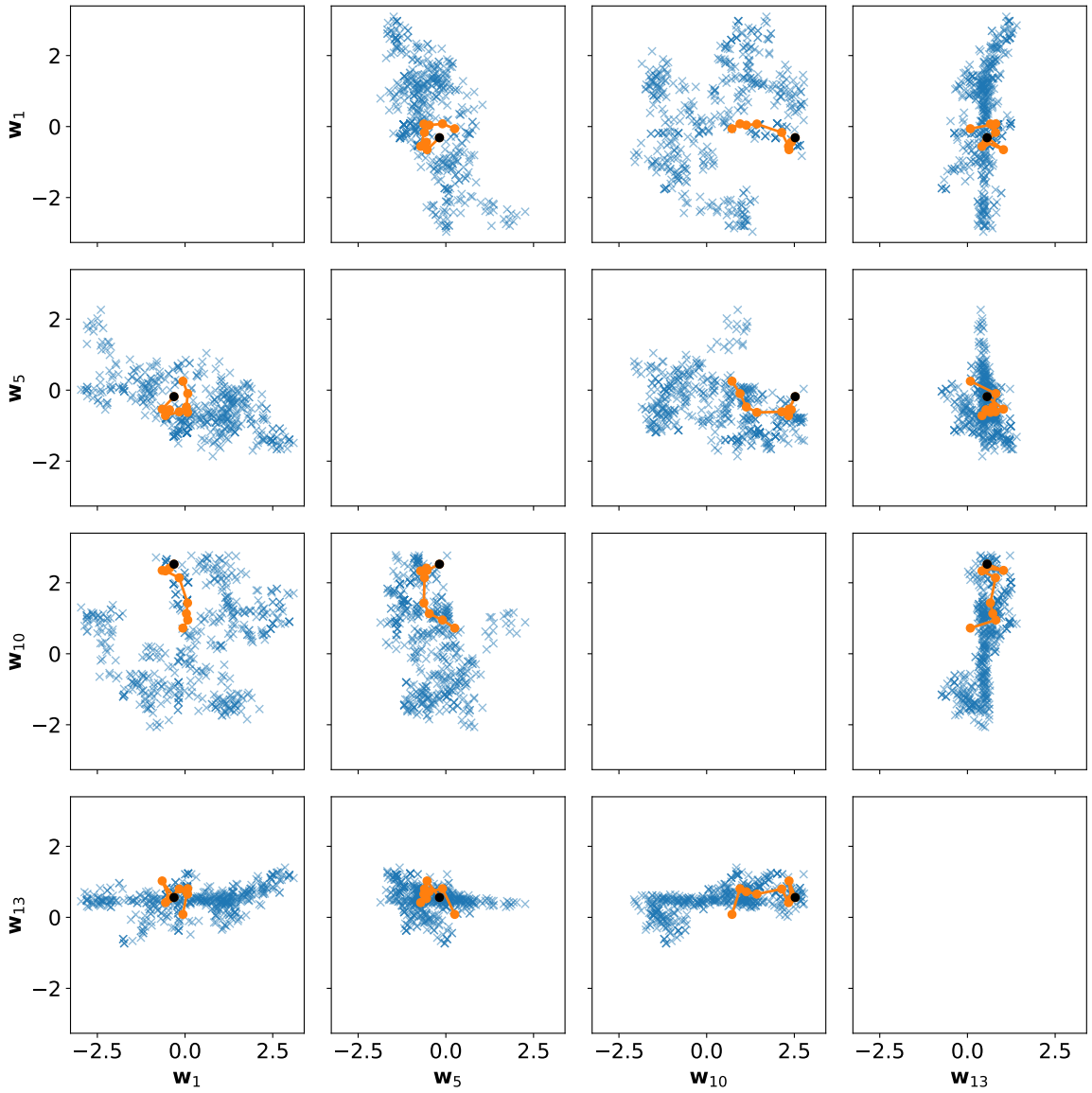


Fig. 4.12 Plot of the trajectory of `MOD-Scheme` on toy regression problem using BNNs. Here we plot a subset of the weights. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.

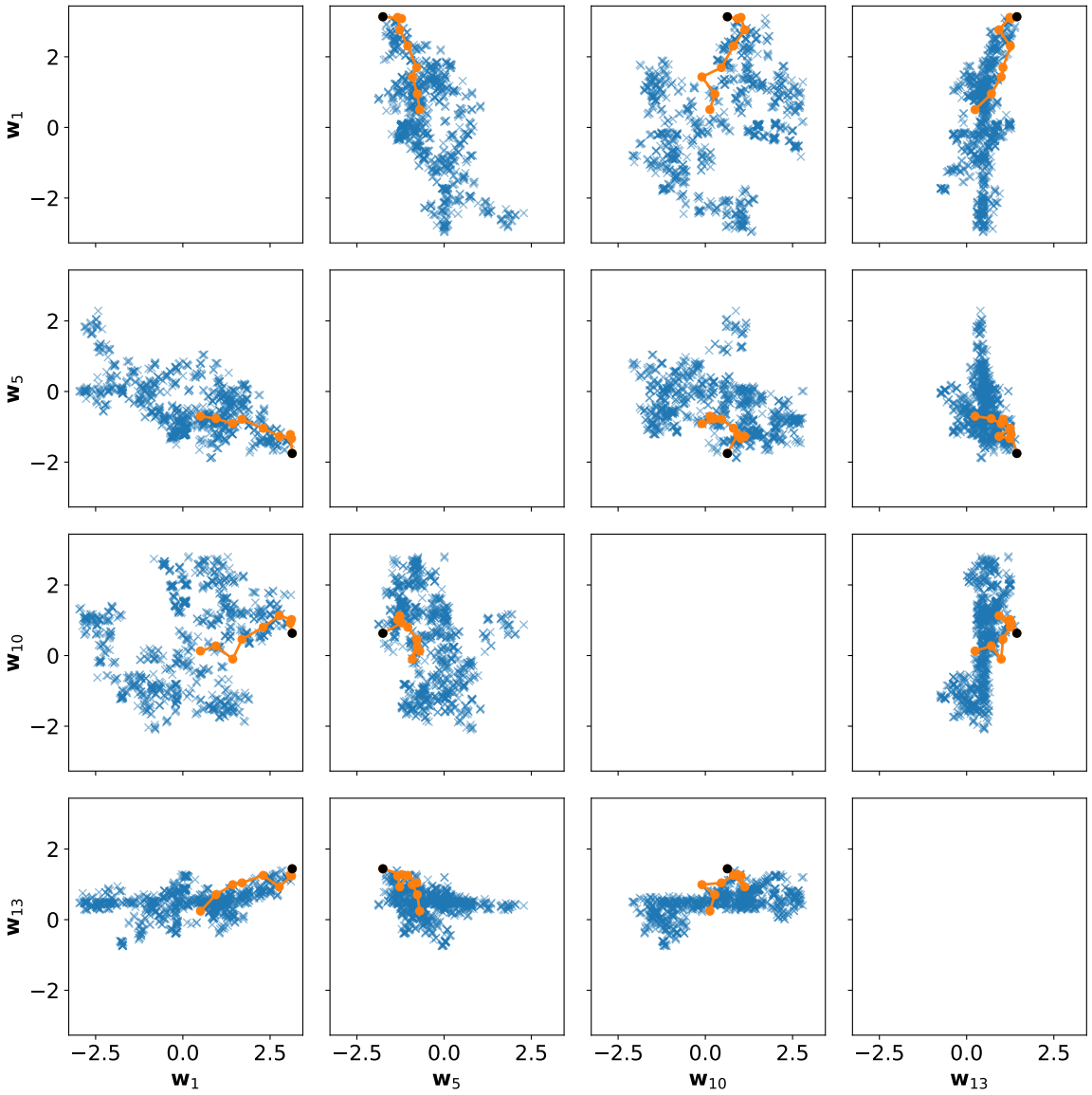


Fig. 4.13 Plot of the trajectory of `KDE-Scheme` on toy regression problem using BNNs. Here we plot a subset of the weights. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.

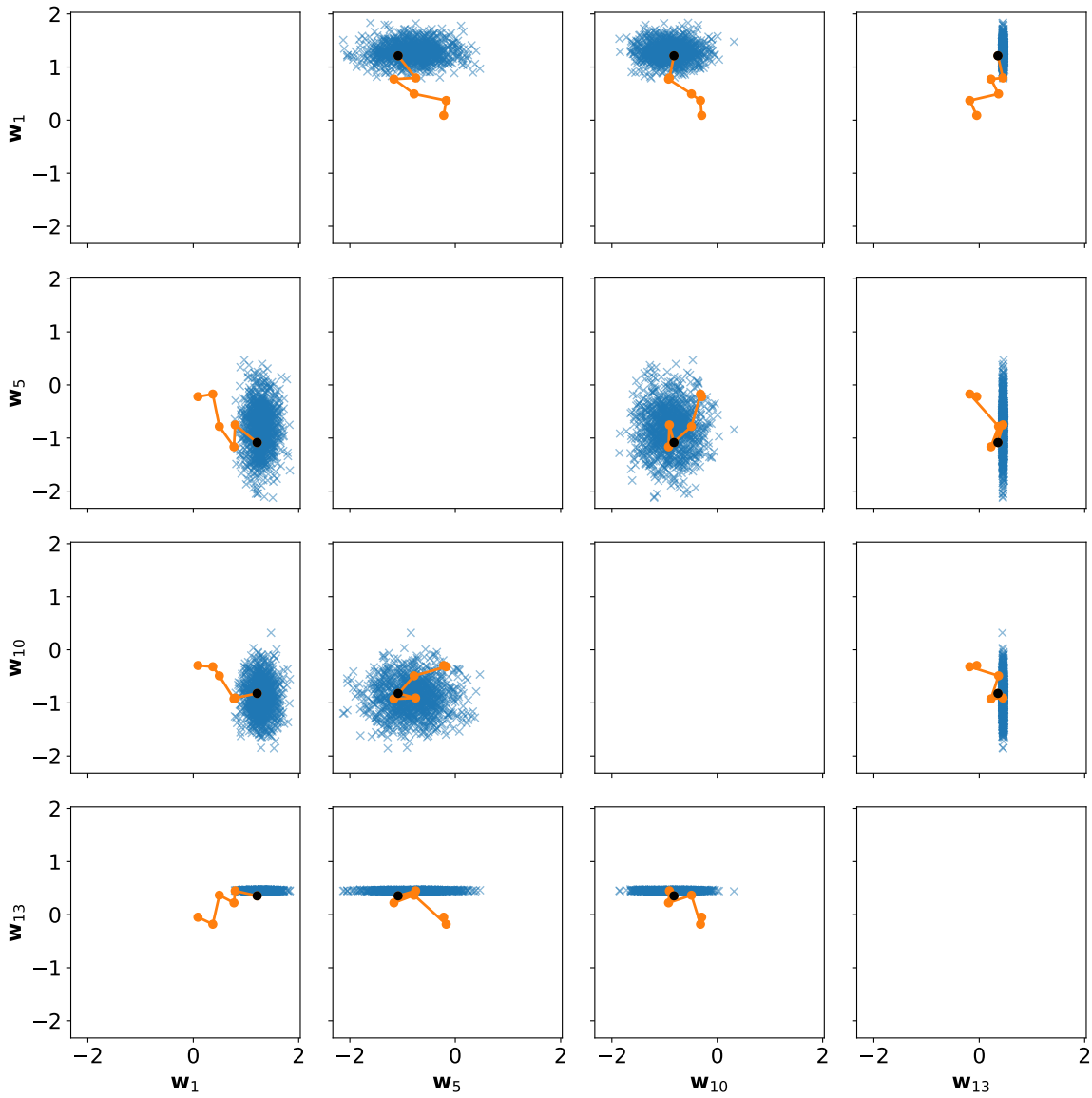


Fig. 4.14 Plot of the trajectory of `FG-Scheme` on toy regression problem using BNNs. Here we plot a subset of the weights. Compared to the other trajectory plots in Figure 4.12 and Figure 4.13, the target is much smaller and has less variance. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample.

4.5 BNN Regression - UCI Energy

This section analyses the performance of compressed samples from BNNs which model regression on the UCI Energy data-set [Tsanas and Xifara, 2012].

4.5.1 Modelling the noise

For this data-set, unlike the previous ones, we do not know a priori the observation noise term σ^2. One approach would be to model the noise in a heteroscedastic fashion, i.e. make our model output both a mean and variance conditional on the input. Another method is to learn homoscedastic noise; a fixed noise σ that is assumed to be equal for all data points. Since the first method amounts to simply adding a node to the output layer, we instead learn homoscedastic noise σ and model it in a way that we can compress it along with the model weights. Specifically, we consider the parameter $\rho \sim p(\rho) = \mathcal{N}(\rho \mid 0, 1)$ and compute

$$\sigma = \text{softplus}(\rho) = \log(1 + \exp(\rho)). \tag{4.20}$$

This function constrains the noise to be positive and is depicted in Figure 4.15. We first trained MAP estimates on this data-set to find a suitable range of values for the homoscedastic noise. These yielded $\sigma \in [1, 3]$ and so our prior over ρ is flexible enough to yield posteriors on ρ with significant probability mass in these ranges.

4.5.2 Data-set and Architecture

We create 'gaps' in the data-set akin to Foong et al. [2019]. For each dimension of the inputs $\{\mathbf{x}_n\}_{n=1}^N$, we create a training/test split whereby we sort the points in ascending order in the specific dimension, and then remove the middle third of the points as test points. The remaining two-thirds are the training points and so our training set has a 'gap' which allows us to observe the model's handling of 'in-between' uncertainty. In this way, for a D dimensional input $\mathbf{x}$, we create D training/test splits and perform the regression task on each split.

Again, we implement a single hidden layer BNN with 3 hidden units and tanh activation functions. We place isotropic Gaussian priors over the weights $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}_N, \mathbb{I}_N)$, meaning we can consider the set of parameters $\mathbf{z} = \{\mathbf{w}, \rho\}$ with prior distribution $p(\mathbf{z}) = p(\mathbf{w}, \rho) = \mathcal{N}(\mathbf{0}_{N+1}, \mathbb{I}_{N+1})$ where $\mathbf{w} \in \mathbb{R}^N$. To get uncertainty over our predictions we adapt

the methodology given in Section 4.1 by setting

$$g(\mathbf{y}_t) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{N}(\mathbf{y}_t \mid \mathscr{M}[\mathbf{w}_i](\mathbf{x}_t), \sigma_i^2), \tag{4.21}$$

where $\sigma_i^2 = \text{softplus}(\rho_i)$. For the VI posterior over the parameters $\mathbf{z}$, we will again assume a factorised Gaussian, so we can compress samples using the `FG-Scheme`. Also, we run HMC to target $p(\mathbf{z} \mid \mathscr{D})$. For each split we then create the KDE using sub-samples from the HMC run to approximate the relative entropy between the true posterior and the prior. Then we can compress samples using the `MOD-Scheme`, and the `KDE-Scheme` which directly targets the KDE. Unlike the previous test, we will not use a β training loss. By learning the likelihood noise, and noting that Section 2.2.4 made an equivalence between augmenting the likelihood noise and training with a β objective, we find that changing β would simply result in the posterior on ρ adapting to offset the changes. We can still compare the performance when using $\beta = 1$ and in this case we would expect the models compressed using exact samples to perform better in terms of uncertainty calibration and compression size compared to those targeting the variational approximation.

To compress the samples, we use the same hyper-parameters as for the toy BNN problem, namely $\Omega = 5$, $\varepsilon = 0.2$ and optimised variances. Again, we chose not to implement beam-search for the computational issues discussed previously.

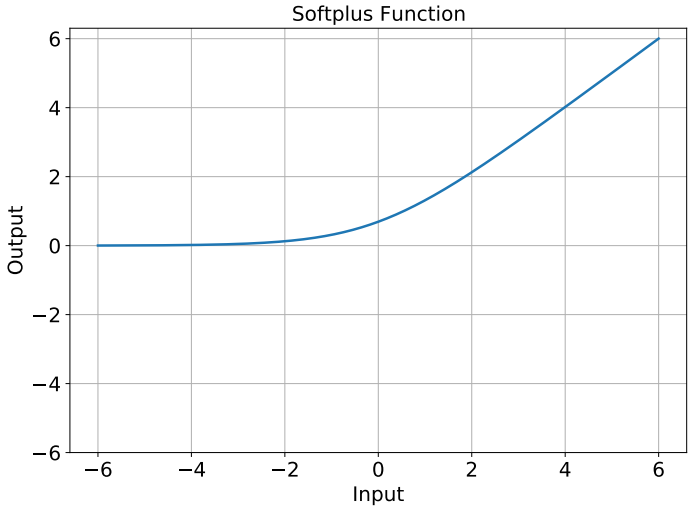


Fig. 4.15 Plot of the softplus activation function. This is what we use to transform the parameter ρ into the homoscedastic noise term σ.

4.5.3 Results

We compress 500 samples for each split and average the results across each split, reporting standard errors in Figure 4.16. Here we find that the `MOD-Scheme` and `KDE-Scheme` significantly outperform the `FG-Scheme` on both test-set log-likelihood and compression efficiency. The recurring theme in our results is that by targeting the true posterior over the parameters $p(\mathbf{z} \mid \mathscr{D})$, we can achieve better uncertainty calibration which results in better data-fit on unseen data-sets. Furthermore, in this test, the compression efficiency for the `MOD-Scheme` and `KDE-Scheme` was better than the `FG-Scheme`.

Our data-fit results show similar trends to those found in Foong et al. [2019], where the variational approximation suffers greatly on the test-set. Interestingly, our performance is better for the factorised Gaussian samples compared to Foong et al. [2019]. Immer et al. [2021] have noted that Mean-Field Variational Inference models often benefit from using simpler architecture, which may explain this phenomenon. Another explanation could be that our models are inferring a posterior on the observation noise σ^2 whose mean value is larger than the noise learnt in the paper. Foong et al. [2019] learn a maximum likelihood estimate of σ^2 but its value is not given, so it is hard to determine which of these explanations is most likely. These effects are exaggerated when using compressed samples since the compression method introduces noise into the samples, further increasing the epistemic uncertainty in predictions. This links to the results found in Section 4.3, where the compressed samples from the `FG-Scheme` can often outperform true samples if the variational approximation has underestimated the uncertainty.

It's clear that the `FG-Scheme` samples are overfitting by examining the training-set log-likelihood, where it far outperforms the other methods, consistent with results from our other experiments. Furthermore, the standard error bars in Figure 4.16 are much larger for the `FG-Scheme` samples since their performance was very different depending on each of the training/test splits, which is consistent with the results in Foong et al. [2019]. For some of the splits, overfitting wasn't an issue. For those specific dimensions, the data in the missing 'gap' may not vary drastically from the trend in the observed training data. The difference in the splits also increased the variance of the compression size; on the data-sets where the variational approximation overfits, the relative entropy can grow much larger since the data-fit term in the ELBO can be trained to be very small. As a result, more nats are required to compress these models which is reflected in the variance of the compression size. This shows that using true samples also makes the compression size more robust to augmentations in the data, and the final models are less sensitive to the training methods. Finally, these

results reveal that, by placing a prior over the learnt noise parameter ρ, we can compress information relevant to our model that is different to the model weights.

Overall, we have seen that our methods extend to models trained on real-life data-sets. In accordance with the link between the MDL principle and the ELBO, using exact samples from the true posterior both improves test-set performance and compression size.

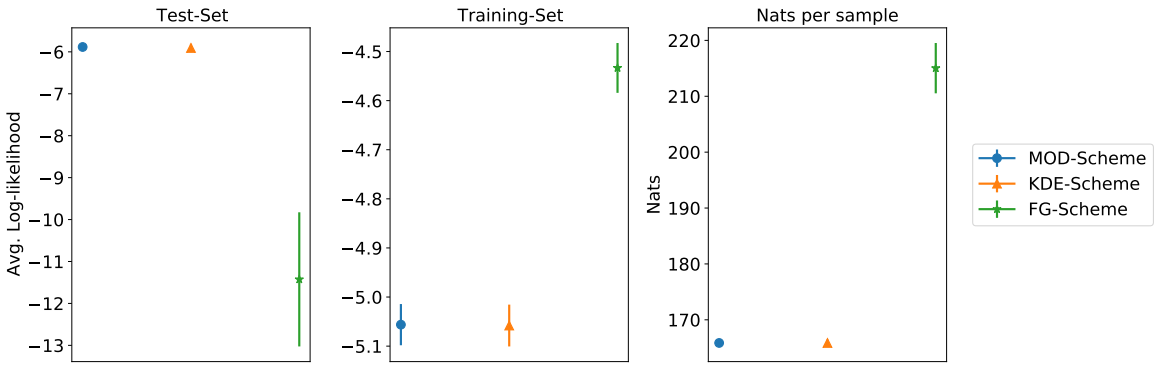


Fig. 4.16 Plot of training and test-set log-likelihood along with expected coding efficiency per ensemble member. Here we see that the samples compressed from the variational approximation have much worse performance on the test-set and require many more nats to encode.

Chapter 5

Related and Future Works

In this section, we begin by reviewing some common methods for model compression in machine learning literature. We detail these approaches and contrast them with ours. Afterwards, we draw upon these ideas for inspiration to improve our method. We propose two novel methods to improve iREC for model compression: (1) specific coding goals, and (2) dynamical proposal distributions.

5.1 Related Works

In this thesis, we presented a method for model compression that corresponded to transmitting a sample from a distribution over the model's weights. Our method sends a random sample from the weight posterior rather than a deterministic set of weights. This makes it well-suited for the task of sending multiple samples from the posterior and performing inference, thus calibrating the uncertainty in our predictions. The model compression literature, however, is focused on transmitting a single setting of the weights, and so differs from our work. Since weight pruning and quantization are the most prominent, we focus on these here.

5.1.1 Weight Pruning and Quantization

By identifying redundancies and removing them, data can be compressed with minimal distortion. This idea can apply to models, particularly neural networks since they are known to be over-parameterised in many applications [Alvarez and Salzmann, 2017]. If specific weights are not useful for making predictions, then removing them will reduce the footprint of the model without significant impact on its performance.

This idea has been used to reduce the size of neural networks for many decades, with the earliest being Biased Weight Decay from Hanson and Pratt [1988]. By appealing to Occam's Razor, they aimed to reduce the number of parameters in their neural networks to make them generalise better. To do this, they added weight decay terms to their loss function and removed weights that made minimal impact on the performance. Whilst their method was not originally meant for compression, by reducing the number of weights, fewer parameters need to be communicated to the receiver. Later, in *Optimal Brain Damage*, LeCun et al. [1990] used the Hessian matrix to determine which weights to prune and showed better performance than in Biased Weight Decay.

Deep Compression proposed by Han et al. [2015] used these ideas directly for compression. They performed magnitude-based weight pruning by taking the weights of the model whose values were below a threshold and setting them to zero. Then they systematically retrained to account for the pruning. To further reduce the size of the networks, they used k-means clustering to conduct weight-sharing. For a hidden layer with n connections, using b bits to represent each weight value and quantizing the weights into k clusters results in $kb + n\log_2(k)$ bits to transmit per layer, in contrast to nb bits without the weight-sharing. The kb term refers to the number of bits used for the weight value of each cluster, and the $n\log(k)$ is the number of bits needed to determine which cluster each weight belongs to. In this way, quantization happens on two levels: (1) reduction in precision in the weights through b and (2) clustering to reduce the total unique weights in the network. Afterwards, the final quantized weights are encoded with Huffman coding.

Links to our method

Weight pruning involves manually removing the weights that don't impact the performance of the model. With iREC, since we place a prior over the weights, any weight that has no impact on the performance will have a posterior that collapses onto the prior. Consequently, these redundant weights do not contribute to the relative entropy of the full posterior and so cause no increase in the codelength. This shows that our method effectively weight prunes without us needing to explicitly remove the weights from the model[1], and using the β training objective allows us to control the degree of this. On the other hand, weight-sharing could be useful as it would reduce the dimension of the target and prior distributions which should reduce the relative entropy leading to more efficient encodings, but also reduce the runtimes of our compression algorithms - as we have shown in Section 3.6.

[1] This means we don't impact the overall architecture compared to the uncompressed model.

5.1.2 Probabilistic Methods

The most notable probabilistic method of model compression is *Bayesian Compression* by Louizos et al. [2017]. It takes inspiration from the equivalence between Bayesian model comparison and the Minimum Description Length Principle to help guide weight pruning. Rather than pruning at the weight level, they prune entire hidden units by using the ELBO with a specific sparsity inducing prior as their training objective. Furthermore, they look at the variance in their variational posterior to determine what bit precision to use for encoding each weight parameter. Weights with high posterior variances are thought to be less informative and given reduced precision. Whilst probabilistic methods are used to guide pruning and quantization, this method still yields a deterministic setting for the weights that is then encoded using the standard methods. Overall, this method uses the connection made in 'Bits-back' argument [Hinton and Van Camp, 1993] to help compress the size of the neural network, but doesn't aim for the postulated coding efficiency of KL $\left[q_\phi \| p\right]$. A method that does aim for the 'optimal' coding efficiency is MIRACLE [Havasi et al., 2018].

MIRACLE

The most closely related method to iREC is MIRACLE [Havasi et al., 2018]. This method was the first to introduce the importance sampling procedure that we described in Chapter 3. MIRACLE is a similar but more rudimentary compression method, where the model parameters are split up into chunks and each chunk is compressed using a similar method to iREC. A nice feature of the MIRACLE method is that a specific coding goal of C nats can be set, and an adaptive β-ELBO training objective is used to ensure the coding goal is achieved.

Explicitly, a *global coding goal* of C nats is set, and the weight vector $\mathbf{w}$ is split into blocks, where the number of blocks is determined by $K = \left\lceil \frac{C}{C_{loc}} \right\rceil$, where C_{loc} is the *local coding goal*. The parameter C_{loc} should remind the reader of the parameter Ω from the iREC scheme since it determines the number of blocks (auxiliary variables) the scheme will utilise. Each block is then trained with a β-ELBO objective that is specific to the block. In this way, if the relative entropy of a specific block exceeds its constraints C_{loc}, the value of β for that block is increased to encourage the posterior on the block to lie closer to the prior - reducing the cost of compressing a sample. Analogously, if the relative entropy is too small, the value of β for that block can be relaxed to allow the posterior to differ more from the prior. By uniformly splitting the weights into blocks, only the number of blocks needs to be communicated which can easily be done using the shared random number generator.

In practice, to compress a sample, a prior is set over the weights and the variational parameters of the approximate posterior $q(\mathbf{w})$ are trained using a β-ELBO until convergence. Then, one

of the K blocks is randomly sampled and encoded like in the iREC scheme with the index importance sampling scheme. Subsequently, the method involves updating the variational distribution over the blocks which have not yet been encoded to ensure the global coding goal is maintained. The model is conditioned on the compressed block and the remaining blocks are retrained using the block-specific β-ELBO to both ensure the global coding goal is met and help the model adjust to the compressed sample. Intuitively, conditioning on the already compressed samples allows for the network to 'learn' how to account for poor samples. Finally, they employ a more rudimentary weight-sharing method than that seen in *Deep Compression*. Akin to [Chen et al., 2015], they use a random hashing function to tie weights together pre-training, resulting in lower-dimensional distributions $p(\mathbf{w})$ and $q(\mathbf{w})$. This subsequently reduces the per block relative entropy - improving the compression rate. This method achieves state-of-the-art performance on typical network compression benchmarks and is bits-back efficient.

One can view MIRACLE as a special case of our method where the auxiliary variables are the blocks and the function f such that $\mathbf{z} = f(\mathbf{a}_{1:K})$ is concatenation. One drawback of MIRACLE compared to our method is that the receiver needs to know how to piece back the blocks to recreate the sample. In contrast, for our method, the receiver simply sums the decoded auxiliary variable samples to recreate the intended weight sample. There are a several features of MIRACLE that could be useful to improve our method (e.g. explicit coding goals) and we discuss these later.

5.2 Future Works

Drawing on inspiration from the related works, here we detail some ways in which we could extend/improve our model compression method. The main issues with our method currently are the compression speed and the inability to explicitly control the compression size. Here we detail two possible extensions to tackle both.

5.2.1 Specific Coding Goals

Whilst we can use the β training objective to implicitly set the compression size, it is not able to set a specific coding goal like in MIRACLE. It is possible to train models with a variety of β values and choose the one that gives the relative entropy closest to the intended compression size, but this is a tedious and inelegant solution. Rather than break up posterior into chunks and enforce local coding goals as in MIRACLE, we propose an alternative method that has minimal impact on the current iREC workflow. Instead, we rephrase the

β-ELBO in the following way:

$$\text{maximise} \quad \mathbb{E}_{q_\phi(\mathbf{w})}\left[\log p(\mathscr{D} \mid \mathbf{w})\right] \tag{5.1}$$

$$\text{subject to} \quad \mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right] \leq C. \tag{5.2}$$

This objective would yield the explicit coding threshold of C nats. To make this a trainable objective we need to rewrite it as a differentiable function on an unconstrained domain. To do this we can use a logarithmic barrier function:

$$\text{maximise} \quad \mathbb{E}_{q_\phi(\mathbf{w})}\left[\log p(\mathscr{D} \mid \mathbf{w})\right] - \log\left(C - \mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right]\right). \tag{5.3}$$

This objective enforces the constraint that $\mathrm{KL}\left[q_\phi(\mathbf{w}) \| p(\mathbf{w})\right] \leq C$ and is differentiable. This objective would be interesting to explore in future works to see if we are able to explicitly set coding thresholds for our compression algorithms.

5.2.2 Dynamic Proposal Distributions

In Hinton and Van Camp [1993] the optimal compression size of encoding a sample from distribution q using distribution p is shown to be $\mathrm{KL}[q(\mathbf{z}) \| p(\mathbf{z})]$. Our work focused on improving the approximation to the true posterior, q, while using the prior p as the proposal distribution. It's clear that improving the proposal distribution will also have a big impact and so should be studied in more detail. In the case of sending multiple samples, each decoded sample gives information to the decoder about q. Therefore they should update their proposal distribution p accordingly. We illustrate the importance of this with a simple toy example. From here on we define the first proposal distribution p_0 to initially coincide with the prior used to train the model p. After decoding the samples, we can then make an update to the proposal by performing a Bayesian update on the proposal parameters. In the case of Gaussian targets and proposals, this update is exact and so we detail an example of this below. In this example, we only update the mean as it is a simply illustrative example, but for more complicated problems, the covariance matrix could similarly be updated.

Consider compressing a 1D Gaussian target with a 1D Gaussian proposal. Specifically, set $q(z) = \mathscr{N}(z \mid \nu, \rho)$ and consider a proposal with a prior over its mean, that is $p(z \mid \mu) = \mathscr{N}(z \mid \mu, \sigma^2)$ with $p(\mu) = \mathscr{N}(\mu \mid \mu_0, \sigma_0^2)$. This means that the sender and receiver agree to the distributions $p(z \mid \mu)$ and $p(\mu)$ beforehand. Marginalising over the mean we get the

initial proposal distribution[2],

$$p_0(z) = \mathcal{N}(z \mid \mu_0, \sigma^2 + \sigma_0^2). \tag{5.4}$$

Now we can perform iREC with $q(z)$ and $p_0(z)$. Suppose the random sample $\hat{z}$ is transmitted using iREC, then both the sender and receiver can perform the Bayesian update on the mean as follows:

$$p(\mu \mid \hat{z}) = \mathcal{N}\left(\mu \mid \mu_1, \sigma_1^2\right), \tag{5.5}$$

where

$$\sigma_1^2 = \left(\frac{1}{\sigma^2} + \frac{1}{\sigma_0^2}\right)^{-1}, \tag{5.6}$$

$$\mu_1 = \frac{1}{\sigma^2 + \sigma_0^2}\left[\sigma_0^2 \hat{z} + \sigma^2 \mu_0\right]. \tag{5.7}$$

We can now marginalise over μ with our posterior to compute the new proposal,

$$p_1(z) = \mathcal{N}(z \mid \mu_1, \sigma^2 + \sigma_1^2). \tag{5.8}$$

Since this update is exact, there is no information needed to be transmitted for the decoder to perform the update. Consequently, the new proposal can be used with the iREC scheme and this process of updating the proposal can continue. We implement this method on a 1D compression task. We find that the compression size (derived from the relative entropy) falls rapidly (see Figure 5.2) and converges as the proposals adapt to have their mean coincide with the mean of the target (see Figure 5.1). Altogether, the average codelength of the samples decreases with the number of samples sent, which results in a much smaller compression size for the full set of compressed samples. Since the compression size reduces, the relative entropy between the target and proposal is smaller, and so fewer samples need to be drawn at each step, this reduces the compute needed to compress each additional sample. Furthermore, the transmitted samples are far closer to the true distribution compared to the samples compressed with standard iREC. Overall, this demonstrates that the dynamic proposals improve the compression size, sample quality and computational costs of performing iREC to compress multiple samples. In particular, this method works well with factorised

[2]In general we would set σ^2 and σ_0^2 such that they initially coincide with the prior over z, this way we can fairly compare this new method to the standard iREC scheme.

Gaussian posteriors since the Bayesian updates are exact. We leave further exploration and experimentation to future work.

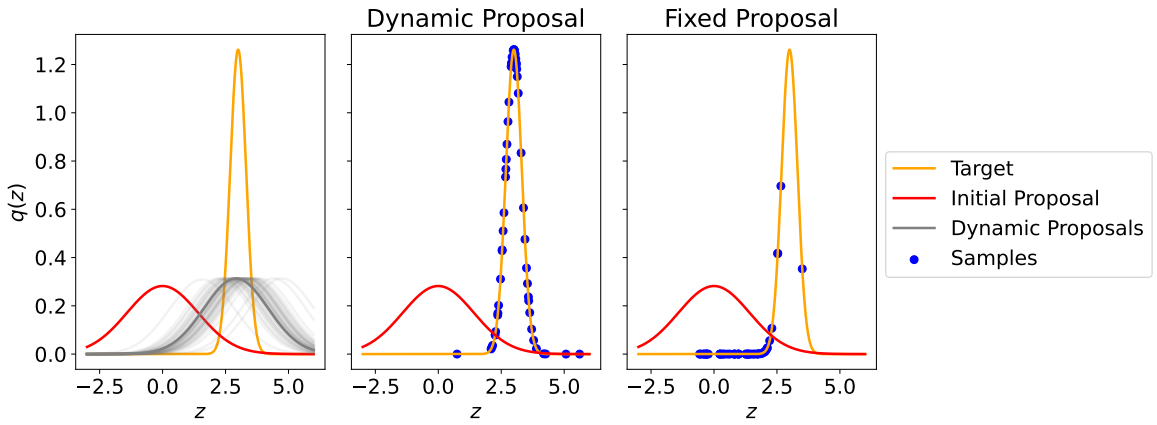


Fig. 5.1 The first plot shows how the proposal distribution changes over time, the mean eventually coincides with the target mean. The second and third plots show the difference between the compressed samples with the dynamic proposal and a fixed proposal. The dynamic proposal samples are far less biased than the fixed proposal.

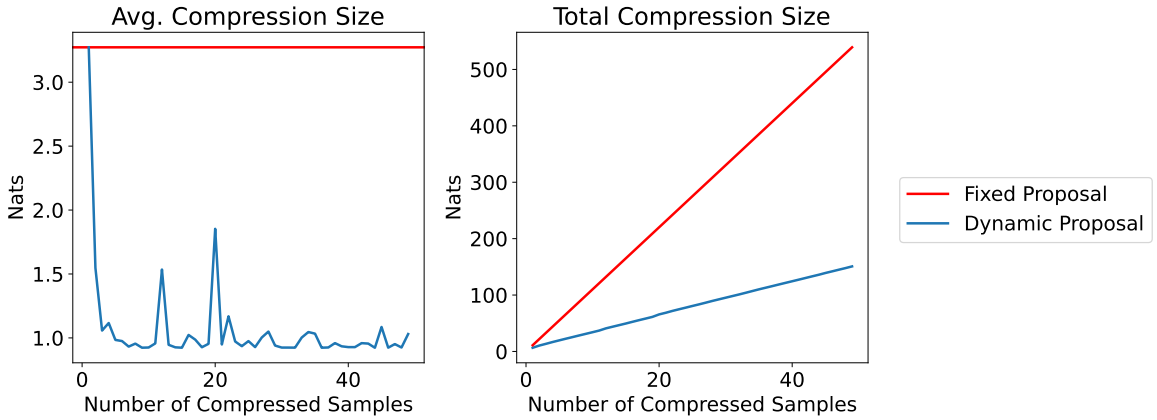


Fig. 5.2 The first plot shows that the average compression size reduces with each additional sample since the proposal can change to better suit the target distribution, this directly relates to a smaller compression size. This plot has fluctuations since the selected sample may not always lie on the target mode. The second plot shows how the total size of all the compressed samples grows with a fixed proposal and a dynamic proposal. The dynamic proposal is much cheaper. The fluctuations in the average compression size have minimal impact on the overage compression size and so the plot appears mostly smooth.

Chapter 6

Conclusions

This thesis began with an introduction to source coding and probabilistic modelling. We showed that there is a link between Bayesian model selection and the Minimum Description Length principle. This motivated us to investigate improving upon the factorised Gaussian variational approximation that was used in previous work from Flamich et al. [2021] and Havasi et al. [2018].

In Chapter 3, we described the iREC compression method and gave a detailed analysis of its hyper-parameters and limitations. We presented 2 novel schemes, namely the `MOD-Scheme` and `KDE-Scheme`, which utilise samples from the true posterior.

In Chapter 4, we tested the compression methods on the task of sending multiple samples from a posterior of a model. We demonstrated that targeting exact samples results in better compression size and uncertainty calibration. We also demonstrated that using a β training objective allows one to trade-off model fit and compressibility.

In Chapter 5, we gave an overview of current model compression methods in the literature and contrasted these to our method. We then drew inspiration from these to suggest some ways in which we can improve our method; specifically introducing an explicit coding threshold and using dynamic proposal distributions.

This thesis focused on improving model compression by using approximations that target exact samples from the posterior over weights, rather than a variational approximation. This idea can be applied to compressing samples from any posterior. In particular, we believe that applying these ideas to the task of image compression, as conducted in Flamich et al. [2021], should see significant benefits in both image quality for lossy image compression, and reduced codelength. Furthermore, our methods provide a way to compress high quality

samples from any generative model, and so it has more general applications outside of those discussed in this thesis. We simply need to model the data such that a random sample from a posterior distribution corresponds to a sample from the data distribution. As latent variable models become more sophisticated, more complex data will be able to be compressed with REC algorithms.

On a final note, the biggest barrier to applying the compression methods shown in this thesis to real world problems is the computational cost incurred to compress a sample. As the number of parameters belonging to the model grows, the speed of the compression algorithm reduces considerably, and the methods needed to reduce the bias, like beam-search, exacerbate this. Our method struggles from needing to keep track of many high dimensional tensors which causes memory allocation issues. Furthermore, since we sample the auxiliary variables in sequence, is is not trivial to parallelize the compression method. Both the dynamic proposal and explicit coding threshold ideas from Section 5.2 could help by reducing the relative entropy, and therefore the number of samples drawn at each step of the algorithm, but even so, storing multiple very high dimensional tensors in a computer's memory is impractical. One way to tackle this would be to split up the parameter space into chunks, akin to MIRACLE [Havasi et al., 2018], to spread the computational cost across each chunk. Whilst this would complicate the iREC algorithm, it may be necessary to apply the method in wider use cases. We believe shifting the focus of future research towards making better proposals p and reducing the computational requirements is key to make these compression methods practical in the real world.

References

Alvarez, J. M. and Salzmann, M. (2017). Compression-aware training of deep networks. *Advances in neural information processing systems*, 30:856–867.

Barron, A., Rissanen, J., and Yu, B. (1998). The minimum description length principle in coding and modeling. *IEEE transactions on information theory*, 44(6):2743–2760.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2018). Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chatterjee, S. and Diaconis, P. (2017). The sample size required in importance sampling. *arXiv preprint arXiv:1511.01437*.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. (2015). Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*.

Cover, T. M. (1999). *Elements of information theory*. John Wiley & Sons.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE transactions on information theory*, 21(2):194–203.

Flamich, G., Havasi, M., and Hernández-Lobato, J. M. (2021). Compressing images by encoding their latent representations with relative entropy coding. *arXiv preprint arXiv:2010.01185*.

Foong, A. Y. K., Li, Y., Hernández-Lobato, J. M., and Turner, R. E. (2019). 'in-between' uncertainty in bayesian neural networks. *arXiv preprint arXiv:1906.11537*.

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hanson, S. and Pratt, L. (1988). Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1:177–185.

Havasi, M., Peharz, R., and Hernández-Lobato, J. M. (2018). Minimal random code learning: Getting bits back from compressed model parameters. *arXiv preprint arXiv:1810.00440*.

Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Immer, A., Korzepa, M., and Bauer, M. (2021). Improving predictions of bayesian neural nets via local linearization. *arXiv preprint arXiv:2008.08400*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28:2575–2583.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

Li, C. T. and El Gamal, A. (2018). Strong functional representation lemma and applications to coding theorems. *IEEE Transactions on Information Theory*, 64(11):6967–6978.

Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian compression for deep learning. *arXiv preprint arXiv:1705.08665*.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Tsanas, A. and Xifara, A. (2012). Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540.

Appendix A

Proofs and Derivations

A.1 β Objective Equivalences

Consider the β-ELBO training objective as a functional over the distribution $q(\mathbf{w})$,

$$\begin{aligned}\mathscr{L}^{(1)}[q] &= \int q(\mathbf{w}) \log p(\mathscr{D} \mid \mathbf{w}) + \beta \cdot q(\mathbf{w}) \left(\log p(\mathbf{w}) - \log q(\mathbf{w})\right) \, \mathrm{d}\mathbf{w} \\ &= \int q(\mathbf{w}) \log p(\mathscr{D} \mid \mathbf{w}) + \beta q(\mathbf{w}) \log p(\mathbf{w}) - \beta q(\mathbf{w}) \log q(\mathbf{w}) \, \mathrm{d}\mathbf{w}.\end{aligned}$$

We can consider its variation,

$$\begin{aligned}\mathscr{L}^{(1)}[q + \varepsilon\eta] = \int &(q(\mathbf{w}) + \varepsilon\eta(\mathbf{w})) \log p(\mathscr{D} \mid \mathbf{w}) + \beta (q(\mathbf{w}) + \varepsilon\eta(\mathbf{w})) \log p(\mathbf{w}) \\ &- \beta (q(\mathbf{w}) + \varepsilon\eta(\mathbf{w})) \log (q(\mathbf{w}) + \varepsilon\eta(\mathbf{w})) \, \mathrm{d}\mathbf{w}\end{aligned}$$

where $\eta(\mathbf{w})$ is an arbitrary function in the same functional space as $q(\mathbf{w})$ and $\varepsilon \in \mathbb{R}$.

To compute the functional derivative, we first compute the quantity

$$\begin{aligned}\frac{\mathrm{d}\mathscr{L}^{(1)}[q + \varepsilon\eta]}{\mathrm{d}\varepsilon} = \int &\eta(\mathbf{w}) \log p(\mathscr{D} \mid \mathbf{w}) + \beta \eta(\mathbf{w}) \log p(\mathbf{w}) \\ &- \beta \eta(\mathbf{w}) \log (q(\mathbf{w}) + \varepsilon\eta(\mathbf{w})) - \beta \eta(\mathbf{w}) \, \mathrm{d}\mathbf{w}\end{aligned}$$

and consider,

$$\begin{aligned}\frac{\mathrm{d}\mathscr{L}^{(1)}[q+\varepsilon\eta]}{\mathrm{d}\varepsilon}\Big|_{\varepsilon=0} &= \int \eta(\mathbf{w})\log p(\mathscr{D}\mid\mathbf{w}) + \beta\eta(\mathbf{w})\log p(\mathbf{w}) \\ &\qquad - \beta\eta(\mathbf{w})\log(q(\mathbf{w})) - \beta\eta(\mathbf{w})\,\mathrm{d}\mathbf{w} \\ &= \int \left[\log p(\mathscr{D}\mid\mathbf{w}) + \beta\log p(\mathbf{w}) - \beta\log(q(\mathbf{w})) - \beta\right]\eta(\mathbf{w})\,\mathrm{d}\mathbf{w}.\end{aligned}$$

The functional derivative, is defined

$$\frac{\delta\mathscr{L}^{(1)}}{\delta q} = \log p(\mathscr{D}\mid\mathbf{w}) + \beta\log p(\mathbf{w}) - \beta\log(q(\mathbf{w})) - \beta. \tag{A.1}$$

Often one may want to optimise using a β-ELBO, but without decoupling the unnormalised posterior term $p(\mathscr{D}\mid\mathbf{w})p(\mathbf{w})$. We show that exponentiating the likelihood term with $1/\beta$ gives an equivalent training objective. We begin by defining the functional

$$\begin{aligned}\mathscr{L}^{(2)}[q] &= \int q(\mathbf{w})\log p(\mathscr{D}\mid\mathbf{w})^{1/\beta} + q(\mathbf{w})\log p(\mathbf{w}) - q(\mathbf{w})\log q(\mathbf{w})\,\mathrm{d}\mathbf{w} \\ &= \int \frac{1}{\beta}q(\mathbf{w})\log p(\mathscr{D}\mid\mathbf{w}) + q(\mathbf{w})\log p(\mathbf{w}) - q(\mathbf{w})\log q(\mathbf{w})\,\mathrm{d}\mathbf{w}.\end{aligned}$$

Analogously to the before, consider its variation and compute the functional differential

$$\begin{aligned}\frac{\mathrm{d}\mathscr{L}^{(2)}[q+\varepsilon\eta]}{\mathrm{d}\varepsilon} &= \int \frac{1}{\beta}\eta(\mathbf{w})\log p(\mathscr{D}\mid\mathbf{w}) + \eta(\mathbf{w})\log p(\mathbf{w}) \\ &\qquad - \eta(\mathbf{w})\log(q(\mathbf{w}) + \varepsilon\eta(\mathbf{w})) - \eta(\mathbf{w})\,\mathrm{d}\mathbf{w} \\ \frac{\mathrm{d}\mathscr{L}^{(2)}[q+\varepsilon\eta]}{\mathrm{d}\varepsilon}\Big|_{\varepsilon=0} &= \int \left[\frac{1}{\beta}\log p(\mathscr{D}\mid\mathbf{w}) + \log p(\mathbf{w}) - \log(q(\mathbf{w})) - 1\right]\eta(\mathbf{w})\,\mathrm{d}\mathbf{w}.\end{aligned}$$

The functional derivative is given by

$$\begin{aligned}\frac{\delta\mathscr{L}^{(2)}}{\delta q} &= \frac{1}{\beta}\log p(\mathscr{D}\mid\mathbf{w}) + \log p(\mathbf{w}) - \log(q(\mathbf{w})) - 1 \\ &= \frac{1}{\beta}\frac{\delta\mathscr{L}^{(1)}}{\delta q},\end{aligned}$$

so training with the objective function $\mathscr{L}^{(2)}[q]$ is equivalent to training with the conventional β-ELBO.

A.2 Update Scheme Derivations

A.2.1 `MOD-Scheme`

In our thesis, we propose a new scheme which uses samples from the true posterior. We directly draw samples from $\mathbf{z}_1, ..., \mathbf{z}_D \sim p(\mathbf{z} \mid \mathbf{x})$ and approximate it with a mixture of deltas,

$$q(\mathbf{z}) = \frac{1}{D} \sum_{i=1}^{D} \delta_{z_i}(\mathbf{z}). \tag{A.2}$$

To derive the corresponding update equations in Equation (3.8), we first derive $q(\mathbf{z} \mid \mathbf{a}_{1:k})$. Substituting $q(\mathbf{z})$ into Equation (3.6) yields,

$$\begin{aligned} q(\mathbf{a}_{1:k-1}) &= \int q(\mathbf{a}_{1:k-1} \mid \mathbf{z}) \frac{1}{D} \sum_{i=1}^{D} \delta_{\mathbf{z}_i}(\mathbf{z}) \, \mathrm{d}\mathbf{z} \\ &= \int p(\mathbf{a}_{1:k-1} \mid \mathbf{z}) \frac{1}{D} \sum_{i=1}^{D} \delta_{\mathbf{z}_i}(\mathbf{z}) \, \mathrm{d}\mathbf{z} \\ &= \frac{1}{D} \sum_{i=1}^{D} p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i). \end{aligned} \tag{A.3}$$

Using Bayes' rule we can derive the second recursive equation in Equation (3.8),

$$\begin{aligned} q(\mathbf{z} \mid \mathbf{a}_{1:k-1}) &= \frac{q(\mathbf{a}_{1:k-1} \mid \mathbf{z}) q(\mathbf{z})}{q(\mathbf{a}_{1:k-1})} \\ &= \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}) q(\mathbf{z})}{q(\mathbf{a}_{1:k-1})} \\ &= \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}) \frac{1}{D} \sum_{i=1}^{D} \delta_{\mathbf{z}_i}(\mathbf{z})}{\frac{1}{D} \sum_{i=1}^{D} p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)} \\ &= \sum_{i=1}^{D} \delta_{\mathbf{z}_i}(\mathbf{z}) \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)}{\sum_{j=1}^{D} p(\mathbf{a}_{1:k-1} \mid z_j)}. \end{aligned} \tag{A.4}$$

Now we are ready to compute the first recursive equation,

$$\begin{aligned} q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) &= \int p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z}) q(\mathbf{z} \mid \mathbf{a}_{1:k-1}) \, \mathrm{d}\mathbf{z} \\ &= \int p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, z) \sum_{i=1}^{D} \delta_{\mathbf{z}_i}(\mathbf{z}) \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)}{\sum_{j=1}^{D} p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_j)} \, \mathrm{d}\mathbf{z} \\ &= \sum_{i=1}^{D} \frac{p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_i)}{\sum_{j=1}^{D} p(\mathbf{a}_{1:k-1} \mid \mathbf{z}_j)} p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z}_i). \end{aligned} \tag{A.5}$$

Setting $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \sigma^2 \mathbf{I})$ and assuming that $\mathbf{z} = \sum_{k=1}^{K} \mathbf{a}_k$ where $\mathbf{a}_k \sim \mathcal{N}(\mathbf{a}_k \mid \mathbf{0}, \sigma_k^2 \mathbf{I})$, it follows from Equation (3.9) that $q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1})$ is a mixture of Gaussians.

A.2.2 `KDE-Scheme`

We consider a kernel density estimate using Gaussian kernels, this is equivalent to using a mixture of Gaussians with uniform mixing weights and equal isotropic noise σ_{KDE}^2. This results in the approximation,

$$q(\mathbf{z}) = \frac{1}{D} \sum_{d=1}^{D} \mathcal{N}(\mathbf{z}; \mathbf{z}_d, \sigma_{KDE}^2 \mathbf{I}).$$

Here the means $\mathbf{z}_d$ are exact samples from the true posterior gotten using HMC.

Using the approximation we have,

$$\begin{aligned} q(\mathbf{a}_{1:k}) &= \int p(\mathbf{a}_{1:k} \mid \mathbf{z}) q(\mathbf{z}) d\mathbf{z} \\ &= \frac{1}{D} \sum_{d=1}^{D} \int p(\mathbf{a}_{1:k} \mid \mathbf{z}) \mathcal{N}(\mathbf{z}; \mathbf{z}_d, \sigma_{KDE}^2 \mathbf{I}) d\mathbf{z}. \end{aligned}$$

By Bayes' rule,

$$p(\mathbf{a}_{1:k} \mid \mathbf{z}) \propto p(\mathbf{z} \mid \mathbf{a}_{1:k}) p(\mathbf{a}_{1:k})$$

and by noticing that

$$\mathbf{z} = \sum_{i=1}^{k} \mathbf{a}_i + \sum_{i=k+1}^{K} \mathbf{a}_i := \mathbf{b}_k + \sum_{i=k+1}^{K} \mathbf{a}_i$$

where

$$\mathbf{b}_k \sim \mathcal{N}\left(\mathbf{b}_k; \mathbf{0}, \sum_{i=1}^{k} \sigma_k^2 \mathbf{I}\right)$$

and

$$\sum_{i=k+1}^{K} \mathbf{a}_i \sim \mathcal{N}\left(\sum_{i=k+1}^{K} \mathbf{a}_i; \mathbf{0}, \sum_{i=k+1}^{K} \sigma_i^2 \mathbf{I}\right) := \mathcal{N}\left(\sum_{i=k+1}^{K} \mathbf{a}_i; \mathbf{0}, s_k^2 \mathbf{I}\right),$$

it follows that

$$p(\mathbf{z} \mid \mathbf{a}_{1:k}) = \mathcal{N}(\mathbf{z}; \mathbf{b}_k, s_k^2 \mathbf{I}).$$

Putting this all together,

$$\begin{aligned}
q(\mathbf{z} \mid \mathbf{a}_{1:k}) &\propto p(\mathbf{a}_{1:k} \mid \mathbf{z}) q(\mathbf{z}) \\
&\propto p(\mathbf{a}_{1:k}) \mathcal{N}(\mathbf{z}; \mathbf{b}_k, s_k^2 \mathbf{I}) q(\mathbf{z}) \\
&= p(\mathbf{a}_{1:k}) \mathcal{N}(\mathbf{z}; \mathbf{b}_k, s_k^2 \mathbf{I}) \frac{1}{D} \sum_{d=1}^{D} \mathcal{N}(\mathbf{z}; \mathbf{z}_d, \sigma_{KDE}^2 \mathbf{I}) \\
&\propto \sum_{d=1}^{D} \mathcal{N}(\mathbf{z}; \mathbf{b}_k, s_k^2 \mathbf{I}) \mathcal{N}(\mathbf{z}; \mathbf{z}_d, \sigma_{KDE}^2 \mathbf{I}) \\
&= \sum_{d=1}^{D} \lambda_d^{-1(k)} \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)})
\end{aligned}$$

where the last line follows from the product of multivariate Gaussian densities. Here we have that

$$\begin{aligned}
\lambda_d^{-1(k)} &= \mathcal{N}(\mathbf{b}_k; \mathbf{z}_d, (s_k^2 + \sigma_{KDE}^2) \mathbf{I}) \\
\mathbf{C}_d^{(k)} &= \left(\frac{1}{s_k^2} + \frac{1}{\sigma_{KDE}^2}\right)^{-1} \mathbf{I} \\
\mathbf{c}_d^{(k)} &= \mathbf{C}_d^{(k)} \left(\frac{\mathbf{b}_k}{s_k^2} + \frac{\mathbf{z}_d}{\sigma_{KDE}^2}\right).
\end{aligned}$$

To find the normalising constant, note that

$$\int \sum_{d=1}^{D} \lambda_d^{-1(k)} \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}) d\mathbf{z} = \sum_{d=1}^{D} \lambda_d^{-1(k)},$$

and so it follows that

$$q(\mathbf{z} \mid \mathbf{a}_{1:k}) = \sum_{d=1}^{D} \frac{\lambda_d^{-1(k)}}{\sum_{j=1}^{D} \lambda_j^{-1(k)}} \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}). \tag{A.6}$$

To compute the recursive update for the auxiliary variables we can examine the quantity

$$\begin{aligned}
q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) &= \int p(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}, \mathbf{z}) q(\mathbf{z} \mid \mathbf{a}_{1:k-1}) d\mathbf{z} \\
&= \int \mathcal{N}\left(\mathbf{a}_k; (\mathbf{z} - \mathbf{b}_{k-1}) \frac{\sigma_k^2}{s_{k-1}^2}, \frac{s_k^2 \sigma_k^2}{s_{k-1}^2} \mathbf{I}\right) \sum_{d=1}^{D} \frac{\lambda_d^{-1(k)}}{\sum_{j=1}^{D} \lambda_j^{-1(k)}} \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}) d\mathbf{z} \\
&= \sum_{d=1}^{D} \frac{\lambda_d^{-1(k)}}{\sum_{j=1}^{D} \lambda_j^{-1(k)}} \int \mathcal{N}\left(\mathbf{a}_k; (\mathbf{z} - \mathbf{b}_{k-1}) \frac{\sigma_k^2}{s_{k-1}^2}, \frac{s_k^2 \sigma_k^2}{s_{k-1}^2} \mathbf{I}\right) \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}) d\mathbf{z}.
\end{aligned}$$

Considering the first Gaussian density, we can express it as the generative process

$$\mathbf{a}_k = \frac{\sigma_k^2}{s_{k-1}^2} (\mathbf{z} - \mathbf{b}_{k-1}) + \varepsilon \sqrt{\frac{s_k^2 \sigma_k^2}{s_{k-1}^2}}$$

where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and rearranging yields

$$\mathbf{z} = \frac{s_{k-1}^2}{\sigma_k^2} \mathbf{a}_k + \mathbf{b}_{k-1} + \varepsilon \sqrt{\frac{s_k^2 s_{k-1}^2}{\sigma_k^2}}.$$

Therefore it follows that

$$\mathcal{N}\left(\mathbf{a}_k; (\mathbf{z} - \mathbf{b}_{k-1}) \frac{\sigma_k^2}{s_{k-1}^2}, \frac{s_k^2 \sigma_k^2}{s_{k-1}^2} \mathbf{I}\right) \propto \mathcal{N}\left(\mathbf{z}; \frac{s_{k-1}^2}{\sigma_k^2} \mathbf{a}_k + \mathbf{b}_{k-1}, \frac{s_k^2 s_{k-1}^2}{\sigma_k^2} \mathbf{I}\right).$$

Using again the rule for products of Gaussian densities,

$$\begin{aligned}
&\mathcal{N}\left(\mathbf{a}_k; (\mathbf{z} - \mathbf{b}_{k-1}) \frac{\sigma_k^2}{s_{k-1}^2}, \frac{s_k^2 \sigma_k^2}{s_{k-1}^2} \mathbf{I}\right) \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}) \\
\propto &\mathcal{N}\left(\mathbf{z}; \frac{s_{k-1}^2}{\sigma_k^2} \mathbf{a}_k + \mathbf{b}_{k-1}, \frac{s_k^2 s_{k-1}^2}{\sigma_k^2} \mathbf{I}\right) \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}) \\
= &\gamma_d^{-1} \mathcal{N}(\mathbf{z}; \mathbf{m}_d, \mathbf{M}_d).
\end{aligned}$$

Substituting this back in we get

$$
\begin{aligned}
q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) &\propto \sum_{d=1}^{D} \frac{\lambda_d^{-1^{(k)}}}{\sum_{j=1}^{D} \lambda_j^{-1^{(k)}}} \int \mathcal{N}\left(\mathbf{z}; \frac{s_{k-1}^2}{\sigma_k^2}\mathbf{a}_k + \mathbf{b}_{k-1}, \frac{s_k^2 s_{k-1}^2}{\sigma_k^2}\mathbf{I}\right) \mathcal{N}(\mathbf{z}; \mathbf{c}_d^{(k)}, \mathbf{C}_d^{(k)}) d\mathbf{z}. \\
&= \sum_{d=1}^{D} \frac{\lambda_d^{-1^{(k)}}}{\sum_{j=1}^{D} \lambda_j^{-1^{(k)}}} \int \gamma_d^{-1} \mathcal{N}(\mathbf{z}; \mathbf{m}_d, \mathbf{M}_d) d\mathbf{z}. \\
&= \sum_{d=1}^{D} \frac{\lambda_d^{-1^{(k)}}}{\sum_{j=1}^{D} \lambda_j^{-1^{(k)}}} \gamma_d^{-1}.
\end{aligned}
$$

Note that γ_d^{-1} is itself a Gaussian pdf of the form

$$
\gamma_d^{-1} = \mathcal{N}\left(\mathbf{c}_d^{(k)}; \frac{s_{k-1}^2}{\sigma_k^2}\mathbf{a}_k + \mathbf{b}_{k-1}, \frac{s_k^2 s_{k-1}^2}{\sigma_k^2}\mathbf{I} + \mathbf{C}_d^{(k)}\right)
$$

and again we can write the generative process

$$
\mathbf{c}_d^{(k)} = \frac{s_{k-1}^2}{\sigma_k^2}\mathbf{a}_k + \mathbf{b}_{k-1} + \varepsilon
$$

where $\varepsilon \sim \mathcal{N}\left(\mathbf{0}, \frac{s_{k-1}^2 s_k^2}{\sigma_k^2}\mathbf{I} + \mathbf{C}_d^{(k)}\right)$. Rearranging this yields

$$
\mathbf{a}_k = \frac{\sigma_k^2}{s_{k-1}^2}(\mathbf{c}_d^{(k)} - \mathbf{b}_{k-1}) + \frac{\sigma_k^2}{s_{k-1}^2}\varepsilon.
$$

It follows that

$$
\gamma_d^{-1} \propto \mathcal{N}\left(\mathbf{a}_k; \frac{\sigma_k^2}{s_{k-1}^2}(\mathbf{c}_d^{(k)} - \mathbf{b}_{k-1}), \frac{s_k^2 \sigma_k^2}{s_{k-1}^2}\mathbf{I} + \frac{\sigma_k^4}{s_{k-1}^4}\mathbf{C}_d^{(k)}\right)
$$

and putting this altogether we get

$$
q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) \propto \sum_{d=1}^{D} \frac{\lambda_d^{-1^{(k)}}}{\sum_{j=1}^{D} \lambda_j^{-1^{(k)}}} \mathcal{N}\left(\mathbf{a}_k; \frac{\sigma_k^2}{s_{k-1}^2}(\mathbf{c}_d^{(k)} - \mathbf{b}_{k-1}), \frac{s_k^2 \sigma_k^2}{s_{k-1}^2}\mathbf{I} + \frac{\sigma_k^4}{s_{k-1}^4}\mathbf{C}_d^{(k)}\right).
$$

Integrating both sides with respect to $\mathbf{a}_k$ shows that the RHS is normalised, and so we can conclude:

$$q(\mathbf{a}_k \mid \mathbf{a}_{1:k-1}) = \sum_{d=1}^{D} \frac{\lambda_d^{-1(k)}}{\sum_{j=1}^{D} \lambda_j^{-1(k)}} \mathcal{N}\left(\mathbf{a}_k; \frac{\sigma_k^2}{s_{k-1}^2}(\mathbf{c}_d^{(k)} - \mathbf{b}_{k-1}), \frac{s_k^2 \sigma_k^2}{s_{k-1}^2}\mathbf{I} + \frac{\sigma_k^4}{s_{k-1}^4}\mathbf{C}_d^{(k)}\right) \tag{A.7}$$

which is another mixture of Gaussians.

A.3 Uniform Variances Maximise Euclidean Distance

First we show for $u \geq 0$,

$$\frac{1+u-(u-1)^2}{2} \leq \sqrt{u} \leq \frac{u+1}{2}. \tag{A.8}$$

For the lower bound, note that for $u \geq 0$,

$$(u-1)^2 \geq (\sqrt{u}-1)^2$$

which rearranges to give

$$\frac{1}{2}(1+u-(u-1)^2) \leq \sqrt{u}.$$

For the upper bound, when $u \geq 0$,

$$0 \leq (u-1)^2 = u^2 - 2u + 1$$

adding $4u$ to both sides gives

$$4u \leq u^2 + 2u + 1 = (u+1)^2.$$

Taking square root on both sides and rearranging gives

$$\sqrt{u} \leq \frac{1}{2}(u+1)$$

and so we have shown Equation (A.8).

Suppose we encode $\mathbf{w} \in \mathbb{R}^N$ using K auxiliary variables and without loss of generality assume $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, \mathbb{I})$, then we have that each auxiliary variable $\mathbf{a}_i \sim \mathcal{N}(\mathbf{a}_i; 0, \sigma_i^2 \mathbb{I})$ with $\sum_{i=1}^{K} \sigma_i^2 = 1$.

We want to find which values for σ_i^2 maximise $\mathbb{E}(\|\mathbf{w}\|)$. Consider $Y = \|\mathbf{w}\|^2$, and set $U := \frac{1}{N} Y^2 = \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{K} \mathbf{a}_{ij}^2$, where $\mathbf{a}_{ij}^2$ is the j-th element of the i-th auxiliary. To start note

the following

$$\begin{aligned}\mathbb{E}[U] &= \mathbb{E}\left[\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{K}\mathbf{a}_{ij}^2\right] \\ &= \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{K}\mathbb{E}[\mathbf{a}_{ij}^2] \\ &= 1\end{aligned}$$

$$\begin{aligned}\mathrm{Var}(\mathbf{a}_{ij}^2) &= \mathbb{E}[\mathbf{a}_{ij}^2] - \mathbb{E}^2[\mathbf{a}_{ij}^2] \\ &= 2\sigma_i^4\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(U-1)^2] &= \mathrm{Var}(U) \\ &= \mathrm{Var}\left[\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{K}\mathbf{a}_{ij}^2\right] \\ &= \frac{1}{N^2}\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{K}\mathrm{Var}[\mathbf{a}_{ij}^2] \\ &= \frac{2}{N}\sum_{i=1}^{K}\sigma_i^4.\end{aligned}$$

Now we can substitute U into Equation (A.8) and take expectations to yield

$$1 - \frac{1}{N}\sum_{i=1}^{K}\sigma_i^4 \leq \frac{1}{\sqrt{N}}\mathbb{E}[\|\mathbf{w}\|] \leq 1. \tag{A.9}$$

We want to set σ_i^2 to make this bound as tight as possible. To see this is achieved when $\sigma_i^2 = 1/K$ for all $i = 1, ..., K$, we have by the Cauchy-Schwartz inequality,

$$\begin{aligned}1 = \left(\sum_{i=1}^{K}\sigma_i^2\right) &= \left(\sum_{i=1}^{K}\sigma_i^2 \cdot 1\right) \\ &\leq \left(\sum_{i=1}^{K}\sigma_i^4\right)\left(\sum_{i=1}^{K}1^2\right) \\ &= K\sum_{i=1}^{K}\sigma_i^4\end{aligned}$$

thus

$$\frac{1}{K} \leq \sum_{i=1}^{K} \sigma_i^4.$$

Since

$$\sum_{i=1}^{K} (\frac{1}{K})^2 = \frac{1}{K}$$

setting

$$\sigma_i^2 = \frac{1}{K} \tag{A.10}$$

for all $i = 1, ..., K$ achieves the bound.

Appendix B

Further Results and Model Details

Below we give specific implementation details and some extra results. In particular, note that we used HMC implementation from Pyro [Bingham et al., 2018] which allows a tune-able step-size.

B.1 Bayesian Linear Regression

B.1.1 Extra Results

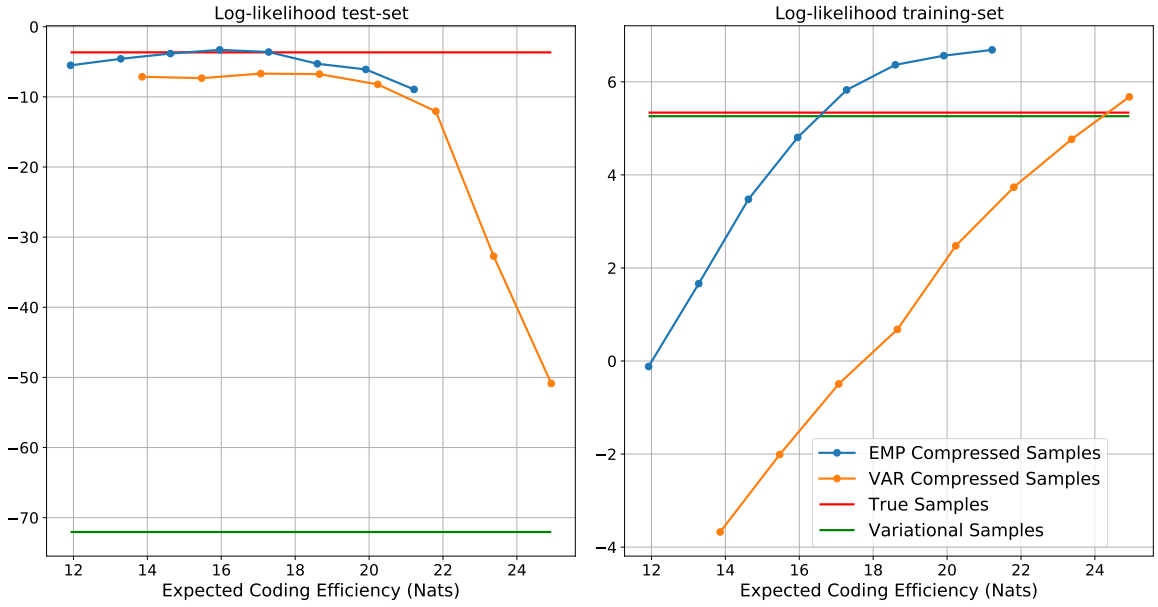


Fig. B.1 Plot of the performance of samples on a 5D linear regression task.

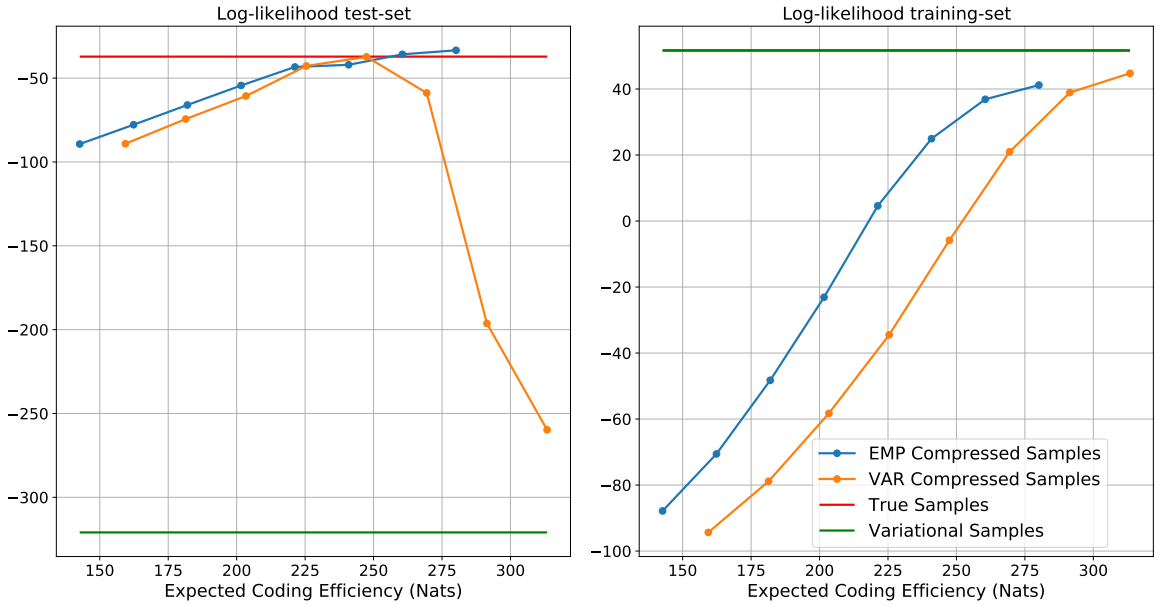


Fig. B.2 Plot of the performance of samples on a 50D linear regression task.

B.1.2 Trajectory Plots

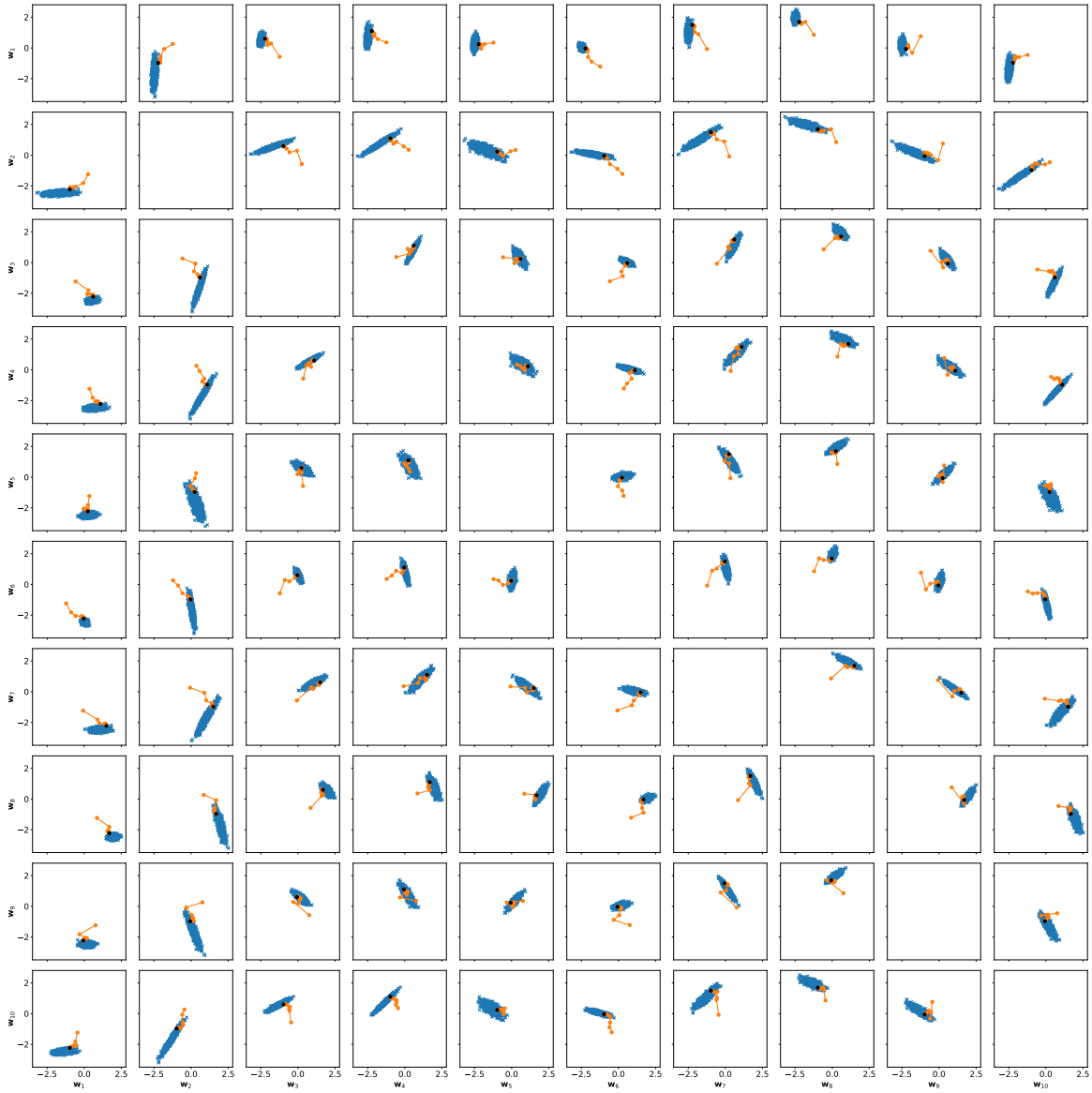


Fig. B.3 Plot of the trajectory of `MOD-Scheme` on 10D linear regression task with medium compression allowance. The reader is encouraged to zoom in to see the trajectories in detail.

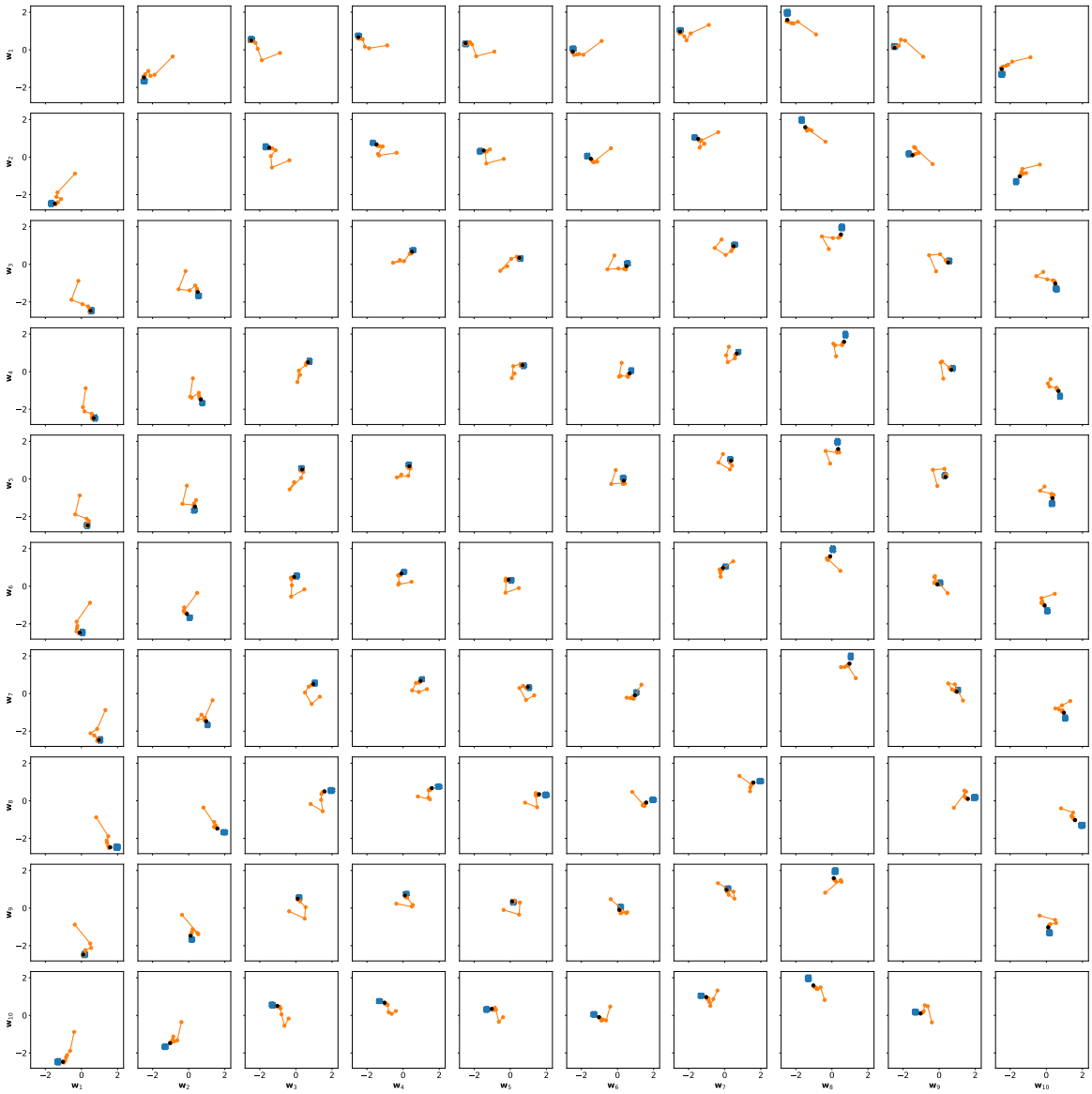


Fig. B.4 Plot of the trajectory of `FG-Scheme` on 10D linear regression task with medium compression allowance. The reader is encouraged to zoom in to see the trajectories in detail.

B.2 BNN Regression - Toy Problem

B.2.1 Implementation Details

HMC

- Total Samples: 1000 samples
- Warmup Samples: 1000 samples
- Steps per sample: 15
- Initial Step Size: 0.001
- Desired Acceptance Rate: 0.8

KDE

- # Samples used from HMC: 500
- Training Objective: optimise isotropic noise for each Gaussian component according to β-ELBO
- Training Epochs: 10000
- Optimiser: Adam [Kingma and Ba, 2014], learning-rate 0.001

Factored Gaussian

- Training Objective: β-ELBO
- Training Epochs: 10000
- Optimiser: Adam, learning-rate 0.001

B.2.2 Trajectory Plots

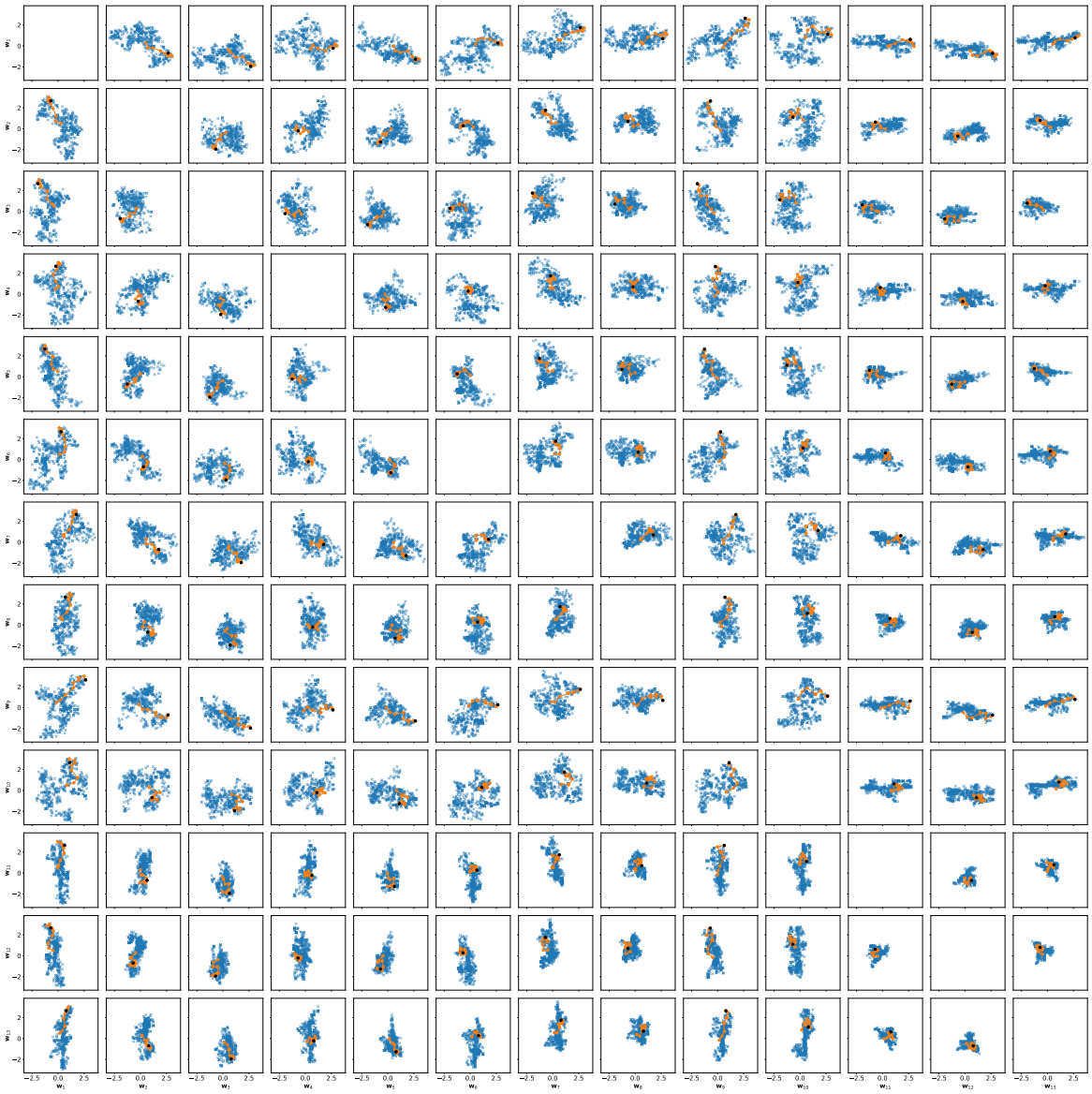


Fig. B.5 Plot of the trajectory of `MOD-Scheme` on toy regression problem using BNNs. Here we plot all the weights against one another. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample. The reader is encouraged to zoom in to see the trajectories in detail.

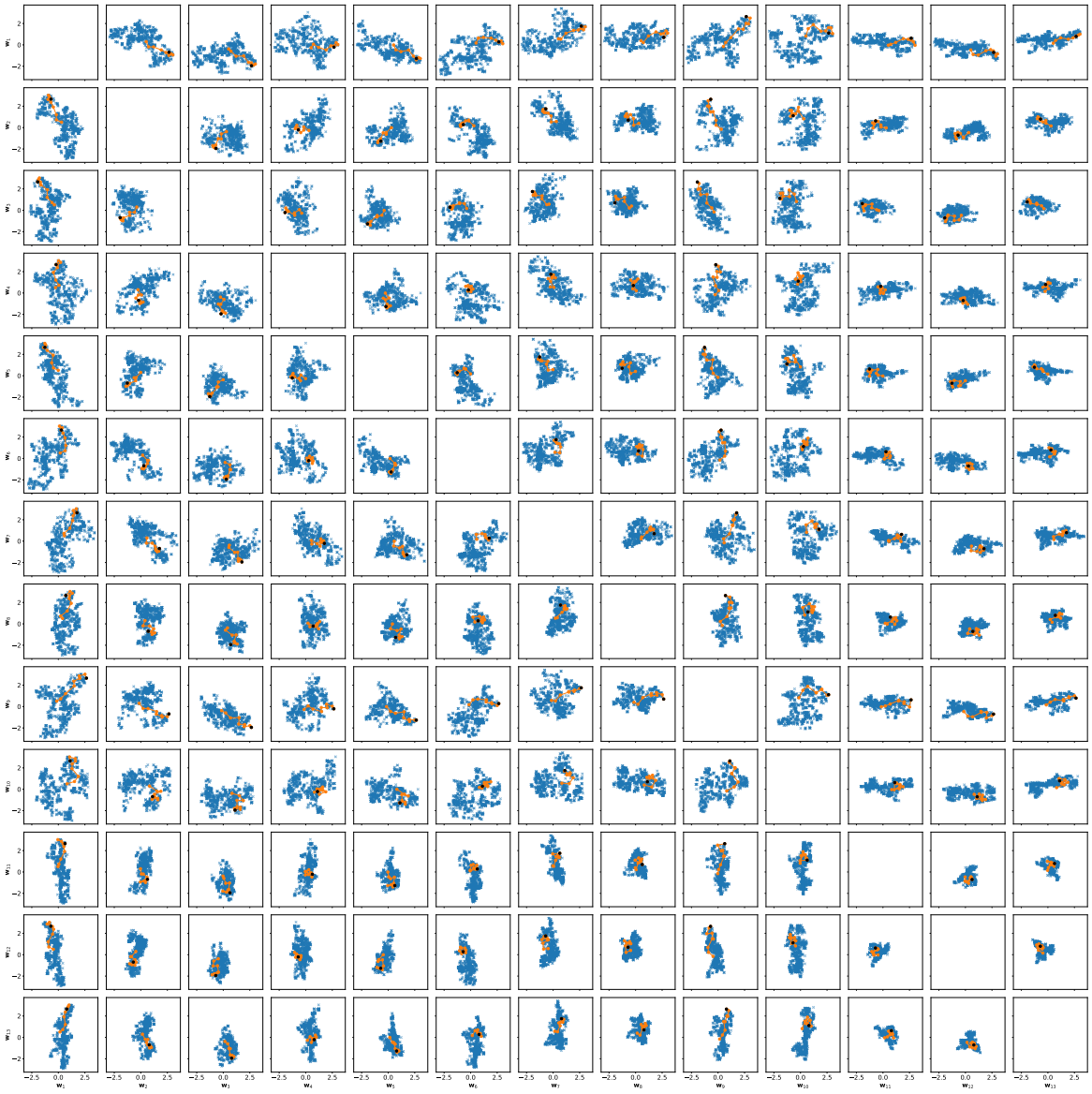


Fig. B.6 Plot of the trajectory of `KDE-Scheme` on toy regression problem using BNNs. Here we plot all the weights against one another. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample. The reader is encouraged to zoom in to see the trajectories in detail.

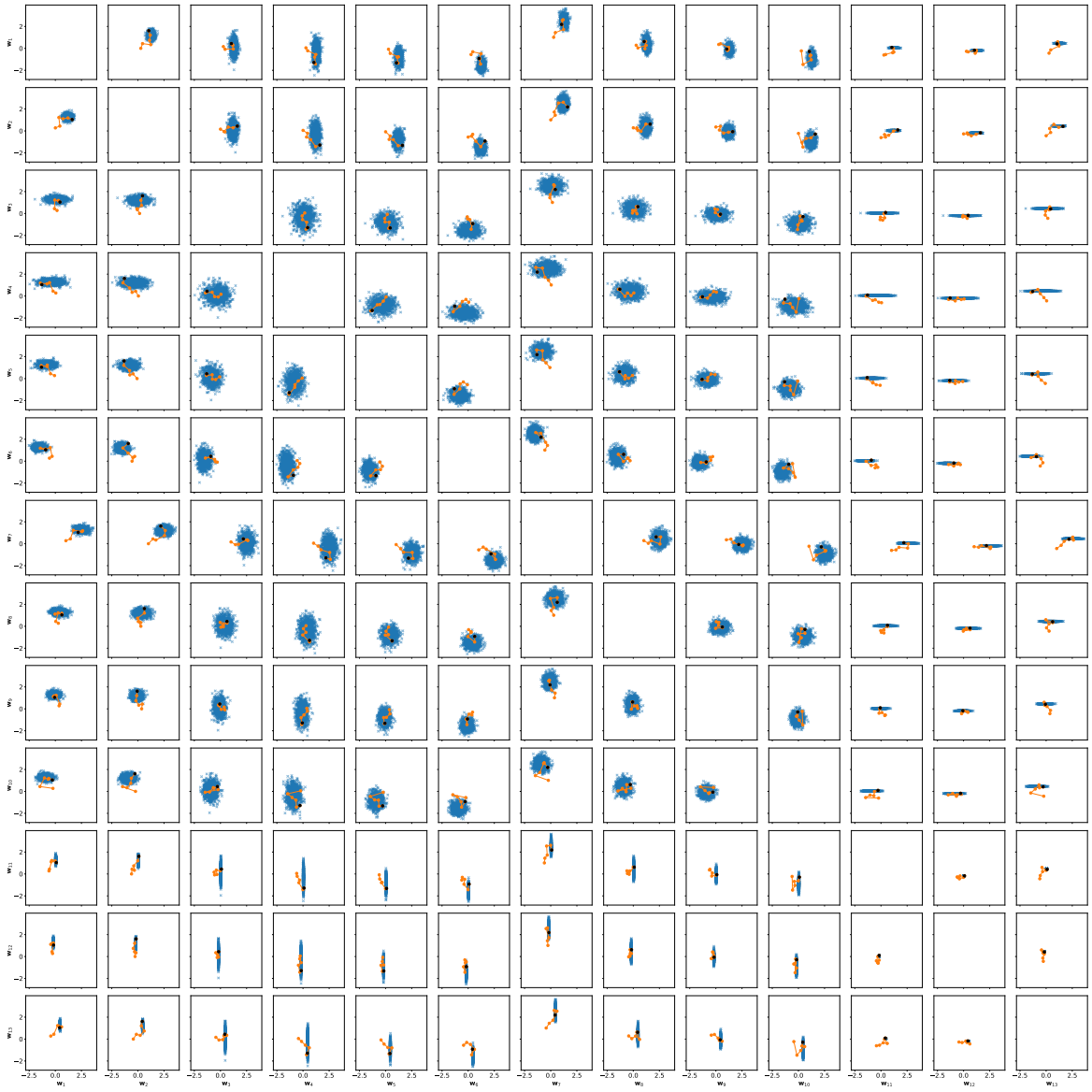


Fig. B.7 Plot of the trajectory of `FG-Scheme` on toy regression problem using BNNs. Here we plot all the weights against one another. Compared to the other trajectory plots in Figure B.5 and Figure B.6, the target is much smaller and has less variance. The solid lines depict the trajectory of auxiliary variables used to compress a sample. The markers show each additional auxiliary variable, and the black marker shows the final position of the compressed sample. The reader is encouraged to zoom in to see the trajectories in detail.

B.3 BNN Regression - UCI Energy

B.3.1 Implementation Details

HMC

- Total Samples: 1000 samples
- Warmup Samples: 1000 samples
- Steps per sample: 15
- Initial Step Size: 0.001
- Desired Acceptance Rate: 0.8

KDE

- # Samples used from HMC: 100
- Training Objective: optimise isotropic noise for each Gaussian component according to β-ELBO
- Training Epochs: 10000
- Optimiser: Adam, learning-rate 0.001

Factored Gaussian

- Training Objective: β-ELBO
- Training Epochs: 10000
- Optimiser: Adam, learning-rate 0.001