Probabilistic reasoning for uncertainty & compression in deep learning

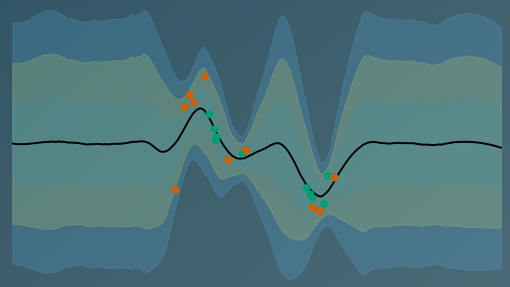
Louizos, C.

Publication date
2022
Document Version
Final published version

Probabilistic Reasoning for Uncertainty & Compression in Deep Learning Christos Louizos

Christos Louizos

Probabilistic Reasoning for Uncertainty & Compression in Deep Learning



Probabilistic Reasoning for Uncertainty & Compression in Deep Learning

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 16 februari 2022, te 10.00 uur

door Christos Louizos
geboren te Amarousio

Promotiecommissie

Promotor:	prof. dr. M. Welling	Universiteit van Amsterdam
Copromotor:	prof. dr. J.M. Mooij	Universiteit van Amsterdam
Overige leden:	prof. dr. Y.W. Teh	University of Oxford
	prof. dr. D. Vetrov	HSE University Moscow
	dr. P.D. Forré	Universiteit van Amsterdam
	dr. E. Gavves	Universiteit van Amsterdam
	prof. dr. ir. C.I. Sánchez Gutiérrez	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Funding of this research was provided by NWO and TNO.

Dedicated to the loving memory of Angelliki Gerardi.
1969 – 2009

PROBABILISTIC REASONING FOR UNCERTAINTY AND COMPRESSION IN DEEP LEARNING: SUMMARY

There is a wide range of tasks where the predictive capabilities of neural networks and deep learning currently excel. As a result, this technology is being employed in various applications that play an important role in our everyday lives. Therefore, potential improvements of this technology have become an even more important topic. In this thesis, we work towards improving two important aspects of deep learning models; the ability to represent the uncertainty of their predictions and the inherent need for large amounts of compute and resources.

We begin this work by providing an introduction and state the two main research questions that this thesis answers. We further provide the necessary background knowledge for the main techniques that will be used throughout this thesis. We describe neural networks and Bayesian neural networks, i.e. neural networks where their parameters (aka weights and biases) are random and governed by a probability distribution instead of being fixed, along with (variational) Bayesian inference, a way to update the probability distribution over the parameters in light of observed data. Finally, we also provide a brief introduction to neural network compression via pruning, which removes irrelevant parameters and parts of the network by explicitly setting them to zero, and quantization, which represents the numerical values of the weights and intermediate representations of the network in hardware friendly formats (e.g. fixed point).

The first part of this thesis describes three contributions which improve the uncertainty estimation capabilities of neural networks. The first two revolve around improving the uncertainty quality of variational Bayesian neural networks through better approximations to the probability distribution over their parameters when we observe the data. Under this view, we propose a simple way to introduce linear dependencies between the neural network weights through *matrix variate Gaussian* distributions; they are distributions over random matrices and can readily allow for modelling the correlations among the input and output neurons in each layer, an ability which leads to improved performance as we experimentally show. We then propose *multiplicative normalizing flows*, a general framework that induces non-linear dependencies among the parameters of the network. This is realized by combining auxiliary random variables and parametrized bijections in a way that allows for flexible correlations among the weights of each layer while still being computationally tractable. Experimentally, we show that the uncertainty quality is improved when we compare against the simpler Gaussian variational approximations of prior work. The last contribution of this part corresponds to the *functional neural process*, a model which employs a different viewpoint; instead of positing probability distributions and performing (variational) inference over the neural network weights, it adopts the modelling framework of stochastic processes and thus posits probability distributions and performs inference over the function space of the neural network. This has the extra benefit of an

easier inference and a more intuitive modelling task, as it allows us to reason about the relations among points in the dataset, realized through the introduction of a "reference" set of points, instead of the non-interpretable neural network parameters. Experimentally, we show that such a model provides better uncertainty quality while simultaneously maintaining comparable predictive performance.

The second part describes three novel compression techniques that can allow us to learn neural networks that are both smaller and faster, thus reducing the amount of compute and resources that are required. The first contribution corresponds to *Bayesian compression*, a variational Bayesian inference procedure that through well chosen probability distributions over the parameters of the network can uncover performant and computationally efficient architectures via joint pruning and quantization. While such an approach can lead to highly compressed architectures, it lacks the fine-grained adaptation of either pruning or quantization to a specific task or problem. For this reason the other two contributions aim at tackling pruning and quantization separately. The second contribution corresponds to a novel optimization method for the L_0 norm, the golden standard for sparsity, of neural networks. To this end, we propose a general purpose technique that through appropriate amounts of noise can allow for *gradient based optimization of the non-differentiable* L_0 *norm*. Empirically, we show that such an approach leads to accurate and highly sparse models while it can allow for sparse training through conditional computation and proper software, a fact that can facilitate faster training as well. Finally, for the last contribution we employ similar ideas and introduce *relaxed quantization*; a gradient based optimization procedure that allows for learning neural networks where their parameters and activations lie on an (adaptive) quantization grid. We empirically show that this allows us to train accurate neural networks on large scale tasks, while spending as few as 4 bits per weight and activation.

We conclude this thesis by providing the answers to the research questions while also discussing the pitfalls and drawbacks of the proposed methods and stating promising research directions.

CONTENTS

1

INTRODUCTION & BACKGROUND

Deep learning is becoming an increasingly relevant and important part for our day to day lives. It lies at the core of multiple applications; it is used in Facebook as a way to identify faces in pictures with close to human-level performance [1], it provides automatic translations at Google translate [2], it aides in breast cancer diagnoses [3] and can allow for personalization on mobile devices without compromising privacy [4]. These are only a handful of mentions and the volume of such applications is likely to increase in the future, as tools such as Tensorflow [5] and PyTorch [6] allow one to easily apply the deep learning machinery to novel tasks [7].

As a result, we need to be serious about this technology and the implications it can have in our society. It has the potential to significantly improve our lives, by allowing for more automation and by complementing the set of skills that we humans have. Nevertheless, we also need to be honest and aware about its limitations, while collectively working towards improving them. In this way we can avoid wide range negative effects that can arise from an improper use or undesirable behaviour of deep learning models. Improving upon such limitations is the main focus of this thesis.

In the following, we provide an introduction of the main concepts that are used throughout this work as well as describe its main research objectives. The details about the notation are provided at appendix 10. Section 1.1, provides details about neural networks, the cornerstone of deep learning, whereas section 1.1.1 illustrates two core limitations of current deep learning models that constitute the research questions of this work. The rest of the sections include appropriate background knowledge, sections 1.2,1.3,1.4, while section 1.5 provides a summary of the contributions of this thesis.

1.1 DEEP LEARNING

Deep learning [54] is the resurgence of the concept of neural networks. In their simplest form, they are comprised from a sequence of layers, each of which performs a linear

1 DeepFace
2 Deep Learning & Google Translate
3 Deep Learning & breast cancer diagnosis
4 Apple core ML-3
5 https://www.tensorflow.org
6 https://pytorch.org
7 AI powered catflap

transformation of its input followed by an element-wise nonlinearity. They are flexible functions that can effectively model many complex input - output relationships; mappings from natural images to fine-grained categories, text to speech and translation from a source to a target language, just to name a few. The parameters of each linear transformation can be optimized for the given task, and the flexibility of the function can be increased by either adding more layers or by adding more parameters at each layer.

More formally, given a D-dimensional input vector $\mathbf{x}$ a neural network can provide an output $\mathbf{h}$ via the following procedure

$$\mathbf{h} = (f_L \circ \cdots \circ f_1)(\mathbf{x}) \tag{1}$$

where L denotes the number of layers in the network and each $f_i(\cdot)$ is defined as

$$f_i(\mathbf{x}) = \psi(\mathbf{W}_i^\top \mathbf{x} + \mathbf{b}_i) \tag{2}$$

with $\mathbf{W}_i \in \mathbb{R}^{D\times H}$, $\mathbf{b}_i \in \mathbb{R}^H$ being the weights and biases, i.e. the parameters of the layer, and $\psi(\cdot)$ being an element-wise nonlinearity such as the ReLU [131], $\psi(x) = \max(0, x)$. Each $f_i(\cdot)$ can be considered as an intermediate "activation" or "feature" representation of the input $\mathbf{x}$.

Let $\theta = \{\mathbf{W}_{1:L}, \mathbf{b}_{1:L}\}$ be the collection of parameters and $\mathcal{D} = \{(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_N, y_N)\}$, be a dataset of independent and identically distributed (i.i.d.) datapoints, with $\mathbf{x}_i$ being the inputs, e.g. natural images, and y_i being the desired outputs, e.g. a real value for regression or a category for classification. We can view the neural network as a *probabilistic model* for the given dataset, where the network output provides the parameters of the assumed distribution over the outputs y_i. For example, when we are faced with supervised learning on an i.i.d. dataset we have that

$$p(\mathcal{D}) = \prod_{i\in\mathcal{D}} p(y = y_i|\mathbf{x}_i, \theta), \tag{3}$$

with $p(y|\mathbf{x}, \theta)$ being the probability distribution given by the neural network. In case of classification $p(y|\mathbf{x}, \theta)$ is assumed to be a categorical distribution with the output of the network being the probabilities π_j of each category obtained via a softmax transformation, $\pi_j = \frac{\exp(h_j)}{\sum_k \exp(h_k)}$, with $\mathbf{h} = (f_L \circ \cdots \circ f_1)(\mathbf{x})$. For a regression task, $p(y|\mathbf{x}, \theta)$ is usually assumed to be a unit variance Gaussian distribution with a mean given by the network, i.e. $p(y|\mathbf{x}, \theta) = \mathcal{N}(y|(\mathbf{h}, 1)$.

Having defined the neural network, we are then interested in optimizing its parameters, θ, for a given task, e.g. correctly predicting the category of a given image. In this way, the network can "learn" the underlying mechanism for prediction and thus provide us answers for novel inputs. Given the aforementioned probabilistic view, we can optimize θ in a way that *maximizes* the probability of the observed dataset $\mathcal{D}$. This corresponds to the well-known *maximum likelihood* (ML) criterion [21]. For example, in the case of classification we can maximize

$$\begin{aligned}\log p(\mathcal{D}|\theta) &= \sum_{i\in\mathcal{D}} \log p(y = y_i|\mathbf{x}_i, \theta) \\ &= \sum_{i\in\mathcal{D}} \sum_j \mathbb{I}[j = y_i] \left(h_j - \log \sum_k \exp(h_k) \right),\end{aligned} \tag{4}$$

which is equivalent to minimizing the cross-entropy loss, the de-facto loss for classification problems with neural networks [54]. For regression we can similarly derive that

$$\begin{aligned}\log p(\mathcal{D}|\theta) &= \sum_{i\in\mathcal{D}} -\frac{1}{2}\left((y_i - h_i)^2 + \log 2\pi\right) \\ &\propto \sum_{i\in\mathcal{D}} -\frac{(y_i - h_i)^2}{2}\end{aligned} \tag{5}$$

which is equivalent to the well know mean square loss for regression problems [54]. Scalable optimization for such objectives is usually performed with *backpropagation* [54]; it corresponds to propagating the errors made in the output "backward" to the parameters of the network. This is achieved by computing the gradient of the objective on (a subset of) the data with respect to the parameters via the chain-rule and then performing updates in a way that maximizes the objective or, equivalently, minimizes the loss. Essentially, this is a form of *stochastic gradient descent* (SGD) [54].

After obtaining the optimized parameters θ^* from our optimizer of choice, we often seek to make predictions for novel inputs $\mathbf{x}^*$. Such predictions are usually made by picking the most probable category / value (mode) of the output distribution of the network

$$y^* = \arg\max_y p(y|\mathbf{x}, \theta^*) = \arg\max_y \log p(y|\mathbf{x}, \theta^*) \tag{6}$$

which corresponds to the most probable class in the case of classification or the mean of the Gaussian likelihood in the case of regression.

1.1.1 *Limitations of Deep Learning & Research Questions*

While being straightforward, deep learning has, as almost every technology, its limitations. Consider the breast cancer diagnosis example from the introduction; while the high predictive accuracy that deep learning can achieve due to its flexible form is definitely desirable, for a real world application we also require an accurate estimation of the *uncertainty* of the given prediction. How likely is it that the model is correct in the prediction of cancer? Or else, how good is the predictive distribution $p(y|\mathbf{x}, \theta^*)$? This will subsequently help in deciding whether some form of, usually expensive, treatment is necessary. Consider the alternative example of a self-driving car entering in a previously unseen situation or environment; an accurate form of uncertainty could help in identifying such cases and encourage the autonomous system to ask for expert advice (i.e. pass the steering wheel to the human driver). Unfortunately, vanilla neural networks do not currently possess a way to obtain reliable and robust uncertainty estimates. For this reason, we would like to work towards principled methods, as they have theoretically grounded guarantees about their behaviour thus can provide us with robust uncertainties, i.e. good behaviour on a broad range of tasks. This brings us to the first research question of this thesis:

Research question 1: *Can we enhance neural networks with the ability to represent their uncertainty in a principled and robust way?*

In order to address this question we will employ Bayesian modelling, an elegant framework that naturally accommodates for uncertainty. We provide the necessary background at section 1.2 and its application to deep learning at section 1.3.

Another main limitation is that the success of deep learning comes at a significant computational cost. Modern deep learning architectures can easily scale to millions of parameters and can require large amounts of compute [8]. As a result, practical applications in resource constrained devices, such as mobile phones and IoT devices, cannot employ the large architectures devised currently; the amount of parameters θ is too large, the intermediate activations $f_i(\mathbf{x})$ can consume a large amount of memory and necessary matrix multiplications require a large amount of floating point operations (FLOPs). Furthermore, deep learning systems that are employed at large scale tasks can make billions of predictions per day, a fact that not only comes with substantial energy costs and power consumption but can negatively impact the environment as well [9]. This leads to the second research question of this thesis:

Research question 2: *Can we make neural networks smaller and faster while maintaining good performance?*

To answer this question we will explore neural network compression; making neural networks smaller and faster while preserving their predictive capabilities. We provide the appropriate background knowledge at section 1.4.

1.2 BAYESIAN & APPROXIMATE BAYESIAN INFERENCE

Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the observed dataset that is comprised from inputs $\mathbf{x}$ and outcomes y, and let θ be the set of the latent (i.e. unobserved) variables of the model. Bayesian inference is at its core the application of Bayes rule over the unknowns of a given model. It involves the *likelihood*, the probability of the observed dataset $p(\mathcal{D}|\theta)$ given an instantiation of the latent parameters, and the *prior* $p(\theta)$, our a-priori defined beliefs about the latent variables. Bayesian inference corresponds to *updating* our prior beliefs in light of the observed dataset $\mathcal{D}$ to the *posterior* distribution over those beliefs

$$\underbrace{p(\theta|\mathcal{D})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}\overbrace{p(\theta)}^{\text{prior}}}{\underbrace{\int p(\mathcal{D}|\theta)p(\theta)d\theta}_{\text{evidence}}} = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \tag{7}$$

where the normalizing constant $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$ is denoted as the *marginal likelihood* or else *evidence*. It is the probability of the observed dataset under our assumed model. We provide an example illustration of this concept at Figure 1.

Bayesian inference is attractive as it can allow us to encode prior knowledge about the data in the prior and likelihood. Furthermore, it naturally handles the *uncertainty* we have

8 AI & compute
9 Deep learning & carbon emissions

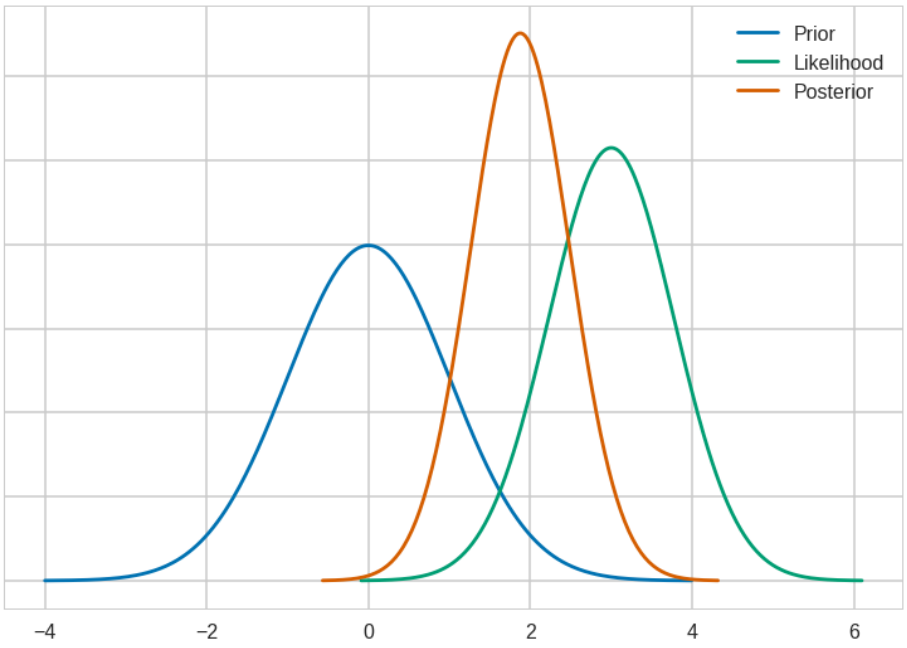


Figure 1: Illustrative example of Bayesian inference. Given a standard normal prior distribution over a scalar parameter θ, $p(\theta) = \mathcal{N}(0, 1)$, and a simple Gaussian likelihood with variance $\sigma^2 = 0.6$, i.e. $p(x|\theta) = \mathcal{N}(\theta, 0.6)$, we visualize the posterior distribution over θ given a dataset $\mathcal{D} = \{3\}$, which for this specific case is $p(\theta|\mathcal{D}) = \mathcal{N}\left(\frac{x}{\sigma^2+1}, \left(\frac{1}{\sigma^2}+1\right)^{-1}\right)$. We can see that, intuitively, the mean of the posterior distribution is a weighted average of the prior mean and the given datapoint, while the uncertainty about this value is based on the uncertainty of the prior, i.e. the prior variance, and the uncertainty of the data, i.e. the likelihood variance.

about the model, as it provides a formal mechanism to update the uncertainty about our beliefs over the latent variables θ. We provide an illustrative example at Figure 1. It also provides a principled way to make predictions about the "future", conditioned on what we have seen in "the past" (e.g. the dataset $\mathcal{D}$). Such predictions are made with the *posterior predictive distribution*:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p(y^*|\mathbf{x}^*, \theta) p(\theta|\mathcal{D}) d\theta, \tag{8}$$

where $\mathbf{x}^*$ corresponds to "future" observations and y^* to their possible outcomes.

Unfortunately, analytically obtaining the evidence that is necessary for exact Bayesian inference is usually very hard. For the complex non-linear models that we are interested in this thesis, i.e. neural networks, it is intractable to compute, as exact marginalization over all possible states for the latent variables may not be easily available or can be expensive to obtain. For this reason we are usually interested in performing *approximate* Bayesian inference, where we aim to compute an approximation to the true posterior distribution over said latent variables. For the purpose of this thesis we adopt variational inference [21], a tool to deterministically obtain such approximations. It involves positing an approximate

distribution $q_\phi(\theta)$ that has parameters $\boldsymbol{\phi}$ which are optimized in a way that maximizes the *Evidence Lower Bound* (ELBO)

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}) :&= \int q_{\boldsymbol{\phi}}(\theta)\left(\log p(\mathcal{D}|\theta) + \log p(\theta) - \log q_{\boldsymbol{\phi}}(\theta)\right) d\theta \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\theta)}\left[\log p(\mathcal{D}|\theta)\right] - \mathrm{KL}(q_{\boldsymbol{\phi}}(\theta)||p(\theta)). \end{aligned} \tag{9}$$

As the name implies, the ELBO is a lower bound of the marginal likelihood $p(\mathcal{D})$, which can be shown via the following

$$\log p(\mathcal{D}) = \int q_{\boldsymbol{\phi}}(\theta) \log p(\mathcal{D}) d\theta \tag{10}$$

$$= \int q_{\boldsymbol{\phi}}(\theta) \log \frac{p(\mathcal{D},\theta) q_{\boldsymbol{\phi}}(\theta)}{p(\theta|\mathcal{D}) q_{\boldsymbol{\phi}}(\theta)} d\theta \tag{11}$$

$$= \mathcal{L}(\boldsymbol{\phi}) + \underbrace{\int q_{\boldsymbol{\phi}}(\theta) \log \frac{q_{\boldsymbol{\phi}}(\theta)}{p(\theta|\mathcal{D})} d\theta}_{\mathrm{KL}\left(q_{\boldsymbol{\phi}}(\theta)||p(\theta|\mathcal{D})\right)} \tag{12}$$

$$\geqslant \mathcal{L}(\boldsymbol{\phi}), \tag{13}$$

where the last step is due to the positivity of the KL-divergence, a measure of discrepancy between two distributions. After a simple re-arrangement of the terms we can also arrive at the following expression

$$\mathrm{KL}\left(q_{\boldsymbol{\phi}}(\theta)||p(\theta|\mathcal{D})\right) = \log p(\mathcal{D}) - \mathcal{L}(\phi), \tag{14}$$

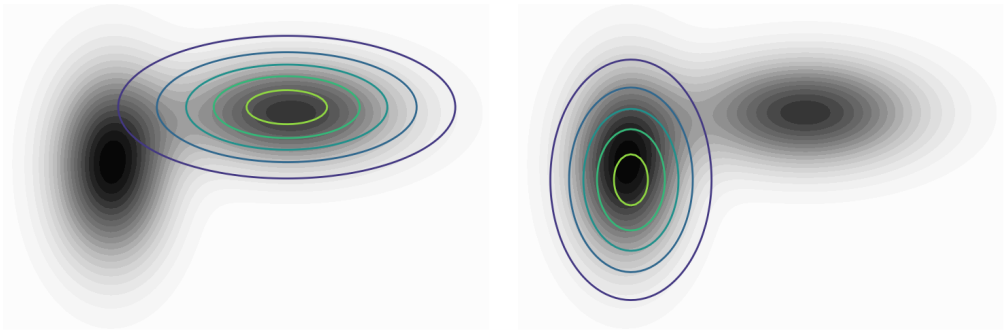
where we can see that maximizing the ELBO corresponds to minimizing the KL-divergence of the approximate distribution to the posterior distribution $p(\theta|\mathcal{D})$ as $\log p(\mathcal{D})$ is a constant w.r.t. the optimization of $\boldsymbol{\phi}$.

1.2.1 *Gradient Based Variational Inference*

We have seen how variational inference can provide us a tool to deterministically obtain an approximation to the posterior distribution over the latent variables. Nevertheless, by observing the expression at Eq. 9, we see that we are still faced with integrals that can be intractable to compute, e.g. in the case that we use a complex non-linear model, such as a neural network, for the likelihood $p(\mathcal{D}|\theta)$. However, advances in variational inference [91, 151] have shown that for continuous latent variables θ we can effectively bypass this problem by optimizing Eq. 9 with backpropagation through *unbiased* stochastic gradients that are cheap to compute. More specifically, we can perform a Monte Carlo approximation of Eq. 9 by drawing i.i.d. random samples from $q_{\boldsymbol{\phi}}(\theta)$

$$\mathcal{L}(\boldsymbol{\phi}) \approx \frac{1}{L}\sum_{l=1}^{L} \log p(\mathcal{D}|\theta^{(l)}) + \log p(\theta^{(l)}) - \log q_{\boldsymbol{\phi}}(\theta^{(l)}) \tag{15}$$

$$\theta^{(1)}, \ldots, \theta^{(L)} \underset{\text{i.i.d.}}{\sim} q_{\boldsymbol{\phi}}(\theta).$$



(a) (b)

Figure 2: Illustrative example of a variational inference problem. We are interested in approximating a simple mixture of two Gaussians $p(\theta)$ (grayscale background), with a single Gaussian $q_\phi(\theta)$ (purple contours), that has variational parameters ϕ. As the variational distribution has limited flexibility it only picks up one of the two modes, depending on the initialization of the variational parameters. This illustrates that $q_\phi(\theta)$ usually underestimates the variance of the original distribution $p(\theta)$ as it can completely ignore regions of (sometimes significant) probability mass.

For most continuous distributions $q_\phi(\theta)$ the random sampling can be expressed in terms of auxiliary noise variables ϵ drawn from a parameter free distribution $p(\epsilon)$ and a deterministic transformation $f(\phi, \epsilon)$ with ϕ being the parameters of the distribution. In this case we can rewrite Eq. 15 as

$$\mathcal{L}(\phi) \approx \frac{1}{L}\sum_{l=1}^{L} \log p(\mathcal{D}|\theta^{(l)}) + \log p(\theta^{(l)}) - \log q_\phi(\theta^{(l)}) \tag{16}$$
$$\epsilon^{(1)}, \ldots, \epsilon^{(L)} \underset{\text{i.i.d.}}{\sim} p(\epsilon), \qquad \theta^{(l)} = f(\phi, \epsilon^{(l)}).$$

An example of the transformation $f(\phi, \epsilon)$ would be the case for the location-scale family of distributions (such as the Gaussian). We can sample the "standard" distribution and then apply a simple linear transformation that involves the location, μ, and scale, σ

$$f(\phi, \epsilon) = \mu + \sigma \odot \epsilon, \qquad \phi = (\mu, \sigma), \tag{17}$$

where $\odot$ corresponds to an elementwise product. Therefore, when $f(\phi, \epsilon)$ is differentiable w.r.t. ϕ, we can obtain the following unbiased stochastic gradient of the bound $\mathcal{L}(\phi)$

$$\nabla_\phi \mathcal{L}(\phi) \approx \frac{1}{L}\sum_{l=1}^{L} \nabla_\phi \Big(\log p\left(\mathcal{D}|f(\phi, \epsilon^{(l)})\right) + \log p(f(\phi, \epsilon^{(l)})) - \log q_\phi(f(\phi, \epsilon^{(l)})) \Big), \tag{18}$$

and use it in any off-the-self gradient based optimizer such as Adam [88]. This technique is known as the "reparametrization trick" [91, 151]. Notice that its applicability is not limited to only the variational inference setting; provided that $q_\phi(\theta)$ admits such a reparametrization, we can optimize any arbitrary function $f(\theta)$ as long as it is differentiable w.r.t. θ. This is something that we will exploit in Chapters 7, 8.

1.3 VARIATIONAL BAYESIAN NEURAL NETWORKS

How can we incorporate the aforementioned elements of Bayesian reasoning to neural networks and deep learning? Bayesian neural networks (BNNs) correspond to an instance of this framework; we posit a prior distribution over (some of) the weights of the network, denoted as θ, and then given this prior, we do *inference* in order to obtain a distribution over possible values of those weights given the observed dataset $\mathcal{D}$. This is in contrast to the standard practice in neural networks, where the optimization procedure leads to a *single* set of weights, those at the (local) minimum of the loss. Essentially, we can view Bayesian neural networks as being an *ensemble* of models, since each possible value of the parameters corresponds to a different neural network. Armed with this posterior distribution, we can then make predictions according to the posterior predictive distribution, i.e. Eq. 8, which properly accounts for the uncertainty we have about the parameters and data.

Inference in BNNs is a hard task as the marginal likelihood is intractable to compute. For this reason approximate inference techniques are usually employed, with variational inference (VI) being a popular choice. The application of VI to this setting is straightforward, as we can directly apply the methodology described at the previous section [23]. Let $p(\theta)$ be the prior over the neural network parameters and let $q_{\phi}(\theta)$ be the variational posterior. The variational lower bound can be expressed as

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\theta)} \left[\sum_{i \in \mathcal{D}} \log p(y = y_i | \mathbf{x}_i, \theta) + \log p(\theta) - \log q_{\boldsymbol{\phi}}(\theta) \right], \tag{19}$$

which, for most distributions over continuous random variables θ, can subsequently be maximized using the reparametrization trick [91, 151]. It is interesting to see that for specific choices of $q_{\phi}(\theta)$, $p(\theta)$ and $p(y = y_i|\mathbf{x}_i, \theta)$ we can obtain a "vanilla" neural network. More specifically, in the case of a deterministic $q_{\boldsymbol{\phi}}(\theta) = \delta(\theta - \boldsymbol{\phi})$, and $p(\theta) \propto c$ we have that

$$\mathcal{L}(\boldsymbol{\phi}) \propto \sum_{i \in \mathcal{D}} \log p(y = y_i | \mathbf{x}_i, \theta) + \log c, \tag{20}$$

which will have the same maxima as the objectives provided at Eq. 4, 5 (depending on the task).

After fitting the parameters $\boldsymbol{\phi}$ of the approximate posterior we can then perform predictions for novel inputs $\mathbf{x}^*$ by similarly selecting the most probable class / value of the (approximate) posterior predictive distribution

$$y^* = \arg\max_y \int p(y|\mathbf{x}^*, \theta) p(\theta|\mathcal{D}) d\theta \approx \arg\max_y \int p(y|\mathbf{x}^*, \theta) q_{\phi}(\theta) d\theta, \tag{21}$$

where we substituted the true posterior over θ with its approximation $q_{\boldsymbol{\phi}}(\theta)$. As usually the necessary integral cannot be computed in closed form, we approximate it with Monte Carlo samples from $q_{\boldsymbol{\phi}}(\theta)$

$$y^* \approx \arg\max_y \frac{1}{L} \sum_{l=1}^{L} p(y|\mathbf{x}, \theta^{(l)}), \qquad \theta^{(1)}, \ldots, \theta^{(L)} \underset{\text{i.i.d.}}{\sim} q_{\boldsymbol{\phi}}(\theta). \tag{22}$$

As previously mentioned, it is interesting to see that we are essentially making predictions according to an *infinite* ensemble of models, since each sample from $q_\phi(\theta)$ provides a different value for the weights. This is in contrast to standard neural networks which make predictions according to a *single* model.

Two of the main modelling points in variational BNNs is the choice of the prior distribution over the parameters, $p(\theta)$, as well as the approximate posterior distribution, $q_\phi(\theta)$. In the first, we can encode a-priori assumptions about the behaviour of the parameters of the network. For example, we can posit sparsity inducing priors in order to obtain neural networks that utilize a subset of their parameters [112], which is the topic of chapter 6. In the second, we have to posit a variational approximation that is flexible enough to recover the true posterior distribution. This will be the topic of chapters 3,4. In this way, we can make progress towards approximations that do not underestimate the uncertainty in the predictions, a behaviour that is inherent in variational methods [22]. We provide a visual example of the aforementioned in Figure 2.

1.4 NEURAL NETWORK COMPRESSION

How can we compress neural networks, such that we can make them smaller and faster, while retaining their predictive capabilities? In this section we will explain the two most prominent techniques for neural network compression; pruning and quantization.

1.4.1 *Pruning*

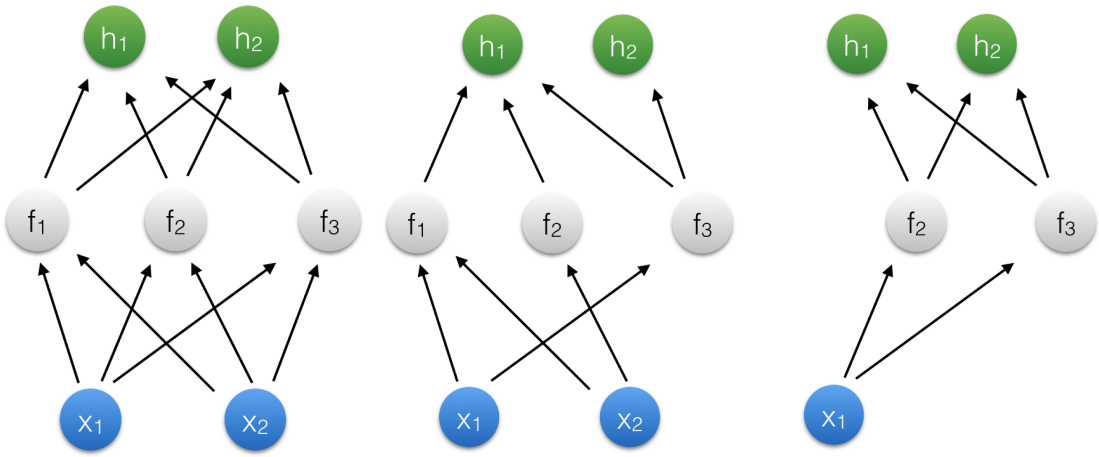
Modern neural network architectures can scale to millions of parameters while requiring large amounts of compute [10]. This has serious implications for real world applications and resource constrained devices; not everyone has access to the appropriate hardware and energy to run such models. Neural network pruning is a framework that can facilitate in improving such costs. Its objective is to "remove" as many parameters as possible from a given network architecture, a fact that can make it both computationally and energy efficient.

Let's consider the example network from the previous section where we have that its output is computed via $(f_L \circ \cdots \circ f_1)(\mathbf{x})$, which each individual function being expressed as $f_i(\mathbf{x}) = \psi(\mathbf{W}_i^\top \mathbf{x} + \mathbf{b}_i)$. Given a set of input-output pairs, the set of parameters $\theta = \{(\mathbf{W}_i, \mathbf{b}_i)_{1:L}\}$ will be subsequently optimized according to the following loss function

$$\mathcal{L}(\theta) = -\log p(\mathbf{y}|\mathbf{X}, \theta) = -\sum_i \log p(y = y_i|\mathbf{x}_i, \theta). \tag{23}$$

How can we encourage the optimization process to "remove" parameters? The traditional way is to include an additional term in the loss function that induces sparsity in the parameters, i.e. it pushes them towards 0. In this way, they will not have an effect on the actual

10 AI & Compute.



(a) Original network (b) Unstructured pruning (c) Group pruning

Figure 3: The difference between **(b)** unstructured, i.e. weight, pruning, and **(c)** structured, i.e. group, pruning. In the first case we remove individual edges from the original network of **(a)**, whereas in the second case we remove entire neurons, which corresponds to removing all of their incoming and outgoing connections. Notice that in **(b)** we still have to compute and store all of the intermediate activations f_1, f_2, f_3, whereas in **(c)** this is not the case as we can easily omit the activations that were pruned, e.g. the f_1.

computation of the prediction and, as a result, they can be successfully omitted. One such sparsity inducing regularizer is the L_1 norm popularized by the Lasso [181]

$$\|\mathbf{W}\|_1 = \sum_{i,j} |w_{ij}|, \tag{24}$$

with the regularized objective defined as

$$\mathcal{L}_{\text{reg}}(\theta) = -\sum_i \log p(y = y_i|\mathbf{x}_i, \theta) + \lambda\|\theta\|_1, \tag{25}$$

with λ being a hyper-parameter that denotes the regularization strength. Such "unstructured" sparsity regularizers aid in reducing the size and memory footprint of a given model. Nevertheless, they do not, practically, reduce the actual computational requirements, as software such as Tensorflow and Pytorch cannot easily ignore multiplications with zero valued weights, thus we still have to compute all of the output neurons $f_i(\mathbf{x})$. Evaluating such neurons is the main bottleneck for computation, especially in the case of convolutional networks [54].

For the latter objective, *group* sparsity is more appropriate. Instead of pushing individual parameters towards zero, group sparsity aims at pushing entire groups of parameters jointly towards zero. In the case of our previous neural network example, we can think of group sparsity as encouraging entire columns of each $\mathbf{W}_i$ to become zero, thus entirely avoiding the computation of the output neuron corresponding to those particular weight vectors. We can view group sparsity as a way to extract an appropriate "sub-architecture" from the

original model that will be more efficient. The extension of the popular Lasso regularizer to the group setting can be written as follows [201]

$$\|\mathbf{W}\|_g = \sum_i \sqrt{\sum_j w_{ij}^2}, \tag{26}$$

where we assumed that each column of the weight matrix constitutes a group. Optimization with such a regularizer can be performed in a similar manner using the following objective function

$$\mathcal{L}_{\text{reg}}(\theta) = -\sum_i \log p(y = y_i|\mathbf{x}_i, \theta) + \lambda\|\theta\|_g. \tag{27}$$

The minimization of the L_1 norm is a well understood convex problem, but it does come with its drawbacks. While pushing insignificant (groups of) parameters towards exactly zero (something which is not possible with e.g. the L_2 norm) it also imposes "shrinkage", i.e. reduces the magnitude, of parameters that could also be important. For this reason, at the chapters 6, 7 of this thesis we will explore alternatives to the Lasso that prevent shrinkage while still encouraging the parameters to be zero.

1.4.2 *Quantization*

Quantization is a procedure in which a collection of values, be it finite or infinite, are mapped to a countable set of values, which is smaller than the original set, by a quantizer function $q(\cdot)$. One of the simplest quantizers is rounding; given a number x, $q(x)$ provides the quantized value via

$$q(x) = \alpha \left\lfloor \frac{x}{\alpha} + \frac{1}{2} \right\rfloor \tag{28}$$

with α being the step size of the quantizer. When $\alpha = 1$ the quantizer $q(x)$ simply pushes the input x towards its nearest integer. The benefits of quantization are two-fold as 1) we can employ it as a tool in order to reduce the size footprint of a given model and 2) we can perform number manipulations according to the image of the quantizer, which is generally faster and more efficient albeit at an accuracy loss.

In this way, quantization can serve as a tool that can aid in speeding up neural network models. Traditional neural networks are implemented using 32-bit floating point arithmetic (FP32). Neural network quantization is usually targeted towards low bit (e.g. 8) fixed-point formats as they can offer practical speedups, memory reduction and energy savings[11]. For maximum efficiency and improvements, quantization is applied to both the weights of the network as well as its activations. This can then allow the deployment of large models to resource constrained devices such as mobile phones and drones.

The two most frequent ways to quantize neural networks is post training quantization and quantization aware training. Post training quantization amounts to estimating an appropriate step-size α for a rounding quantizer $q(\cdot)$ that will be applied to the elements of

11 High-Performance Hardware for Machine Learning.

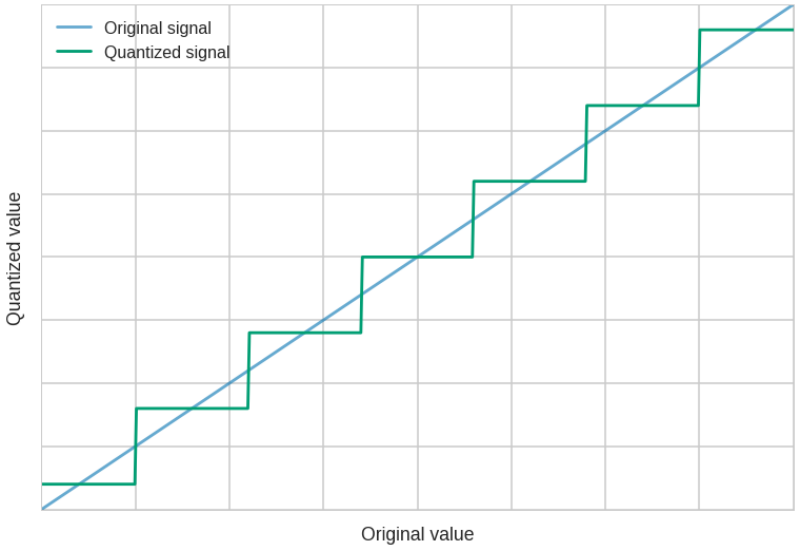


Figure 4: Illustration of a typical quantization function. As we can see, it reminisces a "staircase", a function which has zero gradient almost everywhere besides the change-points where the gradient is infinite. As a result, direct optimization of such functions with the popular gradient based optimizers used for neural networks is infeasible.

the network. Usually a separate α is estimated for each weight tensor $\mathbf{W}_i$, and for each activation tensor $f_i(\cdot)$. It is a conceptually simple approach that for conservative choices of bits (e.g. 8) can lead to significant gains in efficiency without large losses of accuracy.

Quantization aware training is usually employed when we want a more aggressive reduction in the bit-width for the network, e.g. 4 bit or lower. In this case, we "mimic" the quantization procedure on the weights and/or activations, so as to encourage the neural network to be robust to it. More formally, let $\hat{q}(\cdot)$ and $\bar{q}(\cdot)$ be the "fake" quantizers that will be employed during training for the activations and weights respectively. The forward pass of the neural network in this case will be as follows:

$$\hat{\mathbf{h}} = (\hat{q}_L \circ f_L \circ \cdots \circ \hat{q}_1 \circ f_1)(\mathbf{x}), \qquad f_i(\mathbf{x}) = \psi(\bar{q}_i(\mathbf{W}_i)^\top \mathbf{x} + \mathbf{b}_i), \tag{29}$$

where we assumed that the biases were not quantized for simplicity. Maximum likelihood training can then be performed in a similar manner as before, by maximizing

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i \in \mathcal{D}} \sum_j \mathbb{I}[j = y_i] \left(\hat{h}_j - \log \sum_k \exp(\hat{h}_k) \right) \tag{30}$$

for classification or

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i \in \mathcal{D}} -\frac{(\hat{h}_i - y_i)^2}{2} \tag{31}$$

for regression. While conceptually a simple approach, the main bottleneck for this setting is that the quantizers $q(\cdot)$ are not smooth functions, hence gradient based optimization with SGD becomes difficult. At chapter 8 we will describe an approach that can bypass this difficulty by instead optimizing a smooth surrogate objective, essentially ironing the "kinks" of the staircase function illustrated at Figure 4.

1.5 CONTRIBUTIONS

In this section we provide the main contributions of this work, most of which revolve around the two research questions that were stated in section 1.1.1. For the first research question we are interested in an accurate estimate of the uncertainty of a neural network model; for this reason our work will involve the application of Bayesian reasoning. The research findings that we obtain are presented at part i, which is composed from chapters 3, 4 and 5. In the first two we explore variational Bayesian Neural Networks (BNNs) and we describe how we can improve the flexibility of the approximate posterior distribution over the weights of a variational BNN, in an attempt to avoid the pitfall of the underestimated variance in the variational inference setting. More specifically, in chapter 3 we demonstrate how we can posit a *matrix Gaussian*, i.e. a distribution over random matrices, for the weights of each layer of the network. It readily allows for modelling the correlations among the input and output neurons in the network, a fact that leads to improved performance as we experimentally show. In chapter 4 we show how we can improve the posterior approximation even further by employing *multiplicative normalizing flows* over the parameters of the network. They combine auxiliary random variables and normalizing flows [150] in a way that allows for flexible correlations among the weights of each layer while still being computationally tractable.

While variational BNNs are attractive for their flexibility, their uncertainty quality and performance hinges on an appropriate prior over their parameters. Furthermore, inference over the entire parameter space of a neural network is a daunting and challenging task, as it can easily scale to millions of dimensions. For this reason, we conclude part i with chapter 5 where we introduce the *functional neural process* (FNP), a Bayesian model over functions. FNPs can effectively combine the flexibility of neural networks with the modelling properties of a Gaussian Process (GP) [148], a fact that can simplify the inference task and allow us to easily encode inductive biases about the task at hand.

For the second research question we were interested in making neural networks smaller and faster while maintaining their good predictive capabilities. For this reason we focused on the tasks of neural network compression and more specifically, neural network pruning and quantization. The research findings that we obtain are provided at part ii, which is composed from chapters 6, 7 and 8. In chapter 6 we introduce *Bayesian compression* (BC) where we apply the Bayesian machinery that were used in part i in a way that compresses neural networks. More specifically, we show that through the adoption of sparsity inducing priors over the weights of the network, we can prune a large amount of its parameters without hampering performance. Furthermore, we can also use the weight uncertainty in the variational posterior distribution as a proxy for their effective bit-width, which allows for weight quantization and leads to even further compression.

Despite the ability for joint pruning and quantization offered by BC, someone might still require techniques that decouple one from the other in order to allow for more fine grained control. In chapter 7, we propose a technique that allows for L_0 norm regularization in neural networks, the golden standard for sparsity inducing regularizers. Despite its conceptual attractiveness, the L_0 norm is non-differentiable thus not amenable to gradient based optimization, the workhorse of deep neural networks. To this end, we propose

a general purpose smoothing technique through the hard concrete distribution, that can bypass this difficulty and readily allows for L_0 norm regularization through backpropagation. Similarly to pruning, quantization is also a non-differentiable operation. In a similar manner, we propose *relaxed quantization* (RQ) in chapter 8, a general purpose smoothing procedure that can allow for the weights and activations of the network to lie on an a-priori defined quantization grid while allowing for optimization via backpropagation.

Finally, it should be mentioned that these are not the only ways to improve deep learning; deep learning models usually requires large amounts of data [12] in order to be effective, which can have both positive and negative aspects. Given enough data the model can identify the appropriate factors of variation and thus generalize well to novel inputs. Nevertheless, the model might also adopt a-priori biases that are an inherent part of the data. One example where this phenomenon can go wrong was when a deep learning model identified two people of colour as gorillas [13]. While this is not a problem of deep learning itself, it does highlight that one should take extra care in order avoid such outcomes. The practitioner should encode the proper inductive biases to the model, such that it identifies the *causal* factors and their effects.

We have done work during this PhD programme in both of these aspects that is not present in this PhD thesis, so as to preserve a unified theme. In order to tackle unwanted biases that are present in the data, we showed at [111] how we can encode inductive biases in a way that allows the model to successfully find latent representations of the data that are devoid of a given variable. In this way, we can obtain predictive models that are explicitly invariant to said variable, and, in the special case where it is a sensitive variable, we can obtain a model that provides "fair" predictions. Furthermore, we also proposed at [109] the causal effect variational autoencoder (CEVAE), a model that incorporates the causal structure of the problem as an inductive bias and thus can successfully estimate causal effects even in the presence of hidden confounders. Such a model can be highly desirable in e.g. personalized medicine applications as, given some assumptions, can successfully predict the outcomes of treatments to given individuals.

12 Effectiveness of data
13 Deep learning & bias in the data

2

PUBLICATIONS

Below you can find the list of publications that are used in this thesis as well as the contributions of each of the co-authors.

- Christos Louizos and Max Welling, Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.
- Christos Louizos and Max Welling, Multiplicative Normalizing Flows for Variational Bayesian Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.
- Christos Louizos, Karen Ullrich and Max Welling, Bayesian Compression for Deep Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.
- Christos Louizos, Max Welling and Diederik P. Kingma, Learning Sparse Neural Networks through L_0 Regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves and Max Welling, Relaxed Quantization for Discretized Neural Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
- Christos Louizos, Xiahan Shi, Klamer Schutte and Max Welling, The Functional Neural Process. In *Advances in Neural Information Processing Systems (Neurips)*, 2019.

Overall, the majority of work and ideas originated from the first author, with Max Welling contributing important insights and suggestions throughout this research. As for more specific contributions: in the case of the "Bayesian Compression for Deep Learninig" paper, the second author contributed to the experiments and provided insights / ideas around the group sparsity and quantization of the model. The sections of the paper that were originally written by the second author have been rewritten for the purpose of this thesis. Furthermore, there is mutual agreement between the first and second author about using all the figures, tables and algorithms of the original work. For the "Learning Sparse Neural Networks through L_0 regularization" paper the last author contributed important ideas and suggestions. For the "Relaxed Quantization for Discretized Neural Networks"

paper, the second author similarly contributed important ideas and insights around the local grid construction and helped in the experiments and evaluation of the method, whereas the third author provided code for some of the baselines used in the experiments. Finally, for the "Functional Neural Process paper" the second author contributed to the experiments and provided helpful discussions.

Part I

BAYESIAN INFERENCE FOR NEURAL NETWORKS

In this section of the thesis we explore Bayesian inference in the context of neural networks. In the first two chapters we aim to improve the posterior approximation for variational Bayesian neural networks, by either considering linear or non-linear dependencies between the elements of the weight matrices. In the final chapter we adopt a different view and consider Bayesian models in the function space, i.e. stochastic processes, that employ the flexibility of deep neural networks in their construction.

3

STRUCTURED AND EFFICIENT VARIATIONAL DEEP LEARNING WITH MATRIX GAUSSIAN POSTERIORS

In this chapter we introduce a variational Bayesian neural network where the parameters are governed via a probability distribution on random matrices. Specifically, we employ a matrix variate Gaussian [60] parameter posterior distribution where we explicitly model the covariance among the input and output dimensions of each layer. Furthermore, with approximate covariance matrices we can achieve a more efficient way to represent those correlations that is also cheaper than fully factorized parameter posteriors. We further show that with the "local reprarametrization trick" [89] on this posterior distribution we arrive at a Gaussian Process [148] interpretation of the hidden units in each layer and we, similarly with [50], provide connections with deep Gaussian processes. The validity of the proposed approach is verified through extensive experiments. [1]

3.1 INTRODUCTION

While deep learning methods are beating every record in terms of predictive accuracy, they do not yet provide the user with reliable confidence intervals. Yet, for most applications where *decisions* are made based on these predictions, confidence intervals are key. Take the example of an autonomous driving vehicle that enters a new unknown traffic situation: recognizing that predictions become unreliable and handing the steering wheel back to the driver is essential. Similarly, when a physician diagnoses a patient with some ailment and prescribes a drug with potentially severe side effects, it is essential that she/he knows when predictions are unreliable and additional investigation is necessary. These considerations have motivated us to develop a fully Bayesian deep learning framework that is accurate, efficient and delivers reliable confidence intervals.

Furthermore, by being Bayesian we can also harvest another property as a byproduct; natural protection against *overfitting*. Instead of making point estimates for the parameters of the network, which can overfit and provide erroneously certain predictions, we estimate a full posterior distribution over these parameters. Armed with these posterior distributions we can now perform predictions using the posterior predictive distribution, i.e. we can now marginalize over the network parameters and make predictions on the basis of the datapoints alone. As a result we can both obtain the aforementioned confidence intervals

1 This chapter has been adapted from our publication [113].

and better regularize our networks, which is very important in problems where we do not have enough data relative to the amount of features.

Obtaining the parameter posterior distributions for large neural networks is however intractable. To this end, many methods for approximate posterior inference have been devised. Markov Chain Monte Carlo (MCMC) methods are one class of methods that have been explored in this context via Hamiltonian Monte Carlo [134] and stochastic gradient methods [4, 191].

Another family of methods that provide deterministic approximations to the posterior are based on variational inference. These cast inference as an optimization problem and minimize the KL-divergence between the approximate and true posterior. There have been many recent attempts that have adopted this paradigm [23, 56, 72, 89]. However, most of these approaches assume a fully factorized posterior distribution over the neural network weights. We conjecture that this assumption is very restricting as the "true" posterior distribution does have some correlations among the network weights. Therefore by using a fully factorized posterior distribution the learning task becomes "harder" as there is not enough information sharing among the weights.

We therefore introduce a variational Bayesian neural network that instead of treating each element of the weight matrix independently, it treats the weight matrix *as a whole* via a matrix variate Gaussian distribution [60], i.e. a distribution over random matrices. This parametrization can significantly reduce the amount of variance-related parameters that we have to estimate: instead of estimating a separate variance for each weight we can now estimate separate covariances for each row and column of the weight matrix, i.e input and output feature specific covariances. This immediately introduces correlations, and consequently information sharing, among the weights. As a result, it will allow for an easier estimation of the weight posterior uncertainty. In addition, we will also provide a distinct relation between our model and deep (multi-output) Gaussian Processes [40]; this relation arises through the application of the "local reparametrization trick" [89] on the matrix variate Gaussian distribution. We provide an implementation of the proposed approach at `https://github.com/AMLab-Amsterdam/SEVDL_MGP`.

3.2 BEYOND FULLY FACTORIZED PARAMETER POSTERIORS

3.2.1 *Matrix variate Gaussian distribution*

The matrix variate Gaussian [60] is a three parameter distribution that governs a random matrix, e.g. $\mathbf{W}$:

$$p(\mathbf{W}) = \mathcal{MN}(\mathbf{M}, \mathbf{U}, \mathbf{V}) = \frac{\exp\left(-\frac{1}{2}\operatorname{tr}\left[\mathbf{V}^{-1}(\mathbf{W}-\mathbf{M})^T\mathbf{U}^{-1}(\mathbf{W}-\mathbf{M})\right]\right)}{(2\pi)^{np/2}|\mathbf{V}|^{n/2}|\mathbf{U}|^{n/2}} \tag{32}$$

where $\mathbf{M}$ is a $r \times c$ matrix that is the mean of the distribution, $\mathbf{U}$ is a $r \times r$ matrix that provides the covariance of the rows and $\mathbf{V}$ is a $c \times c$ matrix that governs the covariance of

the columns of the matrix. According to [60] this distribution is essentially a multivariate Gaussian distribution where:

$$p(\text{vec}(\mathbf{W})) = \mathcal{N}(\text{vec}(\mathbf{M}), \mathbf{V} \otimes \mathbf{U})$$

where $\text{vec}(\cdot)$ is the vectorization operator (i.e. stacking the columns into a single vector) and $\otimes$ is the Kronecker product. Despite the fact that the matrix variate Gaussian is a simple generalization of the multivariate case it provides us a straightforward way to separate the correlations among the rows and columns of the matrix, which implicitly affects the correlations among the input and output hidden units.

3.2.2 *Variational inference with matrix variate Gaussian posteriors*

For the following we will assume that each input to a layer is augmented with an extra dimension containing $\mathbf{1}$'s so as to account for the biases and thus we are only dealing with weights $\mathbf{W}$ on this expanded input. In order to obtain a matrix variate Gaussian posterior distribution for these weights we can work in a pretty straightforward way: the derivation is similar to [23, 56, 89, 91]. Let $p_\theta(\mathbf{W}), q_\phi(\mathbf{W})$ be a matrix variate Gaussian prior and posterior distribution with parameters θ, ϕ respectively and $(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^N$ be the training data. Then the following lower bound on the marginal log-likelihood can be derived:

$$\begin{aligned}
\log p(\mathbf{Y}|\mathbf{X}) &\geqslant \int q_\phi(\mathbf{W}) \log \frac{p_\theta(\mathbf{W}) p(\mathbf{Y}|\mathbf{X}, \mathbf{W})}{q_\phi(\mathbf{W})} d\mathbf{W} \\
&= \mathbb{E}_{q_\phi(\mathbf{W})}\big[\log p(\mathbf{Y}|\mathbf{X}, \mathbf{W})\big] - \text{KL}(q_\phi(\mathbf{W})||p_\theta(\mathbf{W})) \\
&= \mathcal{L}(\phi; \theta) \qquad (33)
\end{aligned}$$

Following [23, 56, 89] we will refer to $L_{(\mathbf{X},\mathbf{Y})} = \mathbb{E}_{q_\phi(\mathbf{W})}\big[\log p(\mathbf{Y}|\mathbf{X}, \mathbf{W})\big]$ as the *expected log-likelihood* and to $L_c = -\text{KL}(q_\phi(\mathbf{W})||p_\theta(\mathbf{W}))$ as the *complexity loss*. To estimate $L_{(\mathbf{X},\mathbf{Y})}$ we will use simple Monte Carlo integration along with the "reparametrization trick" [91, 151]:

$$\begin{aligned}
&\mathbb{E}_{q_\phi(\mathbf{W})}\big[\log p(\mathbf{Y}|\mathbf{X}, \mathbf{W})\big] = \frac{1}{L} \sum_{i=1}^{L} \log p(\mathbf{Y}|\mathbf{X}, \mathbf{W}^{(i)}) \qquad (34) \\
&\mathbf{W}^{(i)} = \mathbf{M} + \mathbf{U}^{\frac{1}{2}} \mathbf{E}^{(i)} \mathbf{V}^{\frac{1}{2}}, \quad \mathbf{E}^{(i)} \sim \mathcal{MN}(\mathbf{0}, \mathbf{I}, \mathbf{I}) \quad \big(\text{i.e. } E_{ij} \sim \mathcal{N}(0, 1)\big)
\end{aligned}$$

As for the complexity loss L_c; due to the relation with the multivariate Gaussian we can still calculate the KL-divergence between the matrix variate Gaussian prior and posterior efficiently in closed form.

However, maintaining a full covariance over the rows and columns of the weight matrix is both memory and computationally intensive. In order to still have a tractable model we approximate each of the covariances with a diagonal matrix (i.e. independent rows and columns) for simplicity[2]. This approximation provides a per-layer parametrization that re-

2 Note that we could also easily use rank-1 matrices with diagonal corrections [151] and increase the flexibility of our posterior. For example we could apply the rank-1 approximation to the square root of the covariance matrix (as we directly use it for sampling), i.e. $\mathbf{C}^{\frac{1}{2}} = \mathbf{D}_c + \mathbf{u}\mathbf{u}^T$ where $\mathbf{D}_c$ is a diagonal matrix with positive elements.

quires significantly less parameters than a simple fully factorized Gaussian posterior: we have a total of $(n_{in} \times n_{out}) + n_{in} + n_{out}$ parameters, whereas a fully factorized Gaussian posterior has $2(n_{in} \times n_{out})$ per layer. This in turn makes the posterior uncertainty estimation easier as there are fewer parameters to learn.

With this diagonal approximation to the covariance matrices the KL-divergence between the matrix variate Gaussian posterior $q(\mathbf{W}|\mathbf{M}, \sigma_r^2\mathbf{I}, \sigma_c^2\mathbf{I})$ and a standard isotropic matrix variate Gaussian prior $p(\mathbf{W}|\mathbf{o}, \mathbf{I}, \mathbf{I})$ for a matrix of size $r \times c$ corresponds to the following simple expression:

$$\begin{aligned} &\mathrm{KL}(q(\mathbf{W}|\mathbf{M}, \sigma_r^2\mathbf{I}, \sigma_c^2\mathbf{I})\|p(\mathbf{W}|\mathbf{o}, \mathbf{I}, \mathbf{I})) = \\ &\frac{1}{2}\Bigg(\Bigg(\sum_{i=1}^{r}\sigma_{r_i}^2\Bigg)\Bigg(\sum_{j=1}^{c}\sigma_{c_j}^2\Bigg) + \|\mathbf{M}\|_F^2 - rc \\ &\quad - c\Bigg(\sum_{i=1}^{r}\log\sigma_{r_i}^2\Bigg) - r\Bigg(\sum_{j=1}^{c}\log\sigma_{c_j}^2\Bigg)\Bigg) \end{aligned} \tag{35}$$

The derivation for arbitrary covariance matrices is given in the appendix.

3.2.3 *Deep matrix variate Bayesian nets as deep multi-output Gaussian Processes*

Directly using the expected log-likelihood estimator 34 yields increased variance and higher memory requirements, as it was pointed in [89]. Fortunately, similarly to a standard multivariate Gaussian, the inner product between a matrix and a matrix variate Gaussian is again a matrix variate Gaussian [60] and as a result we can use the "local reparametrization trick" [89]. Let $\mathbf{A}_{M\times r}$, with $M \leqslant r$, be a minibatch of M inputs with dimension r that is the input to a network layer; the inner product $\mathbf{B}_{M\times c} = \mathbf{AW}$, where $\mathbf{W}$ is a matrix variate variable with size $r \times c$, has the following distribution:

$$p(\mathbf{B}|\mathbf{A}) = \mathcal{MN}(\mathbf{AM}, \mathbf{AUA}^T, \mathbf{V}) \tag{36}$$

As we can see, after the inner product the inputs $\mathbf{A}$ become *dependent* due to the non-diagonal row covariance $\mathbf{AUA}^T$. Furthermore, the resulting matrix variate Gaussian maintains the same marginalization properties as a multivariate Gaussian. More specifically, if we marginalize out a row from the $\mathbf{B}$ matrix, then the resulting distribution depends only on the remaining inputs, i.e. it corresponds to simply removing that particular input from the minibatch. This fact exposes a Gaussian Process [148] nature for the output $\mathbf{B}$ of each layer.

To make the connection even clearer we can consider an example similar to the one presented in [50]. Let's assume that we have a neural network with one hidden layer and one output layer. Furthermore, let $\mathbf{X}$, with dimensions $N \times D_x$, be the input to the network and $\mathbf{Y}$, with dimensions $N \times D_y$, be the target variable. Finally, let's also assume that for the first weight matrix $p_{\theta_1}(\mathbf{W}_1) = \mathcal{MN}(\mathbf{o}, \mathbf{U}_1^0, \mathbf{V}_1^0)$ and that for the second weight matrix $p_{\theta_2}(\mathbf{W}_2) = \mathcal{MN}(\mathbf{o}, \mathbf{U}_2^0, \mathbf{V}_2^0)$. Now we can define the following generative model:

$$\begin{aligned} &\mathbf{W}_1 \sim \mathcal{MN}(\mathbf{o}, \mathbf{U}_1^0, \mathbf{V}_1^0); \quad \mathbf{W}_2 \sim \mathcal{MN}(\mathbf{o}, \mathbf{U}_2^0, \mathbf{V}_2^0) \\ &\mathbf{B} = \mathbf{XW}_1, \quad \mathbf{F} = \psi(\mathbf{B})\mathbf{W}_2, \quad \mathbf{Y} \sim \mathcal{MN}(\mathbf{F}, \tau^{-1}\mathbf{I}_N, \mathbf{I}_{D_y}) \end{aligned}$$

where $\psi(\cdot)$ is a nonlinearity and $\mathcal{MN}(\mathbf{F}, \tau^{-1}\mathbf{I}_N, \mathbf{I}_{D_y})$ corresponds to an independent multivariate Gaussian over each column of $\mathbf{Y}$. More specifically, let $\mathbf{f}_i$ be a column of $\mathbf{F}$ then $p(\mathbf{Y}|\mathbf{F}) = \prod_{i=1}^{D_y} \mathcal{N}(\mathbf{y}_i|\mathbf{f}_i, \tau^{-1}\mathbf{I}_N)$[3]. Now if we make use of the matrix variate Gaussian property 36 we have that the generative model becomes:

$$\mathbf{B}|\mathbf{X} \sim \mathcal{MN}(\mathbf{o}, \mathbf{X}\mathbf{U}_1^0\mathbf{X}^T, \mathbf{V}_1^0), \quad \mathbf{F}|\mathbf{B} \sim \mathcal{MN}(\mathbf{o}, \psi(\mathbf{B})\mathbf{U}_2^0\psi(\mathbf{B})^T, \mathbf{V}_2^0),$$
$$\mathbf{Y}|\mathbf{F} \sim \mathcal{MN}(\mathbf{F}, \tau^{-1}\mathbf{I}_N, \mathbf{I}_{D_y})$$

or else equivalently:

$$\text{vec}(\mathbf{B})|\mathbf{X} \sim \mathcal{N}(\mathbf{o}, \hat{\mathbf{K}}_{\theta_1}(\mathbf{X}, \mathbf{X})), \quad \text{vec}(\mathbf{F})|\mathbf{B} \sim \mathcal{N}(\mathbf{o}, \hat{\mathbf{K}}_{\theta_2}(\mathbf{B}, \mathbf{B})),$$
$$\text{vec}(\mathbf{Y})|\mathbf{F} \sim \mathcal{N}(\text{vec}(\mathbf{F}), \tau^{-1}(\mathbf{I}_N \otimes \mathbf{I}_{D_y}))$$

where $\hat{\mathbf{K}}_\theta(\mathbf{z}_1, \mathbf{z}_2) = \mathbf{K}_{\text{out}} \otimes \mathbf{K}_{\text{in}}(\mathbf{z}_1, \mathbf{z}_2; \mathbf{U}) = \mathbf{V} \otimes \left(\psi(\mathbf{z}_1)\mathbf{U}\psi(\mathbf{z}_2)^T\right)$[4]. In other words, we have a composition of GPs where the covariance of each GP is governed by a kernel function of a specific form; it is the kroneker product of a global output and an input dependent kernel function, where the latter is composed of fixed dimension nonlinear basis functions (the inputs to each layer) weighted by their covariance. Essentially this kernel provides a distribution for each layer that is similar to a (correlated) multi-output GP, which was previously explored in the context of shallow GPs [24, 25, 200]. Therefore, in order to obtain the marginal likelihood of the targets $\mathbf{Y}$ we have to marginalize over the function values $\mathbf{B}$ and $\mathbf{F}$, which results into a deep GP [40] with the aforementioned kernel function for each GP:

$$\log p(\mathbf{Y}|\mathbf{X}) = \log \mathbb{E}_{p_{\theta_1}(\mathbf{B}|\mathbf{X})p_{\theta_2}(\mathbf{F}|\mathbf{B})}\left[\mathcal{N}(\text{vec}(\mathbf{F}), \tau^{-1}(\mathbf{I}_N \otimes \mathbf{I}_{D_y}))\right]$$

A similar scenario was also considered theoretically in [45]. Now in order to obtain the posterior distribution of the parameters $\mathbf{W}$ we will perform variational inference. We place a matrix variate Gaussian posterior distribution over the weights of the neural network, i.e. $q_{\phi_1}(\mathbf{W}_1)$, $q_{\phi_2}(\mathbf{W}_2)$, as $\mathcal{MN}(\mathbf{M}_1, \mathbf{U}_1, \mathbf{V}_1)$, $\mathcal{MN}(\mathbf{M}_2, \mathbf{U}_2, \mathbf{V}_2)$ respectively, and the marginal likelihood lower bound in eq. 33 becomes:

$$\begin{aligned}\mathcal{L}(\phi_{1,2}, \theta_{1,2}) &= \mathbb{E}_{q_{\phi_1}(\mathbf{W}_1)q_{\phi_2}(\mathbf{W}_2)}\left[\log p(\mathbf{Y}|\mathbf{X}, \mathbf{W}_1, \mathbf{W}_2)\right] - \\ &\qquad \sum_{i=1}^{2} \text{KL}(q_{\phi_i}(\mathbf{W}_i)||p_{\theta_i}(\mathbf{W}_i)) \\ &= L_{(\mathbf{X},\mathbf{Y})}(\phi_1, \phi_2) + \sum_{i=1}^{2} L_c(\phi_i, \theta_i) \end{aligned} \tag{37}$$

3 Note that this is just a simplifying assumption and not a limitation for our method. We could instead also model the correlations among the output variables $\mathbf{Y}$ if we used a full covariance $\mathbf{C}_{D_y}$ instead of $\mathbf{I}_{D_y}$.

4 $\psi(\cdot)$ is the identity function for the input layer.

Noting that $\mathbf{Y}$ only depends on $\mathbf{X}$, $\mathbf{W}_1$, $\mathbf{W}_2$ through $\mathbf{F} = \psi(\mathbf{B})\mathbf{W}_2$ where $\mathbf{B} = \mathbf{X}\mathbf{W}_1$ and applying the reparametrization trick, i.e.

$$\int q_{\phi_1}(\mathbf{W}_1) q_{\phi_2}(\mathbf{W}_2) \log p(\mathbf{Y}|\mathbf{F}(\mathbf{X}, \mathbf{W}_1, \mathbf{W}_2)) d\mathbf{W}_{1,2} = \int \tilde{q}_{\phi_1}(\mathbf{B}|\mathbf{X}) \tilde{q}_{\phi_2}(\mathbf{F}|\mathbf{B}) \log p(\mathbf{Y}|\mathbf{F}) d\mathbf{B} d\mathbf{F}$$

where (using 36),

$$\tilde{q}_{\phi_1}(\mathbf{B}|\mathbf{X}) = \mathcal{N}(\text{vec}(\boldsymbol{\mu}_{\phi_1}(\mathbf{X})), \hat{\mathbf{K}}_{\phi_1}(\mathbf{X}, \mathbf{X})),$$
$$\tilde{q}_{\phi_2}(\mathbf{F}|\mathbf{B}) = \mathcal{N}(\text{vec}(\boldsymbol{\mu}_{\phi_2}(\mathbf{B})), \hat{\mathbf{K}}_{\phi_2}(\mathbf{B}, \mathbf{B}))$$

where $\phi_1 = (\mathbf{M}_1, \mathbf{U}_1, \mathbf{V}_1)$, $\phi_2 = (\mathbf{M}_2, \mathbf{U}_2, \mathbf{V}_2)$ are the variational parameters and $\boldsymbol{\mu}_\phi(\mathbf{z}) = \psi(\mathbf{z})\mathbf{M}$ is the mean function. As we can see, $\tilde{q}_{\phi_1}(\mathbf{B}|\mathbf{X})$, $\tilde{q}_{\phi_2}(\mathbf{F}|\mathbf{B})$ can be considered as approximate posterior GP functions while the local reparametrization trick provides the connection between the primal and dual GP view of the model. The variational objective thus becomes:

$$\mathcal{L}(\phi_{1,2}, \theta_{1,2}) = \mathbb{E}_{\tilde{q}_{\phi_1}(\mathbf{B}|\mathbf{X})\tilde{q}_{\phi_2}(\mathbf{F}|\mathbf{B})}\left[\log p(\mathbf{Y}|\mathbf{F})\right] + \sum_{i=1}^{2} L_c(\phi_i, \theta_i) \tag{38}$$

3.2.4 *Efficient sampling and pseudo-data*

Sampling distribution 36 for every layer is however computationally intensive as we have to calculate the square root of the row covariance $\mathbf{K}_{in}(\mathbf{A}, \mathbf{A}; \mathbf{U}) = \mathbf{A}\mathbf{U}\mathbf{A}^T$ (which has a cubic cost w.r.t. the amount of datapoints in $\mathbf{A}$) every time. A simple solution is to only use its diagonal for sampling. This corresponds to samples from the marginal distribution of each pre-activation latent variable $\mathbf{b}_i$ in the minibatch $\mathbf{A}$. More specifically, we have that $\mathbf{b}_i$ follows a multivariate Gaussian distribution where the covariance is controlled by two sources: the local scalar row variance (i.e. per datapoint feature correlations) and the global column, i.e. pre-activation latent variable (or target variable in the case of the output layer), covariance: $p(\mathbf{b}_i|\mathbf{a}_i) = \mathcal{N}(\mathbf{a}_i\mathbf{M}, \left(\mathbf{a}_i\mathbf{U}\mathbf{a}_i^T\right) \odot \mathbf{V})$.

Despite its simplicity however this approach foregoes the correlations among the elements in the given minibatch. In order to fully utilize this property we adopt an idea from the GP literature: the concept of pseudo-data [169]. More specifically, we introduce pseudo inputs $\tilde{\mathbf{A}}$ and pseudo outputs $\tilde{\mathbf{B}}$ for each layer in the network and sample the distribution of each pre-activation latent variable $\mathbf{b}_i$ conditioned on the pseudo-data:

$$p(\mathbf{b}_i|\mathbf{a}_i, \tilde{\mathbf{A}}, \tilde{\mathbf{B}}) = \mathcal{N}\Big(\mathbf{a}_i\mathbf{M} + \boldsymbol{\sigma}_{12}^T\boldsymbol{\Sigma}_{11}^{-1}\left(\tilde{\mathbf{B}} - \tilde{\mathbf{A}}\mathbf{M}\right), \left(\sigma_{22} - \boldsymbol{\sigma}_{12}^T\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\sigma}_{12}\right) \odot \mathbf{V}\Big) \tag{39}$$

where each of the covariance terms can be estimated as:

$$\boldsymbol{\Sigma}_{11} = \tilde{\mathbf{A}}\mathbf{U}\tilde{\mathbf{A}}^T;\ \boldsymbol{\sigma}_{12} = \tilde{\mathbf{A}}\mathbf{U}\mathbf{a}_i^T;\ \sigma_{22} = \mathbf{a}_i\mathbf{U}\mathbf{a}_i^T \tag{40}$$

As can be seen, the pseudo-data directly affect the distribution of each pre-activation latent variable: if the inputs are similar to the pseudo-inputs then the variance of the latent variable $\mathbf{b}_i$ decreases and the mean is shifted towards the pseudo-data. This allows each layer in the network to be more certain in particular regions of the input space. However, if the inputs are not similar to the pseudo-inputs then the distribution of $\mathbf{b}_i$ depends mostly on the parameters of the underlying matrix variate Gaussian posterior.

It should be noted that the amount of pseudo-data for each layer N_p should be $N_p < D$, where D is the dimensionality of the input, as we are using a linear kernel for the row covariance (that becomes non-linear via the neural network nonlinearities) that has finite rank D. This enforces that the pseudo-data combined with a real input $\mathbf{a}_i$ provide a positive definite kernel $\hat{\mathbf{K}}$ for the joint Gaussian output distribution $p(\tilde{\mathbf{B}}, \mathbf{b}_i|\tilde{\mathbf{A}}, \mathbf{a}_i)$. Furthermore, we also "dampen" $\boldsymbol{\Sigma}_{11}$ by adding to it a small diagonal matrix $\sigma^2\mathbf{I}$ where $\sigma^2 = 1e^{-8}$. This corresponds to assuming "noisy" pseudo-observations $\tilde{\mathbf{B}}$ [148] (where the noise is i.i.d. from $\mathcal{N}(0, \sigma^2)$) which helps avoiding numerical instabilities during optimization (this is particularly helpful with limited precision floating-point).

At first glance it might seem that we now overparametrize each neural network layer, however in practice this does not seem to be the case. From our experience relatively few pseudo-data per layer (compared to the input dimensionality) are necessary for increased performance. This still yields less parameters than fully factorized Gaussian posteriors. In addition, note that with the pseudo data formulation we could also assume that the weight posterior has zero mean $\mathbf{M} = \mathbf{0}$ (in GP parlance this corresponds to removing the mean function); this would reduce the number of parameters even further and still provide a useful model. This assumption leads to sampling the following distribution:

$$p(\mathbf{b}_i|\mathbf{a}_i, \tilde{\mathbf{A}}, \tilde{\mathbf{B}}) = \mathcal{N}\left(\boldsymbol{\sigma}_{12}^T\boldsymbol{\Sigma}_{11}^{-1}\tilde{\mathbf{B}}, \left(\sigma_{22} - \boldsymbol{\sigma}_{12}^T\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\sigma}_{12}\right) \odot \mathbf{V}\right) \tag{41}$$

Finally, since we want a fully Bayesian model, we also place fully factorized multiplicative Gaussian posteriors on both $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ along with log-uniform priors, as it was described in [89]. The final form of the bound 38 with the inclusion of the pseudo-data is:

$$\begin{aligned}\mathcal{L}(\phi_{1,2}, \theta_{1,2}) = \mathbb{E}_{q_{\phi_1}(\mathbf{B},\tilde{\mathbf{A}}_1,\tilde{\mathbf{B}}_1|\mathbf{X})q_{\phi_2}(\mathbf{F},\tilde{\mathbf{A}}_2,\tilde{\mathbf{B}}_2|\mathbf{B})}\left[\log p(\mathbf{Y}|\mathbf{F})\right] \\ + \sum_{i=1}^{2} L_c(\phi_i, \theta_i)\end{aligned} \tag{42}$$

where:

$$\begin{aligned}q_{\phi_i}(\mathbf{B}, \tilde{\mathbf{A}}, \tilde{\mathbf{B}}|\mathbf{X}) &= \tilde{q}_{\phi_i}(\mathbf{B}|\mathbf{X}, \tilde{\mathbf{A}}, \tilde{\mathbf{B}})q_{\phi_i}(\tilde{\mathbf{A}})q_{\phi_i}(\tilde{\mathbf{B}}) \\ L_c(\phi_i, \theta_i) &= -KL(q(\mathbf{W}_i)||p(\mathbf{W}_i)) - KL(q(\tilde{\mathbf{A}}_i)||p(\tilde{\mathbf{A}}_i)) \\ &\quad - KL(q(\tilde{\mathbf{B}}_i)||p(\tilde{\mathbf{B}}_i))\end{aligned}$$

where now ϕ_i, θ_i also include the parameters of the distributions of the pseudo-data. The KL-divergence for these can be found at [89]. We can thus readily optimize the marginal likelihood lower bound of eq. 42 w.r.t. the parameters of the posterior and the pseudo data with stochastic gradient ascent.

3.2.5 *Computational complexity*

A typical variational Bayesian neural network with a fully factorized Gaussian posterior sampled "locally" [89] has asymptotic per-datapoint time complexity $\mathcal{O}(D^2)$ for the mean and variance in each layer, where D is the input/output dimensionality. Our model adds the extra cost of inverting $\boldsymbol{\Sigma}_{11}^{-1}$, that has cubic complexity with respect to the amount of pseudo-data M for each layer. Therefore the asymptotic time complexity is $\mathcal{O}(D^2 + M^3)$ and since usually $M << D$, this does not incur a significantly extra computational cost.

3.2.6 *Bias in the objective*

It should be noted that the objective function that we arrived at eq. 42, is in fact biased. This bias can be better understood by re-writing the distribution of the pre-activations at eq. 41 as

$$p(\mathbf{b}_i|\mathbf{a}_i, \tilde{\mathbf{A}}, \tilde{\mathbf{B}}) = \mathcal{N}(\mathbf{a}_i\mathbf{U}\tilde{\mathbf{A}}^T\boldsymbol{\Sigma}_{11}^{-1}\tilde{\mathbf{B}}, \mathbf{a}_i(\mathbf{U} - \mathbf{U}\tilde{\mathbf{A}}^T\boldsymbol{\Sigma}_{11}^{-1}\tilde{\mathbf{A}}\mathbf{U})\mathbf{a}_i^T \odot \mathbf{V}). \tag{43}$$

We can then observe that eq. 43 corresponds to the local re-parametrization trick, according to a "modified" matrix Gaussian distribution, i.e.

$$q(\hat{\mathbf{W}}|\tilde{\mathbf{A}}, \tilde{\mathbf{B}}) = \mathcal{MN}(\mathbf{U}\tilde{\mathbf{A}}^T\boldsymbol{\Sigma}_{11}^{-1}\tilde{\mathbf{B}}, \mathbf{U} - \mathbf{U}\tilde{\mathbf{A}}^T\boldsymbol{\Sigma}_{11}^{-1}\tilde{\mathbf{A}}\mathbf{U}, \mathbf{V}), \tag{44}$$

$$\mathbf{b}_i = \mathbf{a}_i\hat{\mathbf{W}}, \qquad \hat{\mathbf{W}} \sim q(\hat{\mathbf{W}}|\tilde{\mathbf{A}}, \tilde{\mathbf{B}}), \tag{45}$$

where now we have a distribution over the weight matrix that is conditioned on the pseudo inputs and outputs, $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$. As a result, we should consider $q(\hat{\mathbf{W}}|\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$ as our approximate posterior and thus employ the KL-divergence $KL(q(\hat{\mathbf{W}}|\tilde{\mathbf{A}}, \tilde{\mathbf{B}})||p(\mathbf{W}_i))$, instead of $KL(q(\mathbf{W}_i)||p(\mathbf{W}_i))$. This constitutes the bias in the objective, as now the pseudo input-outputs are not properly regularized by the KL divergence of the weight matrix. Having said that, we empirically observed in the experimental section that despite the aforementioned bias, the overall model can work and produce meaningful results. The extra flexibility from not needing to conform $q(\hat{\mathbf{W}}|\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$ to $p(\mathbf{W}_i)$, allows the pseudo data to better fit the actual data and hence improve predictive performance.

3.3 RELATED WORK

[56] firstly introduced a practical way of variational inference for neural networks. Despite the fact that the proposed (biased) estimator had good performance on a recurrent neural network task, it was not as effective on the regression task of [72]. [23] proposed to use an alternative unbiased estimator that samples on the relatively high variance weight space [89] but nonetheless provided good performance on a reinforcement learning task. The authors of [89] subsequently presented the "local reparametrization trick", which makes use of Gaussian properties so as to sample in the function space, i.e. the hidden units. This provides both reduced memory requirements as well as reduced variance for the expected log-likelihood estimator. However, for their model they still use a fully factorized posterior distribution that doubles the amount of parameters in each layer.

[50] also provides connections between Bayesian neural networks and deep Gaussian processes, but they only consider independent Gaussians for each column of the weight matrix (which in our case correspond to $p(\mathbf{W}) = \mathcal{MN}(\mathbf{M}, \boldsymbol{\sigma}^2\mathbf{I}, \mathbf{I})$) and do not model the variances of the hidden units. Furthermore the approximating variational distribution is quite limited as it corresponds to simple Bernoulli noise and delta approximating distributions for the weight matrix: it is a mixture of two delta peaks for each column of the weight matrix, one at zero and the other at the mean of the Gaussian. This is in contrast to our model where we can explicitly learn the (possibly non-diagonal) covariance for both the input and output dimensions of each layer through the matrix variate Gaussian posterior.

Finally, [72] also assume fully factorized posterior distributions and uses Expectation Propagation [124] instead of variational inference. Closed form approximations bypass the need for sampling in the model, which in turn makes it easier to converge. However their derivation is limited to rectified linear nonlinearities and regression problems, thus limiting the applicability of their model. Furthermore, since each datapoint is treated as new during the update of the parameters, special care has to be given so as to not perform a lot of passes through the dataset since this will in general shrink the variances of the weights of the network.

3.4 EXPERIMENTS

All of the models were coded in Theano [20] and optimization was done with Adam [88], using the default hyper-parameters and temporal averaging. We parametrized the prior for each weight matrix as $p(\mathbf{W}) = \mathcal{MN}(\mathbf{0}, \mathbf{I}, \mathbf{I})$ unless stated otherwise. Following [72] we also divide the input to each layer (both real and pseudo) by the square root of its dimensionality so as to keep the scale of the output (before the nonlinearity) independent of the incoming connections. We used rectified linear units [131] (ReLU) and we initialized the mean of each matrix variate Gaussian via the scheme proposed in [68]. For the initialization of the pseudo-data we sampled the entries of $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ from $\mathcal{U}[-0.01, 0.01]$. We used one posterior sample to estimate the expected log-likelihood before we update the parameters.

We test under two different scenarios: regression and classification. For the regression task we experimented with the UCI [9] datasets that were used in “Probabilistic Backpropagation" (PBP) [72] and in “Dropout as a Bayesian Approximation" [50]. For the classification task we evaluated our model on the permutation invariant MNIST benchmark dataset, so as to compare against other popular neural network models.

Finally we also performed a toy regression experiment on the same artificially generated data as [72], so that we can similarly visualize the predictive distribution that our model provides.

3.4.1 *Regression experiments*

For the regression experiments we followed a similar experimental protocol with [72]: we randomly keep 90% of the dataset for training and use the remaining to test the performance. This process is repeated 20 times (except from the “Protein" dataset where it is per-

formed 5 times and the "Year" dataset where it is performed once) and the average values along with their standard errors are reported at Table 1. Following [72] we also introduce a Gamma prior, $p(\tau) = \mathcal{G}am(a_0 = 6, b_0 = 6)$ and posterior $q(\tau) = \mathcal{G}am(a_1, b_1)$ for the precision of the Gaussian likelihood and we parametrized the matrix variate Gaussian prior for each layer as $p(\mathbf{W}) = \mathcal{MN}(\mathbf{0}, \tau_r^{-1}\mathbf{I}, \tau_c^{-1}\mathbf{I})$, where $p(\tau_r), p(\tau_c) = \mathcal{G}am(a_0 = 1, b_0 = 0.5)$ and $q(\tau_r)q(\tau_c) = \mathcal{G}am(a_r, b_r)\mathcal{G}am(a_c, b_c)$[5]. We optimized a_1, b_1, a_r, b_r, a_c, b_c along with the remaining variational parameters. We do not use a validation set and instead train the networks up until convergence in the training set. We use one hidden layer of 50 units for all of the datasets, except for the larger "Protein" and "Year" datasets where we use 100 units. We normalized the inputs $\mathbf{x}$ of the network to zero mean and unit variance but we did not normalize the targets $\mathbf{y}$. Instead we parametrized the network output as $\mathbf{y} = f(\mathbf{x}) \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y$ where $f(\cdot)$ represents the neural network and $\boldsymbol{\mu}_y, \boldsymbol{\sigma}_y$ are the, per-dimension, mean and standard deviation of the target variable, estimated from the training set. Similarly to [50] we set the upper bound of the variational dropout rate to 0.005, 0.05 and we used 10 pseudo-data pairs for each layer for all of the datasets, except for the smaller "Yacht" dataset where we used 5 and the bigger "Protein" and "Year" where we used 20.

Table 1: Average test set RMSE, predictive log-likelihood and standard errors for the regression datasets. VI, PBP and Dropout correspond to the variational inference method of [56], probabilistic backpropagation [72] and dropout uncertainty [50]. VMG (Variational Matrix Gaussian) corresponds to the proposed model.

	Avg. Test RMSE and Std. Errors				Avg. Test LL and Std. Errors			
Dataset	**VI**	**PBP**	**Dropout**	**VMG**	**VI**	**PBP**	**Dropout**	**VMG**
Boston	4.32±0.29	3.01±0.18	2.97±0.85	**2.70±0.13**	-2.90±0.07	-2.57± 0.09	-2.46±0.25	**-2.46±0.09**
Concrete	7.19±0.12	5.67±0.09	5.23± 0.53	**4.89±0.12**	-3.39±0.02	-3.16±0.02	-3.04±0.09	**-3.01±0.03**
Energy	2.65±0.08	1.80±0.05	1.66±0.19	**0.54±0.02**	-2.39±0.03	-2.04±0.02	-1.99±0.09	**-1.06±0.03**
Kin8nm	0.10±0.00	0.10±0.00	0.10±0.00	**0.08±0.00**	0.90±0.01	0.90±0.01	0.95±0.03	**1.10±0.01**
Naval	0.01±0.00	0.01±0.00	0.01±0.00	**0.00±0.00**	3.73±0.12	3.73±0.01	**3.80±0.05**	2.46±0.00
Pow. Plant	4.33±0.04	4.12±0.03	**4.02±0.18**	4.04±0.04	-2.89±0.01	-2.84±0.01	**-2.80±0.05**	-2.82±0.01
Protein	4.84±0.03	4.73±0.01	4.36±0.04	**4.13±0.02**	-2.99±0.01	-2.97±0.00	-2.89±0.01	**-2.84±0.00**
Wine	0.65±0.01	0.64±0.01	**0.62±0.04**	0.63±0.01	-0.98±0.01	-0.97±0.01	**-0.93±0.06**	-0.95±0.01
Yacht	6.89±0.67	1.02±0.05	1.11±0.38	**0.71±0.05**	-3.43±0.16	-1.63±0.02	-1.55±0.12	**-1.30±0.02**
Year	9.034±NA	8.879±NA	8.849±NA	**8.780±NA**	-3.622±NA	-3.603±NA	**-3.588±NA**	-3.589±NA

As we can see from the results at Table 1 our model overall provides lower root mean square errors, compared to VI [56], PBP [72] and Dropout [50] on most datasets. In addition, we also observe better performance according to the predictive log-likelihoods; our model outperforms VI and PBP on most datasets and is better than Dropout on 6 out of 10. These results empirically verify the effectiveness of our model: with the matrix variate Gaussian posteriors we have a model that is flexible and consequently can both better fit the data, and, in the case of the predictive log-likelihoods, make an accurate estimation of the predictive uncertainty.

5 For this choice of distribution both $\mathbb{E}_{q(\tau_r)q(\tau_c)}[\mathrm{KL}(q(\mathbf{W}|\mathbf{M}, \mathbf{U}, \mathbf{V})||p(\mathbf{W}|\mathbf{0}, \tau_r^{-1}\mathbf{I}, \tau_c^{-1}\mathbf{I}))]$ and the KL-divergence between $q(\tau_r)q(\tau_c)$ and $p(\tau_r)p(\tau_c)$ can be computed in closed form.

3.4.2 *Classification experiments*

For the classification experiments we trained networks with a varying number of layers and hidden units per layer. We used the last 10000 samples of the training set as a validation set for model selection, minibatches of 100 datapoints and set the upper bound for the variational dropout rate to 0.25. We used the same amount of pseudo-data pairs for each layer, but tuned those according to the validation set performance (we set an upper bound of 150 pseudo-data pairs per layer). We did not use any kind of data augmentation or preprocessing. The results can be seen at Table 2.

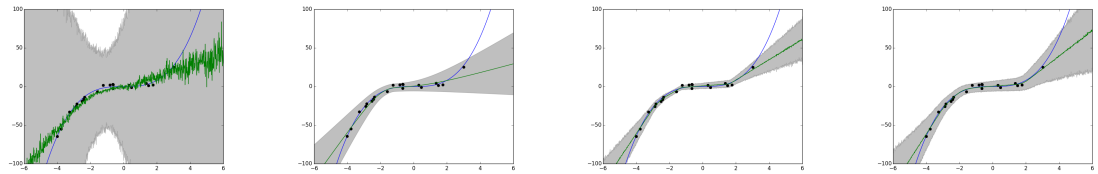
Table 2: Test errors for the permutation invariant MNIST dataset. Bayes B. SM correspond to Bayes by Backprop with the scale mixture prior and the variational dropout results are from the Variational (A) model that doesn't downscale the KL-divergence (so as to keep the comparison fair).

Method	**# layers**	**Test err.**
Max. Likel. [165]	2×800	1.60
Dropout [173]	-	1.25
DropConnect [189]	2×800	1.20
Bayes B. SM [23]	2×400	1.36
	2×800	1.34
	2×1200	1.32
Var. Dropout [89]	3×150	≈ 1.42
	3×250	≈ 1.28
	3×500	≈ 1.18
	3×750	≈ 1.09
VMG	2×400	**1.15**
	3×150	1.18
	3×250	1.11
	3×500	1.08
	3×750	**1.05**

As we can observe our Bayesian neural network performs better than other popular neural networks models for small network sizes. For example, with only three hidden layers of 150 units it achieves 1.18% test error on MNIST, a result that is better than maximum likelihood [165], Dropout [173], DropConnect [189] and Bayes by Backprop [23], where all of the aforementioned methods have bigger architectures than our model. Furthermore, it is also better than a neural network of the same size trained with variational dropout [89].

3.4.3 *Toy experiment*

In order to visually access the quality of the uncertainty that our model provides, we also performed an experiment on the simple toy dataset that was used in [72]. We sampled 20 inputs x from $\mathcal{U}[-4,4]$ and parametrized the target variable as $y_n = x_n^3 + \epsilon_n$ where $\epsilon_n \sim \mathrm{N}(0,9)$. We then fitted a neural network with matrix Gaussian posteriors (with diagonal covariance matrices), a neural network that had a fully factorized Gaussian distribution for the weights and a dropout network. All of the networks had a single hidden layer of 100 units. For our model we used two pseudo-data pairs for the input layer, four for the output layer and set the upper bound of the variational dropout rate to 0.2. The dropout rate for the dropout network was zero for the input layer and 0.2 for the hidden layer. The resulting predictive distributions (after 200 samples) can be seen in figure 5 (with three standard deviations around the mean).



(a) Factorized Gaussian (b) PBP (c) Dropout (d) Matrix Gaussian

Figure 5: Predictive distributions for the toy dataset. Grey areas correspond to ± 3 standard deviations around the mean function.

As we can see the network with matrix Gaussian posteriors provides a realistic predictive distribution that seems slightly better compared to the one obtained from PBP [72]. Interestingly, the simple fully factorized Gaussian (sampled with the “local reparametrization trick") neural network failed to obtain a good fit for the data as it was severely underfitting due to the limited amount of datapoints. This resulted into a very uncertain and noisy predictive distribution that vaguely captured the mean function. This effect is not observed with our model; with the increased flexibility we are able to avoid severe loss in performance. Furthermore, we only have a handful of variance parameters to learn, which consequently provides easier and more robust posterior uncertainty estimation. Finally, it seems that the dropout network provides a predictive distribution that is slightly “overfitted” as the confidence intervals do not diverge as heavily in areas where there are no data.

3.5 CONCLUSIONS

We introduce a scalable variational Bayesian neural network where the parameters are governed by a probability distribution over random matrices: the matrix variate Gaussian. By utilizing properties of this distribution we can see that our model can be considered as a composition of Gaussian Processes with nonlinear kernels of a specific form. This kernel is formed from the kroneker product of two separate kernels; a global output kernel and an

input specific kernel, where the latter is composed from fixed dimension nonlinear basis functions (the inputs to each layer) weighted by their covariance.

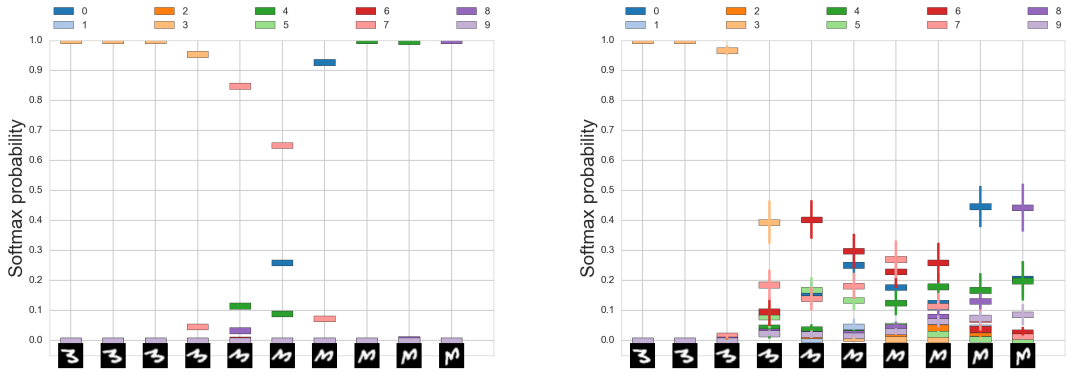
We tested our model in two scenarios: the same regression task as PBP [72] and Dropout uncertainty [50] and the benchmark permutation invariant MNIST classification task. For the regression task we found that our model overall achieves better RMSE and predictive log-likelihoods than VI [56], PBP and Dropout uncertainty. For the classification task we found that our model provides better errors than state of the art methods for small architectures. Finally, we also empirically verified the quality of the predictive distribution that our model provides on the same toy experiment as PBP [72].

4

MULTIPLICATIVE NORMALIZING FLOWS FOR VARIATIONAL BAYESIAN NEURAL NETWORKS

In the previous chapter we showed how to obtain flexible distributions over the neural network weights by considering linear dependencies between the elements of each weight matrix. In this chapter we show how to obtain a more powerful approximation by reinterpreting multiplicative noise in neural networks as auxiliary random variables that augment the approximate posterior. Through this interpretation it is both efficient and straightforward to improve the approximation by employing normalizing flows [150] while still allowing for local reparametrizations [89] and a tractable lower bound [117, 147]. In experiments we show that with this new approximation we can significantly improve upon classical mean field for Bayesian neural networks on both predictive accuracy as well as predictive uncertainty.[1]

4.1 INTRODUCTION



(a) LeNet with weight decay

(b) LeNet with multiplicative formalizing flows

Figure 6: Predictive distribution for a continuously rotated version of a 3 from MNIST. Each colour corresponds to a different class and the height of the bar denotes the probability assigned to that particular class by the network. Visualization inspired by [50].

Neural networks have been the driving force behind the success of deep learning applications. Given enough training data they are able to robustly model input-output relationships

1 This chapter is adapted from our publication [114].

and as a result provide high predictive accuracy. However, they do have some drawbacks. In the absence of enough data they tend to overfit considerably; this restricts them from being applied in scenarios were labeled data are scarce, e.g. in medical applications such as MRI classification. Even more importantly, deep neural networks trained with maximum likelihood or MAP procedures tend to be overconfident and as a result do not provide accurate confidence intervals, particularly for inputs that are far from the training data distribution. A simple example can be seen at Figure 6a; the predictive distribution becomes overly overconfident, i.e. assigns a high softmax probability, towards the wrong class for things it hasn't seen before (e.g. an MNIST 3 rotated by 90 degrees). This in effect makes them unsuitable for applications where decisions are made, e.g. when a doctor determines the disease of a patient based on the output of such a network.

A principled approach to address both of the aforementioned shortcomings is through a Bayesian inference procedure. Under this framework instead of doing a point estimate for the network parameters we infer a posterior distribution. These distributions capture the parameter uncertainty of the network, and by subsequently integrating over them we can obtain better uncertainties about the predictions of the model. We can see that this is indeed the case at Figure 6b; the confidence of the network for the unseen digits is drastically reduced when we are using a Bayesian model, thus resulting into more realistic predictive distributions. Obtaining the posterior distributions is however no easy task, as the nonlinear nature of neural networks makes the problem intractable. For this reason approximations have to be made.

Many works have considered the task of approximate Bayesian inference for neural networks using either Markov Chain Monte Carlo (MCMC) with Hamiltonian Dynamics [134], distilling SGD with Langevin Dynamics [94, 191] or deterministic techniques such as the Laplace Approximation [118], Expectation Propagation [72, 73] and variational inference [23, 50, 56, 89, 113].

In this paper we will also tackle the problem of Bayesian inference in neural networks. We will adopt a stochastic gradient variational inference [91, 151] procedure in order to estimate the posterior distribution over the weight matrices of the network. Arguably one of the most important ingredients of variational inference is the flexibility of the approximate posterior distribution; it determines how well we are able to capture the true posterior distribution and thus the true uncertainty of our models. In Section 4.2 we will introduce *multiplicative normalizing flows* (MNFs) and show how they can produce very flexible distributions in an efficient way by employing auxiliary random variables [3, 117, 147, 157] and normalizing flows [150]. In Section 4.3 we will discuss related work, whereas in Section 4.4 we will evaluate and discuss the proposed framework. Finally we will conclude with Section 4.6, where we will provide some final thoughts along with promising directions for future research. We provide an implementation of MNFs at `https://github.com/AMLab-Amsterdam/MNF_VBNN`.

4.2 MULTIPLICATIVE NORMALIZING FLOWS

4.2.1 *Variational inference for Bayesian Neural Networks*

Let $\mathcal{D}$ be a dataset consisting of input output pairs $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ and let $\mathbf{W}_{1:L}$ denote the weight matrices of L layers. Assuming that $p(\mathbf{W}_i)$, $q_\phi(\mathbf{W}_i)$ are the prior and approximate posterior over the parameters of the i'th layer we can derive the following lower bound on the marginal log-likelihood of the dataset $\mathcal{D}$ using variational Bayes [23, 50, 56, 76, 89, 113, 145]:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{W}_{1:L})}\big[\log p(\mathbf{y}|\mathbf{x}, \mathbf{W}_{1:L}) + \log p(\mathbf{W}_{1:L}) - \log q_\phi(\mathbf{W}_{1:L})\big], \tag{46}$$

where ϕ are the parameters of the variational posterior. For continuous $q(\cdot)$ distributions that allow for the reparametrization trick [91] or stochastic backpropagation [151] we can reparametrize the random sampling from $q(\cdot)$ of the lower bound in terms of noise variables ϵ and deterministic functions $f(\phi, \epsilon)$:

$$\mathcal{L}(\phi) = \mathbb{E}_{p(\epsilon)}\big[\log p(\mathbf{y}|\mathbf{x}, f(\phi, \epsilon)) + \log p(f(\phi, \epsilon)) - \log q_\phi(f(\phi, \epsilon))\big]. \tag{47}$$

This reparametrization allow us to treat approximate parameter posterior inference as a straightforward optimization problem that can be optimized with off-the-shelf (stochastic) gradient ascent techniques.

4.2.2 *Improving the variational approximation*

For Bayesian neural networks the most common family for the approximate posterior is that of mean field with independent Gaussian distributions for each weight. Despite the fact that this leads to a straightforward lower bound for optimization, the approximation capability is quite limiting; it corresponds to just a unimodal "bump" on the very high dimensional space of the parameters of the neural network. There have been attempts to improve upon this approximation with works such as [50] with mixtures of delta peaks and [113] with matrix Gaussians that allow for non-trivial covariances among the weights. Nevertheless, both of the aforementioned methods are still, in a sense, limited; the true parameter posterior is more complex than delta peaks or correlated Gaussians.

There has been a lot of recent work on ways to improve the posterior approximation in latent variable models; normalizing flows [150] and auxiliary random variables [3, 117, 147, 157] have shown that we can improve the flexibility of the approximate posterior in e.g. Deep Latent Gaussian Models [91, 151]. Nevertheless, applying these ideas to the parameters in a neural network has not yet been explored. While it is straightforward to apply normalizing flows to a sample of the weight matrix from $q(\mathbf{W})$, this quickly becomes very expensive; for example with planar flows [150] we will need two extra matrices for each step of the flow. Furthermore, by utilizing this procedure we also lose the benefits of local reparametrizations [89, 113] which are possible with Gaussian approximate posteriors.

In order to simultaneously maintain the benefits of local reparametrizations and increase the flexibility of the approximate posteriors in a Bayesian neural network we will rely on

auxiliary random variables [3, 117, 147, 156, 157]; more specifically we will exploit the well known "multiplicative noise" concept, e.g. as in (Gaussian) Dropout [174], in neural networks and we will parametrize the approximate posterior with the following process:

$$\mathbf{z} \sim q_\phi(\mathbf{z}); \qquad \mathbf{W} \sim q_\phi(\mathbf{W}|\mathbf{z}), \tag{48}$$

where now the approximate posterior becomes a compound distribution, i.e., $q(\mathbf{W}) = \int q(\mathbf{W}, \mathbf{z})\, d\mathbf{z}$, with $\mathbf{z}$ being a vector of random variables distributed according to the mixing density $q(\mathbf{z})$. To allow for local reparametrizations we will parametrize the conditional distribution for the weights to be a fully factorized Gaussian. Therefore we assume the following form for the fully connected layers:

$$q_\phi(\mathbf{W}|\mathbf{z}) = \prod_{i=1}^{D_{in}} \prod_{j=1}^{D_{out}} \mathcal{N}(z_i \mu_{ij}, \sigma_{ij}^2), \tag{49}$$

where D_{in}, D_{out} is the input and output dimensionality, and the following form for the kernels in convolutional networks:

$$q_\phi(\mathbf{W}|\mathbf{z}) = \prod_{i=1}^{D_h} \prod_{j=1}^{D_w} \prod_{k=1}^{D_f} \mathcal{N}(z_k \mu_{ijk}, \sigma_{ijk}^2), \tag{50}$$

where D_h, D_w, D_f are the height, width and number of filters for each kernel. Note that we did not let $\mathbf{z}$ affect the variance of the Gaussian approximation; in a pilot study we found that this parametrization was prone to local optima due to large variance gradients, an effect also observed with the multiplicative parametrization of the Gaussian posterior [89, 130]. We have now reduced the problem of increasing the flexibility of the approximate posterior over the weights $\mathbf{W}$ to that of increasing the flexibility of the mixing density $q(\mathbf{z})$. Since $\mathbf{z}$ is of much lower dimension, compared to $\mathbf{W}$, it is now straightforward to apply normalizing flows to $q(\mathbf{z})$; in this way we can significantly enhance our approximation and allow for e.g. multimodality and nonlinear dependencies between the elements of the weight matrix. This will in turn better capture the properties of the true posterior distribution, thus leading to better performance and predictive uncertainties. We will coin the term *multiplicative normalizing flows* (MNFs) for this family of approximate posteriors. Algorithms 1, 2 describe the forward pass using local reparametrizations for fully connected and convolutional layers with this type of approximate posterior.

For the normalizing flow of $q(\mathbf{z})$ we will use the masked RealNVP [43] using the numerically stable updates introduced in Inverse Autoregressive Flow (IAF) [90]:

$$\begin{aligned} \mathbf{m} &\sim \text{Bern}(0.5), \quad \mathbf{h} = \tanh(f(\mathbf{m} \odot \mathbf{z}_t)), \quad \boldsymbol{\mu} = g(\mathbf{h}), \quad \boldsymbol{\sigma} = \sigma(k(\mathbf{h})) \\ \mathbf{z}_{t+1} &= \mathbf{m} \odot \mathbf{z}_t + (1 - \mathbf{m}) \odot (\mathbf{z}_t \odot \boldsymbol{\sigma} + (1 - \boldsymbol{\sigma}) \odot \boldsymbol{\mu}) \\ \log &\left| \frac{\partial \mathbf{z}_{t+1}}{\partial \mathbf{z}_t} \right| = (1 - \mathbf{m})^T \log \boldsymbol{\sigma}, \end{aligned} \tag{51}$$

where $\odot$ corresponds to element-wise multiplication, $\sigma(\cdot)$ is the sigmoid function[2] and $f(\cdot), g(\cdot), k(\cdot)$ are linear mappings. We resampled the mask $\mathbf{m}$ every time in order to avoid

2 $f(x) = \frac{1}{1+\exp(-x)}$

Algorithm 1 Forward propagation for each fully connected layer h. $\mathbf{M}_w, \mathbf{\Sigma}_w$ are the means and variances of each layer, $\mathbf{H}$ is a minibatch of activations and $\mathrm{NF}(\cdot)$ is the normalizing flow described at eq. 51. For the first layer we have that $\mathbf{H} = \mathbf{X}$ where $\mathbf{X}$ is the minibatch of inputs.

Require: $\mathbf{H}, \mathbf{M}_w, \mathbf{\Sigma}_w$

$\mathbf{Z}_0 \sim q(\mathbf{z}_0)$
$\mathbf{Z}_{T_f} = \mathrm{NF}(\mathbf{Z}_0)$
$\mathbf{M}_h = (\mathbf{H} \odot \mathbf{Z}_{T_f})\mathbf{M}_w$
$\mathbf{V}_h = \mathbf{H}^2 \mathbf{\Sigma}_w$
$\mathbf{E} \sim \mathcal{N}(0, 1)$
return $\mathbf{M}_h + \sqrt{\mathbf{V}_h} \odot \mathbf{E}$

Algorithm 2 Forward propagation for each convolutional layer h. N_f are the number of convolutional filters, $*$ is the convolution operator and we assume the [batch, height, width, feature maps] convention.

Require: $\mathbf{H}, \mathbf{M}_w, \mathbf{\Sigma}_w$

$\mathbf{z}_0 \sim q(\mathbf{z}_0)$
$\mathbf{z}_{T_f} = \mathrm{NF}(\mathbf{z}_0)$
$\mathbf{M}_h = \mathbf{H} * (\mathbf{M}_w \odot \mathrm{reshape}(\mathbf{z}_{T_f}, [1, 1, D_f]))$
$\mathbf{V}_h = \mathbf{H}^2 * \mathbf{\Sigma}_w$
$\mathbf{E} \sim \mathcal{N}(0, 1)$
return $\mathbf{M}_h + \sqrt{\mathbf{V}_h} \odot \mathbf{E}$

a specific splitting over the dimensions of $\mathbf{z}$. For the starting point of the flow $q(\mathbf{z}_0)$ we used a simple fully factorized Gaussian and we will refer to the final iterate as $\mathbf{z}_{T_f}$.

4.2.3 *Bounding the entropy*

Unfortunately, parametrizing the posterior distribution as eq. 48 makes the lower bound intractable as generally we do not have a closed form density function for $q(\mathbf{W})$. This makes the entropy calculation $-\mathbb{E}_{q(\mathbf{W})}[\log q(\mathbf{W})]$ challenging. Fortunately we can make the lower bound tractable again by further lower bounding the entropy in terms of an auxiliary distribution $r(\mathbf{z}|\mathbf{W})$ [3, 117, 147, 156, 157]. This can be seen as if we are performing variational inference on the augmented probability space $p(\mathcal{D}, \mathbf{W}_{1:L}, \mathbf{z}_{1:L})$, that maintains the same true posterior distribution $p(\mathbf{W}|\mathcal{D})$ (as we can always marginalize out $r(\mathbf{z}|\mathbf{W})$ to obtain the original model). The lower bound in this case becomes:

$$\begin{aligned}\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}_{1:L}, \mathbf{W}_{1:L})}\big[&\log p(\mathbf{y}|\mathbf{x}, \mathbf{W}_{1:L}, \mathbf{z}_{1:L}) + \log p(\mathbf{W}_{1:L}) \\ &+ \log r_\theta(\mathbf{z}_{1:L}|\mathbf{W}_{1:L}) - \log q_\phi(\mathbf{W}_{1:L}|\mathbf{z}_{1:L}) - \log q_\phi(\mathbf{z}_{1:L})\big],\end{aligned} \tag{52}$$

where θ are the parameters of the auxiliary distribution $r(\cdot)$. This bound is looser than the previous bound, however the extra flexibility of $q(\mathbf{W})$ can compensate and allow for a tighter bound. Furthermore, the tightness of the bound also depends on the ability of

$r(\mathbf{z}|\mathbf{W})$ to approximate the "auxiliary" posterior distribution $q(\mathbf{z}|\mathbf{W}) = \frac{q(\mathbf{W}|\mathbf{z})q(\mathbf{z})}{q(\mathbf{W})}$. Therefore, to allow for a flexible $r(\mathbf{z}|\mathbf{W})$ we will follow [147] and we will parametrize it with inverse normalizing flows as follows:

$$r(\mathbf{z}_{T_b}|\mathbf{W}) = \prod_{i=1}^{D_z} \mathcal{N}(\tilde{\mu}_i, \tilde{\sigma}_i^2), \tag{53}$$

where for fully connected layers we have that:

$$\tilde{\mu}_i = \big(\mathbf{b}_1 \otimes \tanh(\mathbf{c}^T\mathbf{W})\big)(\mathbf{1} \odot D_{out}^{-1}) \tag{54}$$

$$\tilde{\sigma}_i = \sigma\Big(\big(\mathbf{b}_2 \otimes \tanh(\mathbf{c}^T\mathbf{W})\big)(\mathbf{1} \odot D_{out}^{-1})\Big), \tag{55}$$

and for convolutional:

$$\tilde{\mu}_i = \big(\tanh(\text{mat}(\mathbf{W})\mathbf{c}) \otimes \mathbf{b}_1\big)(\mathbf{1} \odot (D_h D_w)^{-1}) \tag{56}$$

$$\tilde{\sigma}_i = \sigma\Big(\big(\tanh(\text{mat}(\mathbf{W})\mathbf{c}) \otimes \mathbf{b}_2\big)(\mathbf{1} \odot (D_h D_w)^{-1})\Big), \tag{57}$$

where $\mathbf{b}_1, \mathbf{b}_2, \mathbf{c}$ are trainable vectors that have the same dimensionality as $\mathbf{z}$, D_z, $\mathbf{1}$ corresponds to a vector of 1s, $\otimes$ corresponds to the outer product and $\text{mat}(\cdot)$ corresponds to the matricization[3] operator. The $\mathbf{z}_{T_b}$ variable corresponds to the fully factorized variable that is transformed by a normalizing flow to $\mathbf{z}_{T_f}$ or else the variable obtained by the inverse normalizing flow, $\mathbf{z}_{T_b} = \text{NF}^{-1}(\mathbf{z}_{T_f})$. We will parametrize this inverse directly with the procedure described at eq. 51. Notice that we can employ local reparametrizations also in eq. 54,55,56,57, so as to avoid sampling the, potentially big, matrix $\mathbf{W}$. With the standard normal prior and the fully factorized Gaussian posterior of eq. 49 the KL-divergence between the prior and the posterior can be computed as follows:

$$\begin{aligned} -\text{KL}(q(\mathbf{W},\mathbf{z})||p(\mathbf{W},\mathbf{z})) = \mathbb{E}_{q(\mathbf{z}_{T_f})}[-\text{KL}(q(\mathbf{W}|\mathbf{z}_{T_f})||p(\mathbf{W}))] + \\ \mathbb{E}_{q(\mathbf{W},\mathbf{z}_{T_f})}[\log r(\mathbf{z}_{T_f}|\mathbf{W}) - \log q(\mathbf{z}_{T_f})], \end{aligned} \tag{58}$$

where each of the terms corresponds to:

$$\text{KL}(q(\mathbf{W}|\mathbf{z}_{T_f})||p(\mathbf{W})) = \frac{1}{2}\sum_{i,j}(-\log\sigma_{i,j}^2 + \sigma_{i,j}^2 + z_{T_{f_i}}^2\mu_{i,j}^2 - 1) \tag{59}$$

$$\log r(\mathbf{z}_{T_f}|\mathbf{W}) = \log r(\mathbf{z}_{T_b}|\mathbf{W}) + \sum_{t=T_f}^{T_f+T_b} \log\left|\frac{\partial \mathbf{z}_{t+1}}{\partial \mathbf{z}_t}\right| \tag{60}$$

$$\log q(\mathbf{z}_{T_f}) = \log q(\mathbf{z}_0) - \sum_{t=1}^{T_f} \log\left|\frac{\partial \mathbf{z}_{t+1}}{\partial \mathbf{z}_t}\right|. \tag{61}$$

We also found that in general it is beneficial to "constrain" the standard deviations σ_{ij} of the conditional Gaussian posterior $q(\mathbf{W}|\mathbf{z})$ during the forward pass for the computation

3 Converting the multidimensional tensor to a matrix.

of the likelihood to a lower than the true range, e.g. $[0, \alpha]$ instead of the $[0, 1]$ we have with a standard normal prior. This results into a small bias and a looser lower bound, however it helps in avoiding bad local minima in the variational objective. This is akin to the free bits objective described at [90].

4.3 RELATED WORK

Approximate inference for Bayesian neural networks has been pioneered by [118] and [134]. Laplace approximation [118] provides a deterministic approximation to the posterior that is easy to obtain; it is a Gaussian centered at the MAP estimate of the parameters with a covariance determined by the inverse of the Hessian of the log-likelihood. Despite the fact that it is straightforward to implement, its scalability is limited unless approximations are made, which generally reduces performance. Hamiltonian Monte Carlo [134] is so far the golden standard for approximate Bayesian inference; nevertheless it is also not scalable to large networks and datasets due to the fact that we have to explicitly store the samples from the posterior. Furthermore as it is an MCMC method, assessing convergence is non trivial. Nevertheless there is interesting work that tries to improve upon those issues with stochastic gradient MCMC [30] and distillation methods [94].

Deterministic methods for approximate inference in Bayesian neural networks have recently attained much attention. One of the first applications of variational inference in neural networks was in [145] and [76]. More recently [56] proposed a practical method for variational inference in this setting with a simple (but biased) estimator for a fully factorized posterior distribution. [23] improved upon this work with an unbiased estimator and a scale mixture prior. [72] proposed to use Expectation Propagation [124] with fully factorized posteriors and showed good results on regression tasks. [89] showed how Gaussian dropout can be interpreted as performing approximate inference with log-uniform priors, multiplicative Gaussian posteriors and local reparametrizations, thus allowing straightforward learning of the dropout rates. Similarly [50] showed interesting connections between binary Dropout [174] networks and approximate Bayesian inference in deep Gaussian Processes [40] thus allowing the extraction of uncertainties in a principled way. Similarly [113] arrived at the same result through structured posterior approximations via matrix Gaussians and local reparametrizations [89].

It should also be mentioned that uncertainty estimation in neural networks can also be performed without the Bayesian paradigm; frequentist methods such as Bootstrap [140] and ensembles [97] have shown that in certain scenarios they can provide reasonable confidence intervals.

4.4 EXPERIMENTS

All of the experiments were coded in Tensorflow [1] and optimization was done with Adam [88] using the default hyper-parameters. We used the LeNet 5[4] [99] convolutional architecture with ReLU [131] nonlinearities. The means $\mathbf{M}$ of the conditional Gaussian

4 The version from Caffe.

$q(\mathbf{W}|\mathbf{z})$ were initialized with the scheme proposed in [68], whereas the log of the variances were initialized by sampling from $\mathcal{N}(-9, 0.001)$. Unless explicitly mentioned otherwise we use flows of length two for $q(\mathbf{z})$ and $r(\mathbf{z}|\mathbf{W})$ with 50 hidden units for each step of the flow of $q(\mathbf{z})$ and 100 hidden units for each step of the flow of $r(\mathbf{z}|\mathbf{W})$. We used 100 posterior samples to estimate the predictive distribution for all of the models during testing and 1 posterior sample during training.

Table 3: Models considered in this paper. Dropout corresponds to the model used in [49], Deep Ensemble to the model used in [97], FFG to the Bayesian neural network employed in [23], FFLU to the Bayesian neural network used in [89, 130] with the additive parametrization of [130] and MNFG corresponds to the proposed variational approximation. It should be noted that Deep Ensembles use adversarial training [55].

Name	Prior	Posterior
L2	$\mathcal{N}(\mathbf{o}, \mathbf{I})$	delta peak
Dropout	$\mathcal{N}(\mathbf{o}, \mathbf{I})$	mixture of zero and delta peaks
D. Ensem.	-	mixture of peaks
FFG	$\mathcal{N}(\mathbf{o}, \mathbf{I})$	fully factorized additive Gaussian
FFLU	$\log(\lvert\mathbf{W}\rvert) = c$	fully factorized additive Gaussian
MNFG	$\mathcal{N}(\mathbf{o}, \mathbf{I})$	multiplicative normalizing flows

4.4.1 *Predictive performance and uncertainty*

MNIST We trained on MNIST LeNet architectures using the priors and posteriors described at Table 3. We trained Dropout with the way described at [49] using 0.5 for the dropout rate and for Deep Ensembles [97] we used 10 members and $\epsilon = .25$ for the adversarial example generation. For the models with the Gaussian prior we constrained the standard deviation of the conditional posterior to be $\leqslant .5$ during the forward pass. The classification performance of each model can be seen at Table 4; while our overall focus is not classification accuracy per se, we see that with the MNF posteriors we improve upon mean field reaching similar accuracies with Deep Ensembles.

NOTMNIST To evaluate the predictive uncertainties of each model we performed the task described at [97]; we estimated the entropy of the predictive distributions on notMNIST[5] from the LeNet architectures trained on MNIST. Since we a-priori know that none of the notMNIST classes correspond to a trained class (since they are letters and not digits) the ideal predictive distribution is uniform over the MNIST digits, i.e. a maximum entropy distribution. Contrary to [97] we do not plot the histogram of the entropies across the images but we instead use the empirical CDF,which we think is more informative. Curves that are closer to the bottom right part of the plot are preferable, as it denotes that the prob-

5 Can be found at http://yaroslavvb.blogspot.co.uk/2011/09/notmnist-dataset.html

ability of observing a high confidence prediction is low. At Figure 7 we show the empirical CDF over the range of possible entropies, $[0, 2.5]$, for all of the models.

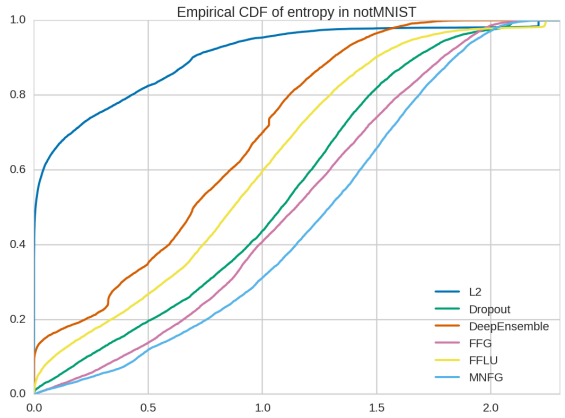


Figure 7: Empirical CDF for the entropy of the predictive distributions on notMNIST.

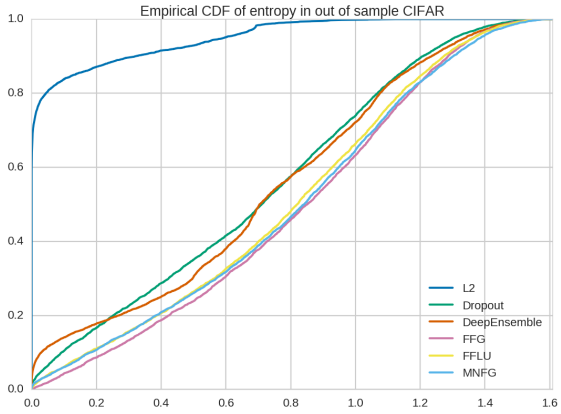


Figure 8: Empirical CDF for the entropy of the predictive distributions on the 5 hidden classes from CIFAR 10.

It is clear from the plot that the uncertainty estimates from MNFs are better than the other approaches, since the probability of a low entropy prediction is overall lower. The network trained with just weight decay was, as expected, the most overconfident with an almost zero median entropy while Dropout seems to be in the middle ground. The Bayesian neural net with the log-uniform prior also showed overconfidence in this task; we hypothesize that this is due to the induced sparsity [130] which results into the pruning of almost all irrelevant sources of variation in the parameters thus not providing enough variability to allow for uncertainty in the predictions. The sparsity levels[6] are 62%, 95.2% for the two convolutional layers and 99.5%, 93.3% for the two fully connected. Similar effects would probably be also observed if we optimized the dropout rates for Dropout. The only source of randomness in the neural network is from the Bernoulli random variables (r.v.) z. By employing the Central Limit Theorem[7] we can express the distribution of the activations as a Gaussian [190] with variance affected by the variance of the Bernoulli r.v., $\mathbb{V}(z) = \pi(1-\pi)$. The maximum variance of the Bernoulli r.v. is when $\pi = 0.5$, therefore any tuning of the Dropout rate will result into a decrease in the variance of the r.v. and therefore a decrease in the variance of the Gaussian at the hidden units. This will subsequently lead into less predictive variance and more confidence.

Finally, whereas it was shown at [97] that Deep Ensembles provide good uncertainty estimates (better than Dropout) on this task using fully connected networks, this result did not seem to apply for the LeNet architecture we considered. We hypothesize that they are sensitive to the hyperparameters (e.g. adversarial noise, number of members in the ensemble) and it requires more tuning in order to improve upon Dropout on this architecture.

CIFAR 10 We performed a similar experiment on CIFAR 10. To artificially create the "unobserved class" scenario, we hid 5 of the labels (dog, frog, horse, ship, truck) and

6 Computed by pruning weights where $\log \sigma^2 - \log \mu^2 \geqslant 5$ [130].
7 Assuming that the network is wide enough.

trained on the rest (airplane, automobile, bird, cat, deer). For this task we used the larger LeNet architecture[8] described at [49]. For the models with the Gaussian prior we similarly constrained the standard deviation during the forward pass to be $\leqslant .4$. For Deep Ensembles we used five members with $\epsilon = .1$ for the adversarial example generation. The predictive performance on these five classes can be seen in Table 4, with Dropout and MNFs achieving the overall better accuracies. We subsequently measured the entropy of the predictive distribution on the classes that were hidden, with the resulting empirical CDFs visualized in Figure 8.

We similarly observe that the network with just weight decay was the most overconfident. Furthermore, Deep Ensembles and Dropout had similar uncertainties, with Deep Ensembles having lower accuracy on the observed classes. The networks with the Gaussian priors also had similar uncertainty with the network with the log uniform prior, nevertheless the MNF posterior had much better accuracy on the observed classes. The sparsity levels for the network with the log-uniform prior now were 94.9%, 99.8% for the convolutional layers and 99.9%, 92.7% for the fully connected. Overall, the network with the MNF posteriors seem to provide the better trade-off in uncertainty and accuracy on the observed classes.

Table 4: Test errors (%) with the LeNet architecture on MNIST and the first five classes of CIFAR 10.

Dataset	**L2**	**Dropout**	**D.Ensem.**	**FFG**	**FFLU**	**MNFG**
MNIST	0.6	0.5	0.7	0.9	0.9	0.7
CIFAR 5	24	16	21	22	23	16

4.4.2 *Accuracy and uncertainty on adversarial examples*

We also measure how robust our models and uncertainties are against adversarial examples [55, 179] by generating examples using the fast sign method [55] for each of the previously trained architectures using Cleverhans [143]. For this task we do not include Deep Ensembles as they are trained on adversarial examples.

MNIST On this scenario we observe interesting results if we plot the change in accuracy and entropy by varying the magnitude of the adversarial perturbation. The resulting plot can be seen in Figure 9. Overall Dropout seems to have better accuracies on adversarial examples; nevertheless, those come at an "overconfident" price since the entropy of the predictive distributions is quite low thus resulting into predictions that have, on average, above 0.7 probability for the dominant class. This is in contrast with MNFs; while the accuracy almost immediately drops close to random, the uncertainty simultaneously increases to almost maximum entropy. This implies that the predictive distribution is more or less uniform over those examples. So despite the fact that our model cannot overcome adversarial examples at least it “knows that it doesn’t know".

8 192 filters at each convolutional layer and 1000 hidden units for the fully connected layer.

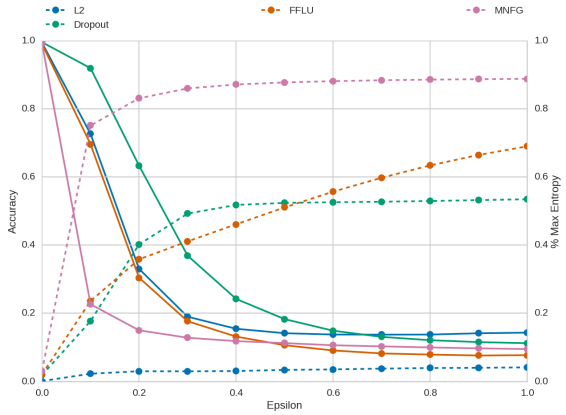


Figure 9: Accuracy (solid) vs entropy (dashed) as a function of the adversarial perturbation ϵ on MNIST.

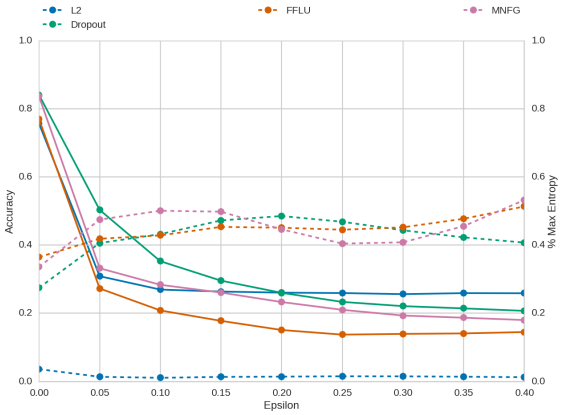
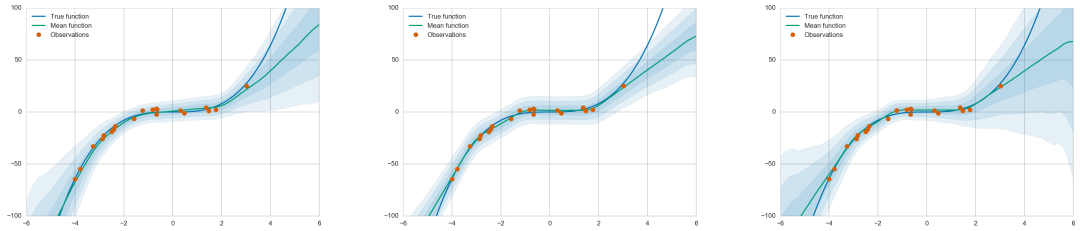


Figure 10: Accuracy (solid) vs entropy (dashed) as a function of the adversarial perturbation ϵ on CIFAR 10 (on the first 5 classes).

CIFAR We performed the same experiment also on the five class subset of CIFAR 10. The results can be seen in Figure 10. Here we however observe a different picture, compared to MNIST, since all of the methods experienced overconfidence. We hypothesize that adversarial examples are harder to escape and be uncertain about in this dataset, due to the higher dimensionality, and therefore further investigation is needed.



(a) Dropout $\pi = 0.5$ (b) FFLU (c) MNFG

Figure 11: Predictive distributions for the toy dataset. Blue areas correspond to ± 3 standard deviations around the mean.

4.4.3 *Regression on toy dataset*

For the final experiment we visualize the predictive distributions obtained with the different models on the toy regression task introduced at [72]. We generated 20 training inputs from $\mathcal{U}[-4, 4]$ and then obtained the corresponding targets via $y = x^3 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 9)$. We fixed the likelihood noise to its true value and then fitted a Dropout network with $\pi = 0.5$ for the hidden layer[9], an FFLU network and an MNFG. The resulting predictive distributions can be seen at Figure 11.

As we can observe, MNF posteriors provide more realistic predictive distributions, closer to the true posterior (which can be seen at [72]) and with the network being more un-

9 No Dropout was used for the input layer since it is 1-dimensional.

certain on areas where we do not observed any data. The uncertainties obtained by Dropout with fixed $\pi = 0.5$ did not diverge as much in those areas but overall they were better compared to the uncertainties obtained with FFLU. We could probably attribute the latter to the sparsification of the network since 95% and 44% of the parameters were pruned for each layer respectively. In general, we see that MNFs are flexible enough to allow for optimizing all of their parameters in a way that does better approximate the true posterior distribution.

4.5 MEMORIZATION CAPABILITIES

As it was shown in [203], deep neural networks can exhibit memorization, even with random labels. Therefore deep neural networks could perfectly fit the training data while having random chance accuracy on the test data. It was further shown that performing regularization such as Dropout or weight decay did not alleviate this phenomenon. [130] instead showed that by employing Sparse Variational Dropout this phenomenon did not appear, thus resulting into the network pruning everything and having random chance accuracy on both training and test sets. We similarly show here that with Gaussian priors and MNF posteriors we also have random chance accuracy on both train and test sets. This suggests that it is Bayesian inference that penalizes memorization.

Table 5: Accuracy (%) with the LeNet architecture on MNIST and the first five classes of CIFAR 10 using random labels. Random chance is 11% on MNIST and 20% on CIFAR 5.

Dataset	**Dropout** train	**Dropout** test	**MNFG** train	**MNFG** test
MNIST	30	11	11	11
CIFAR 5	89	20	20	20

4.6 CONCLUSION

We introduce multiplicative normalizing flows (MNFs); a family of approximate posteriors for the parameters of a variational Bayesian neural network. We have shown that through this approximation we can significantly improve upon mean field on both predictive performance as well as predictive uncertainty. We compared our uncertainty on notMNIST and CIFAR with Dropout [50, 174] and Deep Ensembles [97] using convolutional architectures and found that MNFs achieve more realistic uncertainties while providing predictive capabilities on par with Dropout. We suspect that the predictive capabilities of MNFs can be further improved through more appropriate optimizers that avoid the bad local minima in the variational objective.

There are a couple of promising directions for future research. One avenue would be to explore how much can MNFs sparsify and compress neural networks under either sparsity inducing priors, such as the log-uniform prior [89, 130], or empirical priors [185]. Another promising direction is that of designing better priors for Bayesian neural networks. For example [134] has identified limitations of Gaussian priors and proposes alternative

priors such as the Cauchy. Furthermore, the prior over the parameters also affects the type of uncertainty we get in our predictions; for instance we observed in our experiments a significant difference in uncertainty between Gaussian and log-uniform priors. Since different problems require different types of uncertainty it makes sense to choose the prior accordingly, e.g. use an informative prior so as to alleviate adversarial examples.

5

THE FUNCTIONAL NEURAL PROCESS

In the previous chapters we considered the Bayesian inference setting when we posit priors and approximate posteriors over the weights of the network. In this chapter we take a different view and we present a new family of exchangeable stochastic processes, the Functional Neural Processes (FNPs). FNPs model distributions over functions by learning a graph of dependencies on top of latent representations of the points in the given dataset. In doing so, they define a Bayesian model without explicitly positing a prior distribution over latent global parameters; they instead adopt priors over the relational structure of the given dataset, a task that is much simpler. We show how we can learn such models from data, demonstrate that they are scalable to large datasets through mini-batch optimization and describe how we can make predictions for new points via their posterior predictive distribution. We experimentally evaluate FNPs on the tasks of toy regression and image classification and show that, when compared to baselines that employ global latent parameters, they offer both competitive predictions as well as more robust uncertainty estimates. [1]

5.1 INTRODUCTION

Neural networks are a prevalent paradigm for approximating functions of almost any kind. Their highly flexible parametric form coupled with large amounts of data allows for accurate modelling of the underlying task, a fact that usually leads to state of the art prediction performance. While predictive performance is definitely an important aspect, in a lot of safety critical applications, such as self-driving cars, we also require accurate uncertainty estimates about the predictions.

Bayesian neural networks [23, 56, 119, 134] have been an attempt at imbuing neural networks with the ability to model uncertainty; they posit a prior distribution over the weights of the network and through inference they can represent their uncertainty in the posterior distribution. Nevertheless, for such complex models, the choice of the prior is quite difficult since understanding the interactions of the parameters with the data is a non-trivial task. As a result, priors are usually employed for computational convenience and tractability. Furthermore, inference over the weights of a neural network can be a daunting task due to the high dimensionality and posterior complexity [114, 163].

1 This chapter is adapted from our publication [110].

An alternative way that can "bypass" the aforementioned issues is that of adopting a stochastic process [92]. They posit distributions over functions, e.g. neural networks, directly, without the necessity of adopting prior distributions over global parameters, such as the neural network weights. Gaussian processes [148] (GPs) is a prime example of a stochastic process; they can encode any inductive bias in the form of a covariance structure among the datapoints in the given dataset, a more intuitive modelling task than positing priors over weights. Furthermore, for vanilla GPs, posterior inference is much simpler. Despite these advantages, they also have two main limitations: 1) the underlying model is not very flexible for high dimensional problems and 2) training and inference is quite costly since it generally scales cubically with the size of the dataset.

Given the aforementioned limitations of GPs, one might seek an alternative way to parametrize stochastic processes that can bypass these issues. To this end, we present our main contribution, *Functional Neural Processes* (FNPs), a family of exchangeable stochastic processes that posit distributions over functions in a way that combines the properties of neural networks and stochastic processes. We show that, in contrast to prior literature such as Neural Processes (NPs) [52], FNPs do not require explicit global latent variables in their construction, but they rather operate by building a graph of dependencies among local latent variables, reminiscing more of autoencoder type of latent variable models [91, 151]. We further show that we can exploit the local latent variable structure in a way that allows us to easily encode inductive biases and illustrate one particular instance of this ability by designing an FNP model that behaves similarly to a GP with an RBF kernel. Furthermore, we demonstrate that FNPs are scalable to large datasets, as they can facilitate for minibatch gradient optimization of their parameters, and have a simple to evaluate and sample posterior predictive distribution. Finally, we evaluate FNPs on toy regression and image classification tasks and show that they can obtain competitive performance and more robust uncertainty estimates. We have open sourced an implementation of FNPs for both classification and regression along with example usages at https://github.com/AMLab-Amsterdam/FNP.

5.2 THE FUNCTIONAL NEURAL PROCESS

For the following we assume that we are operating in the supervised learning setup, where we are given tuples of points $(\mathbf{x}, \mathbf{y})$, with $\mathbf{x} \in \mathcal{X}$ being the input covariates and $\mathbf{y} \in \mathcal{Y}$ being the given label. Let $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1) \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$ be a sequence of N observed datapoints. We are interested in constructing a stochastic process that can bypass the limitations of GPs and can offer the predictive capabilities of neural networks. There are two necessary conditions that have to be satisfied during the construction of such a model: exchangeability and consistency [92]. An exchangeable distribution over $\mathcal{D}$ is a joint probability over these elements that is invariant to permutations of these points, i.e.

$$p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) = p(\mathbf{y}_{\sigma(1:N)}|\mathbf{x}_{\sigma(1:N)}), \tag{62}$$

where $\sigma(\cdot)$ corresponds to the permutation function. Consistency refers to the phenomenon that the probability defined on an observed sequence of points $\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$,

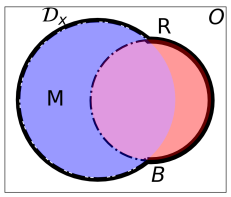


Figure 12: Venn diagram of the sets used in this work. The blue is the training inputs $\mathcal{D}_x$, the red is the reference set R and the parts enclosed in the dashed and solid lines are M, the training points not in R, and B, the union of the training points and R. The white background corresponds to O, the complement of R.

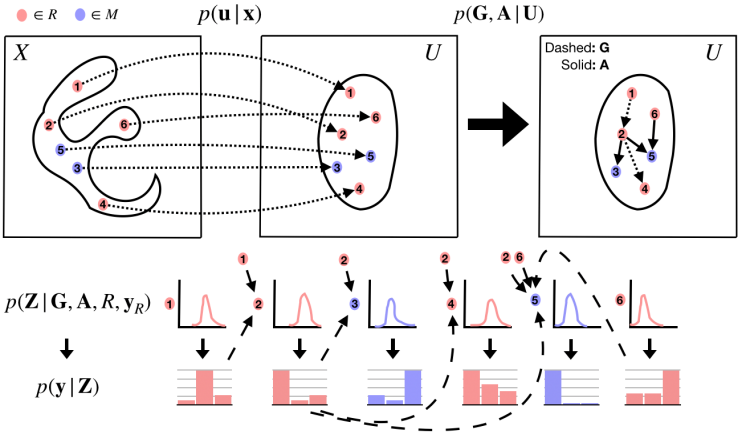


Figure 13: The Functional Neural Process (FNP) model. We embed the inputs (dots) from a complicated domain $\mathcal{X}$ to a simpler domain $\mathcal{U}$ where we then sample directed graphs of dependencies among them, $\mathbf{G}, \mathbf{A}$. Conditioned on those graphs, we use the parents from the reference set R as well as their labels $\mathbf{y}_R$ to parameterize a latent variable $\mathbf{z}_i$ that is used to predict the target y_i. Each of the points has a specific number id for clarity.

$p_n(\cdot)$, is the same as the one defined on an extended sequence $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n+m}, y_{n+m})\}$, $p_{n+m}(\cdot)$, when we marginalize over the new points:

$$p_n(y_{1:n}|\mathbf{x}_{1:n}) = \int p_{n+m}(y_{1:n+m}|\mathbf{x}_{1:n+m})dy_{n+1:n+m}. \tag{63}$$

Ensuring that both of these conditions hold, allows us to invoke the Kolmogorov Extension and de-Finneti's theorems [92], hence prove that the model we defined is an exchangeable stochastic process. In this way we can guarantee that there is an underlying Bayesian model with an implied prior over global latent parameters $p_\theta(\mathbf{w})$ such that we can express the joint distribution in a conditional i.i.d. fashion, i.e., $p_\theta(y_1, \ldots, y_N|\mathbf{x}_1, \ldots, \mathbf{x}_N)$ is expressed as $\int p_\theta(\mathbf{w}) \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \mathbf{w})d\mathbf{w}$.

This constitutes the main objective of this work; how can we parametrize and optimize such distributions? Essentially, our target is to introduce dependence among the points of $\mathcal{D}$ in a manner that respects the two aforementioned conditions. We can then encode prior assumptions and inductive biases to the model by considering the relations among said points, a task much simpler than specifying a prior over latent global parameters $p_\theta(\mathbf{w})$. To this end, we introduce in the following our main contribution, the *Functional Neural Process* (FNP).

5.2.1 *Designing the Functional Neural Process*

On a high level the FNP follows the construction of a stochastic process as described at [41]; it posits a distribution over functions $h \in \mathcal{H}$ from $\mathbf{x}$ to y by first selecting a "reference" set of points from $\mathcal{X}$, and then basing the probability distribution over h around

those points. This concept is similar to the "inducing inputs" that are used in sparse GPs [169, 182]. More specifically, let $R = \{\mathbf{x}_1^r, \ldots, \mathbf{x}_K^r\}$ be such a reference set and let $O = \mathcal{X} \setminus R$ be the "other" set, i.e. the set of all possible points that are not in R. Now let $\mathcal{D}_x = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be any finite random set from $\mathcal{X}$, that constitutes our observed inputs. To facilitate the exposition we also introduce two more sets; $M = \mathcal{D}_x \setminus R$ that contains the points of $\mathcal{D}_x$ that are from O and $B = R \cup M$ that contains all of the points in $\mathcal{D}_x$ and R. We provide a Venn diagram in Fig. 12. In the following we describe the construction of the model, shown in Fig. 13, and then prove that it corresponds to an infinitely exchangeable stochastic process.

EMBEDDING THE INPUTS TO A LATENT SPACE The first step of the FNP is to embed each of the $\mathbf{x}_i$ of B independently to a latent representation $\mathbf{u}_i$

$$p_\theta(\mathbf{U}_B|\mathbf{X}_B) = \prod_{i \in B} p_\theta(\mathbf{u}_i|\mathbf{x}_i), \tag{64}$$

where $p_\theta(\mathbf{u}_i|\mathbf{x}_i)$ can be any distribution, e.g. a Gaussian or a delta peak, where its parameters, e.g. the mean and variance, are given by a function of $\mathbf{x}_i$. This function can be any function, provided that it is flexible enough to provide a meaningful representation for $\mathbf{x}_i$. For this reason, we employ neural networks, as their representational capacity has been demonstrated on a variety of complex high dimensional tasks, such as natural image generation and classification.

CONSTRUCTING A GRAPH OF DEPENDENCIES IN THE EMBEDDING SPACE The next step is to construct a dependency graph among the points in B; it encodes the correlations among the points in $\mathcal{D}$ that arise in the stochastic process. For example, in GPs such a correlation structure is encoded in the covariance matrix according to a kernel function $g(\cdot, \cdot)$ that measures the similarity between two inputs. In the FNP we adopt a different approach. Given the latent embeddings $\mathbf{U}_B$ that we obtained in the previous step we construct two directed graphs of dependencies among the points in B; a directed acyclic graph (DAG) $\mathbf{G}$ among the points in R and a bipartite graph $\mathbf{A}$ from R to M. These graphs are represented as random binary adjacency matrices, where e.g. $\mathbf{A}_{ij} = 1$ corresponds to the vertex j being a parent for the vertex i. The distribution of the bipartite graph can be defined as

$$p(\mathbf{A}|\mathbf{U}_R, \mathbf{U}_M) = \prod_{i \in M} \prod_{j \in R} \text{Bern}\left(\mathbf{A}_{ij}|g(\mathbf{u}_i, \mathbf{u}_j)\right). \tag{65}$$

where $g(\mathbf{u}_i, \mathbf{u}_j)$ provides the probability that a point $i \in M$ depends on a point j in the reference set R. This graph construction reminisces graphon [138] models, with however two important distinctions. Firstly, the embedding of each node is a vector rather than a scalar and secondly, the prior distribution over $\mathbf{u}$ is conditioned on an initial vertex representation $\mathbf{x}$ rather than being the same for all vertices. We believe that the latter is an important aspect, as it is what allows us to maintain enough information about the vertices and construct more informative graphs.

Figure 14: An example of the bipartite graph $\mathbf{A}$ that the FNP learns. The first column of each image is a query point and the rest are the five most probable parents from the R. We can see that the FNP associates same class inputs.

Figure 15: A DAG over R on MNIST, obtained after propagating the means of $\mathbf{U}$ and thresholding edges that have less than 0.5 probability in $\mathbf{G}$. We can see that FNP learns a meaningful $\mathbf{G}$ by connecting points that have the same class.

The DAG among the points in R is a bit trickier, as we have to adopt a topological ordering of the vectors in $\mathbf{U}_R$ in order to avoid cycles. Inspired by the concept of stochastic orderings [160], we define an ordering according to a parameter free scalar projection $t(\cdot)$ of $\mathbf{u}$, i.e. $\mathbf{u}_i > \mathbf{u}_j$ when $t(\mathbf{u}_i) > t(\mathbf{u}_j)$. The function $t(\cdot)$ is defined as $t(\mathbf{u}_i) = \sum_k t_k(\mathbf{u}_{ik})$ where each individual $t_k(\cdot)$ is a monotonic function (e.g. the log CDF of a standard normal distribution); in this case we can guarantee that $\mathbf{u}_i > \mathbf{u}_j$ when individually for all of the dimensions k we have that $\mathbf{u}_{ik} > \mathbf{u}_{jk}$ under $t_k(\cdot)$. This ordering can then be used in

$$p(\mathbf{G}|\mathbf{U}_R) = \prod_{i \in R} \prod_{j \in R, j \neq i} \text{Bern}\left(\mathbf{G}_{ij} | \mathbb{I}[t(\mathbf{u}_i) > t(\mathbf{u}_j)] g(\mathbf{u}_i, \mathbf{u}_j)\right) \tag{66}$$

which leads into random adjacency matrices $\mathbf{G}$ that can be re-arranged into a triangular structure with zeros in the diagonal (i.e. DAGs). In a similar manner, such a DAG construction reminisces of digraphon models [26], a generalization of graphons to the directed case. The same two important distinctions still apply; we are using vector instead of scalar representations and the prior over the representation of each vertex i depends on $\mathbf{x}_i$. It is now straightforward to bake in any relational inductive biases that we want our function to have by appropriately defining the $g(\cdot, \cdot)$ that is used for the construction of $\mathbf{G}$ and $\mathbf{A}$. For example, we can encode an inductive bias that neighboring points should be dependent by choosing $g(\mathbf{u}_i, \mathbf{u}_j) = \exp\left(-\frac{\tau}{2}\|\mathbf{u}_i - \mathbf{u}_j\|^2\right)$. This what we used in practice. We provide examples of the $\mathbf{A}$, $\mathbf{G}$ that FNPs learn in Figures 14, 15 respectively.

PARAMETRIZING THE PREDICTIVE DISTRIBUTION Having obtained the dependency graphs $\mathbf{A}, \mathbf{G}$, we are now interested in how to construct a predictive model that induces them. To this end, we parametrize predictive distributions for each target variable y_i that explicitly depend on the reference set R according to the structure of $\mathbf{G}$ and $\mathbf{A}$. This is

realized via a local latent variable $\mathbf{z}_i$ that summarizes the context from the selected parent points in R and their targets $\mathbf{y}_R$

$$\begin{aligned}
&\int p_\theta(\mathbf{y}_B, \mathbf{Z}_B | R, \mathbf{G}, \mathbf{A}) d\mathbf{Z}_B = \\
&\quad \int p_\theta(\mathbf{y}_R, \mathbf{Z}_R | R, \mathbf{G}) d\mathbf{Z}_R \int p_\theta(\mathbf{y}_M, \mathbf{Z}_M | R, \mathbf{y}_R, \mathbf{A}) d\mathbf{Z}_M \\
&= \prod_{i \in R} \int p_\theta \left(\mathbf{z}_i | \text{par}_{\mathbf{G}_i}(R, \mathbf{y}_R) \right) p_\theta(y_i | \mathbf{z}_i) d\mathbf{z}_i \\
&\quad \prod_{j \in M} \int p_\theta \left(\mathbf{z}_j | \text{par}_{\mathbf{A}_j}(R, \mathbf{y}_R) \right) p_\theta(y_j | \mathbf{z}_j) d\mathbf{z}_j
\end{aligned} \tag{67}$$

where $\text{par}_{\mathbf{G}_i}(\cdot), \text{par}_{\mathbf{A}_j}(\cdot)$ are functions that return the parents of the point i, j according to $\mathbf{G}, \mathbf{A}$ respectively. Notice that we are guaranteed that the decomposition to the conditionals at Eq. 67 is valid, since the DAG $\mathbf{G}$ coupled with $\mathbf{A}$ correspond to another DAG. Since permutation invariance in the parents is necessary for an overall exchangeable model, we define each distribution over $\mathbf{z}$, e.g. $p\left(\mathbf{z}_i | \text{par}_{\mathbf{A}_i}(R, \mathbf{y}_R)\right)$, as an independent Gaussian distribution per dimension k of $\mathbf{z}$[2]

$$\begin{aligned}
p_\theta \left(\mathbf{z}_{ik} | \text{par}_{\mathbf{A}_i}(R, \mathbf{y}_R) \right) = \mathcal{N} \Bigg(\mathbf{z}_{ik} \Bigg| & C_i \sum_{j \in R} \mathbf{A}_{ij} \mu_\theta(\mathbf{x}_j^r, y_j^r)_k, \\
& \exp \left(C_i \sum_{j \in R} \mathbf{A}_{ij} \nu_\theta(\mathbf{x}_j^r, y_j^r)_k \right) \Bigg)
\end{aligned} \tag{68}$$

where the $\mu_\theta(\cdot, \cdot)$ and $\nu_\theta(\cdot, \cdot)$ are vector valued functions with a codomain in $\mathbb{R}^{|z|}$ that transform the data tuples of $R, \mathbf{y}_R$. The C_i is a normalization constant with $C_i = (\sum_j A_{ij} + \epsilon)^{-1}$, i.e. it corresponds to the reciprocal of the number of parents of point i, with an extra small ϵ to avoid division by zero when a point has no parents. By observing Eq. 67 we can see that the prediction for a given y_i depends on the input covariates $\mathbf{x}_i$ only indirectly via the graphs $\mathbf{G}, \mathbf{A}$ which are a function of $\mathbf{u}_i$. Intuitively, it encodes the inductive bias that predictions on points that are "far away", i.e. have very small probability of being connected to the reference set via $\mathbf{A}$, will default to an uninformative standard normal prior over $\mathbf{z}_i$ hence a constant prediction for y_i. This is similar to the behaviour that GPs with RBF kernels exhibit.

Nevertheless, Eq. 67 can also hinder extrapolation, something that neural networks can do well. In case extrapolation is important, we can always add a direct path by conditioning the prediction on $\mathbf{u}_i$, the latent embedding of $\mathbf{x}_i$, i.e. $p(y_i | \mathbf{z}_i, \mathbf{u}_i)$. This can serve as a middle ground where we can allow some extrapolation via $\mathbf{u}$. In general, it provides a knob, as we can now interpolate between GP and neural network behaviours by e.g. changing the dimensionalities of $\mathbf{z}$ and $\mathbf{u}$.

2 The factorized Gaussian distribution was chosen for simplicity, and it is not a limitation. Any distribution is valid for $\mathbf{z}$ provided that it defines a permutation invariant probability density w.r.t. the parents.

PUTTING EVERYTHING TOGETHER: THE FNP AND FNP$^+$ MODELS Now by putting everything together we arrive at the overall definitions of the two FNP models that we propose

$$\mathcal{FNP}_\theta(\mathcal{D}) := \sum_{\mathbf{G},\mathbf{A}} \int p_\theta(\mathbf{U}_B|\mathbf{X}_B)p(\mathbf{G},\mathbf{A}|\mathbf{U}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{G},\mathbf{A})d\mathbf{U}_B d\mathbf{Z}_B dy_{i\in R\setminus\mathcal{D}_x}, \tag{69}$$

$$\mathcal{FNP}^+_\theta(\mathcal{D}) := \sum_{\mathbf{G},\mathbf{A}} \int p_\theta(\mathbf{U}_B,\mathbf{G},\mathbf{A}|\mathbf{X}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{U}_B,\mathbf{G},\mathbf{A})d\mathbf{U}_B d\mathbf{Z}_B dy_{i\in R\setminus\mathcal{D}_x}, \tag{70}$$

where the first makes predictions according to Eq. 67 and the second further conditions on $\mathbf{u}$. Notice that besides the marginalizations over the latent variables and graphs, we also marginalize over any of the points in the reference set that are not part of the observed dataset $\mathcal{D}$. This is necessary for the proof of consistency that we provide later. For this work, we always chose the reference set to be a part of the dataset $\mathcal{D}$ so the extra integration is omitted. In general, the marginalization can provide a mechanism to include unlabelled data to the model which could be used to e.g. learn a better embedding $\mathbf{u}$ or "impute" the missing labels. We leave the exploration of such an avenue for future work. Having defined the models at Eq. 69, 70 we now prove that they both define valid permutation invariant stochastic processes by borrowing the methodology described at [41].

Proposition 1. *The distributions defined at Eq. 69, 70 are valid permutation invariant stochastic processes, hence they correspond to Bayesian models.*

Proof sketch. The full proof can be found in the Appendix. Permutation invariance can be proved by noting that each of the terms in the products are permutation equivariant w.r.t. permutations of $\mathcal{D}$ hence each of the individual distributions defined at Eq. 69, 70 are permutation invariant due to the products. To prove consistency we have to consider two cases [41], the case where we add a point that is part of R and the case where we add one that is not part of R. In the first case, marginalizing out that point will lead to the same distribution (as we were marginalizing over that point already), whereas in the second case the point that we are adding is a leaf in the dependency graph, hence marginalizing it doesn't affect the other points. □

5.2.2 *The FNPs in practice: fitting and predictions*

Having defined the two models, we are now interested in how we can fit their parameters θ when we are presented with a dataset $\mathcal{D}$, as well as how to make predictions for novel inputs $\mathbf{x}^*$. For simplicity, we assume that $R \subseteq \mathcal{D}_x$ and focus on the FNP as the derivations for the FNP$^+$ are analogous. Notice that in this case we have that $B = \mathcal{D}_x = \mathbf{X}_\mathcal{D}$.

FITTING THE MODEL TO DATA Fitting the model parameters with maximum marginal likelihood is difficult, as the necessary integrals / sums of Eq.69 are intractable. For this

reason, we employ variational inference and maximize the following lower bound to the marginal likelihood of $\mathcal{D}$

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{U}_\mathcal{D},\mathbf{G},\mathbf{A},\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D})}[\log p_\theta(\mathbf{U}_\mathcal{D},\mathbf{G},\mathbf{A},\mathbf{Z}_\mathcal{D},\mathbf{y}_\mathcal{D}|\mathbf{X}_\mathcal{D}) - \log q_\phi(\mathbf{U}_\mathcal{D},\mathbf{G},\mathbf{A},\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D})], \tag{71}$$

with respect to the model parameters θ and variational parameters ϕ. For a tractable lower bound, we assume the variational distribution factorization

$$q_\phi(\mathbf{U}_\mathcal{D},\mathbf{G},\mathbf{A},\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D}) = p_\theta(\mathbf{U}_\mathcal{D}|\mathbf{X}_\mathcal{D})p(\mathbf{G}|\mathbf{U}_\mathcal{R})p(\mathbf{A}|\mathbf{U}_\mathcal{D})q_\phi(\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D})$$

with $q_\phi(\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D}) = \prod_{i=1}^{|\mathcal{D}|} q_\phi(\mathbf{z}_i|\mathbf{x}_i)$. This leads to

$$\begin{aligned}
&\mathcal{L}_\mathcal{R} + \mathcal{L}_{\mathcal{M}|\mathcal{R}} = \\
&\mathbb{E}_{p_\theta(\mathbf{U}_\mathcal{R},\mathbf{G}|\mathbf{X}_\mathcal{R})q_\phi(\mathbf{Z}_\mathcal{R}|\mathbf{X}_\mathcal{R})}[\log p_\theta(\mathbf{y}_\mathcal{R},\mathbf{Z}_\mathcal{R}|\mathcal{R},\mathbf{G}) - \log q_\phi(\mathbf{Z}_\mathcal{R}|\mathbf{X}_\mathcal{R})] + \\
&+ \mathbb{E}_{p_\theta(\mathbf{U}_\mathcal{D},\mathbf{A}|\mathbf{X}_\mathcal{D})q_\phi(\mathbf{Z}_\mathcal{M}|\mathbf{X}_\mathcal{M})}[\log p_\theta(\mathbf{y}_\mathcal{M}|\mathbf{Z}_\mathcal{M}) + \log p_\theta(\mathbf{Z}_\mathcal{M}|\mathrm{par}_\mathbf{A}(\mathcal{R},\mathbf{y}_\mathcal{R})) \\
&\qquad - \log q_\phi(\mathbf{Z}_\mathcal{M}|\mathbf{X}_\mathcal{M})]
\end{aligned} \tag{72}$$

where we decomposed the lower bound into the terms for the reference set $\mathcal{R}$, $\mathcal{L}_\mathcal{R}$, and the terms that correspond to $\mathcal{M}$, $\mathcal{L}_{\mathcal{M}|\mathcal{R}}$. For large datasets $\mathcal{D}$ we are interested in doing efficient optimization of this bound. While the first term is not, in general, amenable to minibatching, the second term is. As a result, we can use minibatches that scale according to the size of the reference set $\mathcal{R}$. We provide more details in the Appendix.

In practice, for all of the distributions over $\mathbf{u}$ and $\mathbf{z}$, we use diagonal Gaussians, whereas for $\mathbf{G}, \mathbf{A}$ we use the concrete / Gumbel-softmax relaxations [85, 120] during training. In this way we can jointly optimize θ, ϕ with gradient based optimization by employing the pathwise derivatives obtained with the reparametrization trick [91, 151]. Furthermore, we tie most of the parameters θ of the model and ϕ of the inference network, as the regularizing nature of the lower bound can alleviate potential overfitting of the model parameters θ. More specifically, for $p_\theta(\mathbf{u}_i|\mathbf{x}_i)$, $q_\phi(\mathbf{z}_i|\mathbf{x}_i)$ we share a neural network torso and have two output heads, one for each distribution. We also parametrize the priors over the latent $\mathbf{z}$ in terms of the $q_\phi(\mathbf{z}_i|\mathbf{x}_i)$ for the points in $\mathcal{R}$; the $\mu_\theta(\mathbf{x}_i^r, y_i^r)$, $\nu_\theta(\mathbf{x}_i^r, y_i^r)$ are both defined as $\mu_q(\mathbf{x}_i^r) + \mu_y^r$, $\nu_q(\mathbf{x}_i^r) + \nu_y^r$, where $\mu_q(\cdot), \nu_q(\cdot)$ are the functions that provide the mean and variance for $q_\phi(\mathbf{z}_i|\mathbf{x}_i)$ and μ_y^r, ν_y^r are linear embeddings of the labels.

It is interesting to see that the overall bound at Eq. 72 reminisces the bound of a latent variable model such as a variational autoencoder (VAE) [91, 151] or a deep variational information bottleneck model (VIB) [6]. We aim to predict the label y_i of a given point $\mathbf{x}_i$ from its latent code $\mathbf{z}_i$ where the prior, instead of being globally the same as in [6, 91, 151], it is conditioned on the parents of that particular point. The conditioning is also intuitive, as it is what converts the i.i.d. to the more general exchangeable model. This is also similar to the VAE for unsupervised learning described at associative compression networks (ACN) [57] and reminisces works on few-shot learning [14].

THE POSTERIOR PREDICTIVE DISTRIBUTION In order to perform predictions for unseen points $\mathbf{x}^*$, we employ the posterior predictive distribution of FNPs. More specif-

ically, we can show that by using Bayes rule, the predictive distribution of the FNPs has the following simple form

$$\sum_{\mathbf{a}^*} \int p_\theta(\mathbf{U}_R, \mathbf{u}^*|\mathbf{X}_R, \mathbf{x}^*) p(\mathbf{a}^*|\mathbf{U}_R, \mathbf{u}^*) \\ p_\theta(\mathbf{z}^*|\text{par}_{\mathbf{a}^*}(R, \mathbf{y}_R)) p_\theta(y^*|\mathbf{z}^*) d\mathbf{U}_R d\mathbf{u}^* d\mathbf{z}^* \tag{73}$$

where $\mathbf{u}$ are the representations given by the neural network and $\mathbf{a}^*$ is the binary vector that denotes which points from R are the parents of the new point. We provide more details in the Appendix. Intuitively, we first project the reference set and the new point on the latent space $\mathbf{u}$ with a neural network and then make a prediction y^* by basing it on the parents from R according to $\mathbf{a}^*$. This predictive distribution reminisces the models employed in few-shot learning [187].

5.3 RELATED WORK

There has been a long line of research in Bayesian Neural Networks (BNNs) [23, 56, 72, 89, 114, 163]. A lot of works have focused on the hard task of posterior inference for BNNs, by positing more flexible posteriors [13, 113, 114, 163, 204]. The exploration of more involved priors has so far not gain much traction, with the exception of a handful of works [10, 64, 89, 112]. For flexible stochastic processes, we have a line of works that focus on (scalable) Gaussian Processes (GPs); these revolve around sparse GPs [169, 182], using neural networks to parametrize the kernel of a GP [196, 197], employing finite rank approximations to the kernel [39, 71] or parametrizing kernels over structured data [121, 194]. Compared to such approaches, FNPs can in general be more scalable due to not having to invert a matrix for prediction and, furthermore, they can easily support arbitrary likelihood models (e.g. for discrete data) without having to consider appropriate transformations of a base Gaussian distribution (which usually requires further approximations).

There have been interesting recent works that attempt to merge stochastic processes and neural networks. Neural Processes (NPs) [52] define distributions over global latent variables in terms of subsets of the data, while Attentive NPs [87] extend NPs with a deterministic path that has a cross-attention mechanism among the datapoints. In a sense, FNPs can be seen as a variant where we discard the global latent variables and instead incorporate cross-attention in the form of a dependency graph among local latent variables. Another line of works is the Variational Implicit Processes (VIPs) [116], which consider BNN priors and then use GPs for inference, and functional variational BNNs (fBNNs) [176], which employ GP priors and use BNNs for inference. Both methods have their drawbacks, as with VIPs we have to posit a meaningful prior over global parameters and the objective of fBNNs does not always correspond to a bound of the marginal likelihood. Finally, there is also an interesting line of works that study wide neural networks with random Gaussian parameters and discuss their equivalences with Gaussian Processes [102, 137], as well as the resulting kernel [84].

Similarities can be also seen at other works; Associative Compression Networks (ACNs) [57] employ similar ideas for generative modelling with VAEs and conditions the prior over the

latent variable of a point to its nearest neighbors. Correlated VAEs [180] similarly employ a (a-priori known) dependency structure across the latent variables of the points in the dataset. In few-shot learning, metric-based approaches [14, 93, 168, 177, 187] similarly rely on similarities w.r.t. a reference set for predictions.

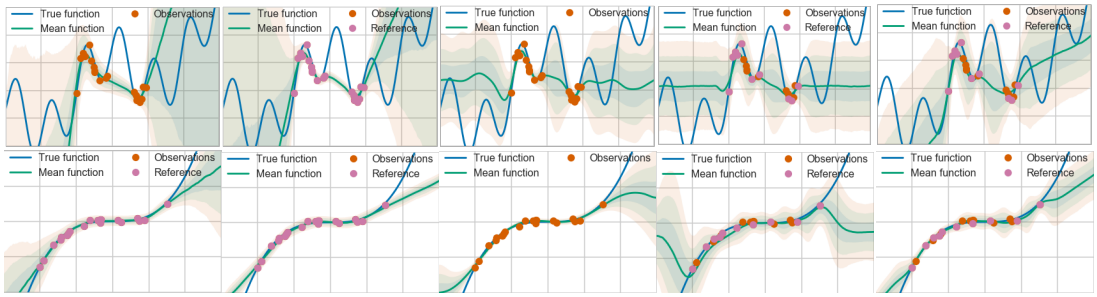
5.4 EXPERIMENTS

We performed two main experiments in order to verify the effectiveness of FNPs. We implemented and compared against 4 baselines: a standard neural network (denoted as NN), a neural network trained and evaluated with Monte Carlo (MC) dropout [50] and a Neural Process (NP) [52] architecture. The architecture of the NP was designed in a way that is similar to the FNP. For the first experiment we explored the inductive biases we can encode in FNPs by visualizing the predictive distributions in toy 1d regression tasks. For the second, we measured the prediction performance and uncertainty quality that FNPs can offer on the benchmark image classification tasks of MNIST and CIFAR 10. For this experiment, we also implemented and compared against a Bayesian neural network trained with variational inference [23]. We provide the experimental details in the Appendix.

For all of the experiments in the paper, the NP was trained in a way that mimics the FNP, albeit we used a different set R at every training iteration in order to conform to the standard NP training regime. More specifically, a random amount from 3 to $num(R)$ points were selected as a context from each batch, with $num(R)$ being the maximum amount of points allocated for R. For the toy regression task we set $num(R) = N - 1$.

EXPLORING THE INDUCTIVE BIASES IN TOY REGRESSION To visually access the inductive biases we encode in the FNP we experiment with two toy 1-d regression tasks described at [140] and [72] respectively. The generative process of the first corresponds to drawing 12 points from $U[0, 0.6]$, 8 points from $U[0.8, 1]$ and then parametrizing the target as $y_i = x_i + \epsilon + \sin(4(x_i + \epsilon)) + \sin(13(x_i + \epsilon))$ with $\epsilon \sim \mathcal{N}(0, 0.03^2)$. This generates a nonlinear function with "gaps" in between the data where we, ideally, want the uncertainty to be high. For the second we sampled 20 points from $U[-4, 4]$ and then parametrized the target as $y_i = x_i^3 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 9)$. For all of the models we used a heteroscedastic noise model. Furthermore, due to the toy nature of this experiment, we also included a Gaussian Process (GP) with an RBF kernel. We used 50 dimensions for the global latent of NP for the first task and 10 dimensions for the second. For the FNP models we used $3, 50$ dimensions for the $\mathbf{u}, \mathbf{z}$ for the first task and $3, 10$ for the second. For the reference set R we used 10 random points for the FNPs and the full dataset for the NP.

The results we obtain are presented in Figure 16. We can see that for the first task the FNP with the RBF function for $g(\cdot, \cdot)$ has a behaviour that is very similar to the GP. We can also see that in the second task it has the tendency to quickly move towards a flat prediction outside the areas where we observe points, something which we argued about at Section 5.2.1. This is not the case for MC-dropout or NP where we see a more linear behaviour on the uncertainty and erroneous overconfidence, in the case of the first task, in the areas in-between the data. Nevertheless, they do seem to extrapolate better compared to the FNP and GP. The FNP$^+$ seems to combine the best of both worlds as it allows for



(a) MC-dropout (b) Neural Process (c) Gaussian Process (d) FNP (e) FNP$^+$

Figure 16: Predictive distributions for the two toy regression tasks according to the different models we considered. Shaded areas correspond to $\pm$ 3 standard deviations.

extrapolation and GP like uncertainty, although a free bits [32] modification of the bound for $\mathbf{z}$ was helpful in encouraging the model to rely more on these particular latent variables. Empirically, we observed that adding more capacity on $\mathbf{u}$ can move the FNP$^+$ closer to the behaviour we observe for MC-dropout and NPs. In addition, increasing the amount of model parameters θ can make FNPs overfit, a fact that can result into a reduction of predictive uncertainty.

PREDICTION PERFORMANCE AND UNCERTAINTY QUALITY For the second task we considered the image classification of MNIST and CIFAR 10. For MNIST we used a LeNet-5 architecture that had two convolutional and two fully connected layers, whereas for CIFAR we used a VGG-like architecture that had 6 convolutional and two fully connected. In both experiments we used 300 random points from $\mathcal{D}$ as R for the FNPs and for NPs, in order to be comparable, we randomly selected up to 300 points from the current batch for the context points during training and used the same 300 points as FNPs for evaluation. The dimensionality of $\mathbf{u}, \mathbf{z}$ was $32, 64$ for the FNP models in both datasets, whereas for the NP the dimensionality of the global variable was 32 for MNIST and 64 for CIFAR.

As a proxy for the uncertainty quality we used the task of out of distribution (o.o.d.) detection; given the fact that FNPs are Bayesian models we would expect that their epistemic uncertainty will increase in areas where we have no data (i.e. o.o.d. datasets). The metric that we report is the average entropy on those datasets as well as the area under an ROC curve (AUCR) that determines whether a point is in or out of distribution according to the predictive entropy. Notice that it is simple to increase the first metric by just learning a trivial model but that would be detrimental for AUCR; in order to have good AUCR the model must have low entropy on the in-distribution test set but high entropy on the o.o.d. datasets. For the MNIST model we considered notMNIST, Fashion MNIST, Omniglot, Gaussian $\mathcal{N}(0, 1)$ and uniform $\mathcal{U}[0, 1]$ noise as o.o.d. datasets whereas for CIFAR 10 we considered SVHN, a tinyImagenet resized to 32 pixels, iSUN and similarly Gaussian and uniform noise. The summary of the results can be seen at Table 6.

Table 6: Accuracy and uncertainty on MNIST and CIFAR 10 from 100 posterior predictive samples. For the all of the datasets the first column is the average predictive entropy whereas for the o.o.d. datasets the second is the AUCR and for the in-distribution it is the test error in %.

	NN	MC-Dropout	VI BNN	NP	FNP	FNP$^+$
MNIST	0.01 / 0.6	**0.05 / 0.5**	0.02 / 0.6	0.01 / 0.6	0.04 / 0.7	0.02 / 0.7
nMNIST	1.03 / 99.73	1.30 / 99.48	1.33 / 99.80	1.31 / 99.90	1.94 / 99.90	**1.77 / 99.96**
fMNIST	0.81 / 99.16	1.23 / 99.07	0.92 / 98.61	0.71 / 98.98	**1.85 / 99.66**	1.55 / 99.58
Omniglot	0.71 / 99.44	1.18 / 99.29	1.61 / 99.91	0.86 / 99.69	1.87 / 99.79	**1.71 / 99.92**
Gaussian	0.99 / 99.63	**2.03 / 100.0**	**1.77 / 100.0**	1.58 / 99.94	1.94 / 99.86	**2.03 / 100.0**
Uniform	0.85 / 99.65	0.65 / 97.58	1.41 / 99.87	1.46 / 99.96	2.11 / 99.98	**1.88 / 99.99**
Average	0.9±0.1 / 99.5±0.1	1.3±0.2 / 99.1±0.4	1.4±0.1 / 99.6±0.3	1.2±0.2 / 99.7±0.2	1.9±0.1 / 99.8±0.1	**1.8±0.1 / 99.9±0.1**
CIFAR10	0.05 / 6.9	0.06 / 7.0	**0.06 / 6.4**	0.06 / 7.5	0.18 / 7.2	0.08 / 7.2
SVHN	0.44 / 93.1	0.42 / 91.3	0.45 / 91.8	0.38 / 90.2	**1.09 / 94.3**	0.42 / 89.8
tImag32	0.51 / 92.7	0.59 / 93.1	0.52 / 91.9	0.45 / 89.8	**1.20 / 94.0**	0.74 / 93.8
iSUN	0.52 / 93.2	0.59 / 93.1	0.57 / 93.2	0.47 / 90.8	**1.30 / 95.1**	0.81 / 94.8
Gaussian	0.01 / 72.3	0.05 / 72.1	0.76 / 96.9	0.37 / 91.9	1.13 / 95.4	**0.96 / 97.9**
Uniform	**0.93 / 98.4**	0.08 / 77.3	0.65 / 96.1	0.17 / 87.8	0.71 / 89.7	**0.99 / 98.4**
Average	0.5±0.2 / 89.9±4.5	0.4±0.1 / 85.4±4.5	0.6±0.1 / 94±1.1	0.4±0.1 / 90.1±0.7	1.1±0.1 / 93.7±1.0	**0.8±0.1 / 94.9±1.6**

We observe that both FNPs have comparable accuracy to the baseline models while having higher average entropies and AUCR on the o.o.d. datasets. FNP$^+$ in general seems to perform better than FNP. The FNP did have a relatively high in-distribution entropy for CIFAR 10, perhaps denoting that a larger R might be more appropriate. We further see that the FNPs have almost always better AUCR than all of the baselines we considered. Interestingly, out of all the non-noise o.o.d. datasets we did observe that Fashion MNIST and SVHN, were the hardest to distinguish on average across all the models. This effect seems to agree with the observations from [133], although more investigation is required. We also observed that, sometimes, the noise datasets on all of the baselines can act as "adversarial examples" [179] thus leading to lower entropy than the in-distribution test set (e.g. Gaussian noise for the NN on CIFAR 10). FNPs did have a similar effect on CIFAR 10, e.g. the FNP on uniform noise, although to a much lesser extent. We leave the exploration of this phenomenon for future work. It should be mentioned that other advances in o.o.d. detection, e.g. [33, 104], are orthogonal to FNPs and could further improve performance.

Table 7: Results obtained by training a NP model with a fixed reference set (akin to FNP) and a FNP$^+$ model with a random reference set (akin to NP).

	NP fixed R	FNP$^+$ random R
MNIST	0.01 / 0.6	0.02 / 0.8
nMNIST	1.09 / 99.78	2.20 / 100.0
fMNIST	0.64 / 98.34	1.58 / 99.78
Omniglot	0.79 / 99.53	2.06 / 99.99
Gaussian	1.79 / 99.96	2.28 / 100.0
Uniform	1.42 / 99.93	2.23 / 100.0
CIFAR10	0.07 / 7.5	0.09 / 6.9
SVHN	0.46 / 91.5	0.56 / 91.4
tImag32	0.55 / 91.5	0.77 / 93.4
iSUN	0.60 / 92.6	0.83 / 94.0
Gaussian	0.20 / 87.2	1.23 / 99.1
Uniform	0.53 / 94.3	0.90 / 97.2

We further performed additional experiments in order to better disentangle the performance differences between NPs and FNPs: we trained an NP with the same fixed reference set R as the FNPs throughout training, as well as an FNP$^+$ where we randomly sample a new R for every batch (akin to the NP) and use the same R as the NP for evaluation. While we argued in the construction of the FNPs that with a fixed R we can obtain a stochastic process, we could view the case with random R as an ensemble of stochastic processes, one for each realization of R. The results from these models can be seen at Table 7. On the one hand, the FNP$^+$ still provides robust uncertainty while the randomness in R seems to improve the o.o.d. detection, possibly due to the implicit regularization. On the other hand the fixed R seems to hurt the NP, as the o.o.d. detection decreased, similarly hinting that the random R has beneficial regularizing effects.

Finally, we provide some additional insights after doing ablation studies on MNIST w.r.t. the sensitivity to the number of points in R for NP, FNP and FNP$^+$, as well as varying the amount of dimensions for $\mathbf{u}, \mathbf{z}$ in the FNP$^+$. The results can be found in the Appendix. We generally observed that NP models have lower average entropy at the o.o.d. datasets than both FNP and FNP$^+$ irrespective of the size of R. The choice of R seems to be more important for the FNPs rather than NPs, with FNP needing a larger R, compared to FNP$^+$, to fit the data well. In general, it seems that it is not the quantity of points that matters but rather the quality; the performance did not always increase with more points. This supports the idea of a "coreset" of points, thus exploring ideas to infer it is a promising research direction that could improve scalability and alleviate the dependence of FNPs on a reasonable R. As for the trade-off between $\mathbf{z}, \mathbf{u}$ in FNP$^+$; a larger capacity for $\mathbf{z}$, compared to $\mathbf{u}$, leads to better uncertainty whereas the other way around seems to improve accuracy. These observations are conditioned on having a reasonably large $\mathbf{u}$, which facilitates for meaningful $\mathbf{G}, \mathbf{A}$.

5.5 DISCUSSION

We presented a novel family of exchangeable stochastic processes, the Functional Neural Processes (FNPs). In contrast to NPs [52] that employ global latent variables, FNPs operate by employing local latent variables along with a dependency structure among them, a fact that allows for easier encoding of inductive biases. We verified the potential of FNPs experimentally, and showed that they can serve as competitive alternatives. We believe that FNPs open the door to plenty of exciting avenues for future research; designing better function priors by e.g. imposing a manifold structure on the FNP latents [46], extending FNPs to unsupervised learning by e.g. adapting ACNs [57] or considering hierarchical models similar to deep GPs [40].

Part II

PROBABILISTIC COMPRESSION FOR NEURAL NETWORKS

In this section of the thesis we propose techniques that can allow for practical neural network compression and speed-ups. In the first chapter we make a connection to part i and show how Bayesian inference through specific choices for the prior distributions over the parameters can lead to highly compressed models through joint pruning and quantization. In the next two chapters, we tease these two objectives apart and take a closer look at sparsity and quantization by proposing two general purpose recipes that can allow for gradient based optimization of such objectives.

6

BAYESIAN COMPRESSION FOR DEEP LEARNING

Compression and computational efficiency in deep learning have become a problem of great significance. In this chapter, we argue that the most principled and effective way to attack this problem is by taking a Bayesian point of view, where through sparsity inducing priors we prune large parts of the network. We introduce two novelties: 1) we use hierarchical priors to prune nodes instead of individual weights, and 2) we use the posterior uncertainties to determine the fixed point precision to encode the weights. Both factors significantly contribute to achieving the state of the art in terms of compression rates, while still staying competitive with methods designed to optimize for speed or energy efficiency.[1]

6.1 INTRODUCTION

Deep neural networks have been improving state of the art in several benchmark tasks, ranging from computer vision to natural language processing. Despite these successes, they face challenges when they are to be deployed for real world applications and at resource constrained devices, such as mobile phones. In such cases, the computational and energy consumption costs can become prohibitive. For this reason, compression, quantization and efficiency have risen as a topic of increasing practical importance.

While these metrics are certainly correlated, optimizing for one might not always yield gains on the other. For example, compressing a convolutional neural network (CNN) by removing parameters from its fully connected layers can, sometimes significantly, reduce its overall number of parameters, but that might not necessarily lead to better efficiency as the majority of computations happen at the convolutional layers. As an example, we have that 96% of the parameters of Alexnet are in the fully connected layers but 91% of the overall computation is from the convolutional layers [178].

In the literature, the most common way to address these problems is by "shrinking" the neural network architecture, since NNs suffer from significant parameter redundancy [42], along with reducing the effective bit precision for each of the neural network parameters. For the former, relevant methods are network pruning, where weights are being removed from the network [59, 66, 101] and student-teacher architectures, where a smaller, and thus more efficient, network is being trained to mimic the predictions of the larger teacher network [12, 77].

1 This chapter is adapted from our publication [112].

From a Bayesian perspective network pruning and reducing bit precision for the weights is aligned with achieving high accuracy, because Bayesian methods search for the optimal model structure (which leads to pruning with sparsity inducing priors), and reward uncertain posteriors over parameters through the bits back argument [76] (which leads to removing insignificant bits). This relation is made explicit in the MDL principle [58] which is known to be related to Bayesian inference.

In this paper we will use the variational Bayesian approximation for Bayesian inference which has also been explicitly interpreted in terms of model compression [76]. By employing sparsity inducing priors for hidden units (and not individual weights) we can prune neurons including all their ingoing and outgoing weights. This avoids more complicated and inefficient coding schemes needed for pruning or vector quantizing individual weights. As a additional Bayesian bonus we can use the posterior uncertainties to assess which bits are significant and remove the ones which fluctuate too much under posterior sampling. From this we derive the fixed point precision per layer, which is still practical on hardware.

6.2 VARIATIONAL BAYES AND MINIMUM DESCRIPTION LENGTH

A core aspect of our proposed method is variational inference [56, 76, 78, 145, 188]. In variational inference, the goal is to approximate the posterior distribution of a probabilistic model by optimizing the parameters of a variational approximation. More specifically, let $\mathcal{D}$ be a dataset consisting of N tuples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1:N}$ denoting our inputs $\mathbf{x}$ and corresponding targets $\mathbf{y}$. Now consider a parametric likelihood for our dataset defined as $p(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$, where $\mathbf{w}$ are the parameters of our model, e.g., a neural network. We can then posit a prior distribution over these parameters $p(\mathbf{w})$ and seek the posterior distribution $p(\mathbf{w}|\mathcal{D})$. Since for neural networks, this posterior distribution is intractable, we will posit a variational approximation $q_\phi(\mathbf{w})$ with parameters ϕ to the true posterior distribution $p(\mathbf{w}|\mathcal{D})$ and we will optimize ϕ in order to maximize the so-called Evidence Lower Bound (ELBO)

$$\mathcal{L}(\phi) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathcal{D}|\mathbf{w})]}_{\mathcal{L}^E} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathbf{w})] + \mathcal{H}(q_\phi(\mathbf{w}))}_{\mathcal{L}^C} \tag{74}$$

where $\mathcal{H}(q)$ corresponds to the entropy of our approximate posterior distribution. Provided that our approximation $q_\phi(\mathbf{w})$ is sufficiently flexible, optimization of Eq. 74 will lead to $q_\phi(\mathbf{w}) = p(\mathbf{w}|\mathcal{D})$.

It is interesting to see that the ELBO is comprised from two terms, the data fit term $\mathcal{L}^E$, i.e., how well does our approximation $q_\phi(\mathbf{w})$ fit the dataset $\mathcal{D}$, and the complexity loss term $\mathcal{L}^C$, i.e., how much does our approximation differ from our prior, and independent of the data, assumptions in $p(\mathbf{w})$. This connects the ELBO with a fundamental theorem in information theory, the minimum description length (MDL) principle [58]. The MDL principle connects to compression directly, in that it defines the best model / hypothesis, to be that which minimizes the sum of the model complexity and the data misfit errors with the minimum number of bits [152, 153].

As a result, by maximizing the ELBO we will naturally arrive at compressed solutions, in that parameters which do not contribute much in explaining the data $\mathcal{D}$ will be "ignored" and thus fall back to our prior assumptions. Therefore, by employing priors that encourage sparsity for groups of weights that are associated with specific neurons, this mechanism will start pruning the unnecessary neurons and will result into neural networks that are more efficient. As an additional bonus, we can further benefit from the bits-back argument [76, 78] due to the stochastic weights $q_\phi(\mathbf{w})$ and the entropy term in the ELBO, leading to even more compression. This is in contrast to deterministic weights that lead to $\mathcal{H}(\delta(\mathbf{w})) = -\infty$[2].

In practice, optimizing Eq. 74 can be challenging, due to the fact that the $\mathcal{L}^E$ term is intractable for neural networks when we maintain a distribution over weights via $q_\phi(\mathbf{w})$. Nevertheless, for continuous $q_\phi(\mathbf{w})$ not all is lost, as we can efficiently perform Monte Carlo approximations of the expectation with the reparametrization trick [91, 151]; we can express the random sampling $\mathbf{w} \sim q_\phi(\mathbf{w})$ as a deterministic and differentiable transformation that involves parameter-less noise ϵ and the parameters of our approximation ϕ, i.e., $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, $\mathbf{w} = f(\boldsymbol{\epsilon}, \phi)$. In this way, we can obtain unbiased stochastic gradients of the ELBO with respect to ϕ and plug them to any off-the-shelf gradient based optimizer. For neural networks, the efficiency of this procedure can be further improved via the local reparametrization trick [89], which samples the neuron pre-activations instead of the weights themselves at each layer, leading to less variance in the stochastic gradients.

6.3 RELATED WORK

One of the earliest ideas and most direct approaches to tackle efficiency is pruning. Originally introduced by [101], pruning has recently been demonstrated to be applicable to modern architectures [59, 65]. It had been demonstrated that an overwhelming amount of up to 99,5% of parameters can be pruned in common architectures. There have been quite a few encouraging results obtained by (empirical) Bayesian approaches that employ weight pruning [23, 56, 130, 132, 185]. Nevertheless, weight pruning is in general inefficient for compression since the matrix format of the weights is not taken into consideration; as a result, Compressed Sparse Column (CSC) format has to be employed. Moreover, note that in conventional CNNs most flops are used by the convolution operation. Inspired by this observation, several authors proposed pruning schemes that take these considerations into account [192, 199] or even go as far as efficiency aware architectures to begin with [44, 79, 81]. From the Bayesian viewpoint, similar pruning schemes have been explored at [86, 98, 119, 134].

Given optimal architecture, NNs can further be compressed by quantization. More precisely, there are 2 common techniques. First, the set of accessible weights can be reduced drastically. As an extreme example, [36, 122, 149, 208] and [35] trained NN to use only binary or tertiary weights with floating point gradients. This approach however is in need of significantly more parameters than their ordinary counterparts. Work by [53] explores various techniques beyond binary quantization: k-means quantization, product quantiza-

2 In practice this term is a large constant determined by the weight precision.

tion and residual quantization. Later studies extent this set to optimal fixed point [106] and hashing quantization [31]. [65] apply k-means clustering and consequent center training. From a practical point of view, however, all these are fairly unpractical during test time. For the computation of each feature map in a net, the original weight matrix must be reconstructed from the indexes in the matrix and a codebook that contains all the original weights. This is an expensive operation and this is why some studies propose a different approach than set quantization. Precision quantization simply reduces the bit size per weight. This has a great advantage over set quantization at inference time since feature maps can simply be computed with less precision weights. Several studies show that this has little to no effect on network accuracy when using 16bit weights [29, 37, 61, 123, 186]. Somewhat orthogonal to the above discussion but certainly relevant are approaches that customize the implementation of CNNs for hardware limited devices[11, 79, 164].

6.4 BAYESIAN COMPRESSION WITH SCALE MIXTURES OF NORMALS

Consider the following prior over a parameter w where its' scale z is governed by a distribution $p(z)$:

$$z \sim p(z); \qquad w \sim \mathcal{N}(w; 0, z^2) \tag{75}$$

with z^2 serving as the variance of the zero-mean normal distribution over w. By treating the scales of w as random variables we can recover marginal prior distributions over the parameters that have heavier tails and more mass at zero; this subsequently biases the posterior distribution over w to be sparse. This family of distributions is known as scale-mixtures of normals [7, 17] and it is quite general, as a lot of well known sparsity inducing distributions are special cases.

One example of the aforementioned framework is the spike-and-slab distribution [127], the golden standard for sparse Bayesian inference. Under the spike-and-slab, the mixing density of the scales is a Bernoulli distribution, thus the marginal $p(w)$ has a delta "spike" at zero and a continuous "slab" over the real line. Unfortunately, this prior leads to a computationally expensive inference since we have to explore a space of 2^M models, where M is the number of the model parameters. Dropout [174], one of the most popular regularization techniques for neural networks, can be interpreted as positing a spike and slab distribution over the weights where the variance of the "slab" is zero [50, 107]. Another example is the Laplace distribution which arises by considering $p(z^2) = \text{Exp}(\lambda)$. The mode of the posterior distribution under a Laplace prior is known as the Lasso [181] estimator and has been previously used for sparsifying neural networks at [158, 192]. While computationally simple, the Lasso estimator is prone to "shrinking" large signals [28] and only provides point estimates about the parameters. As a result it does not provide uncertainty estimates, it can potentially overfit and, according to the bits-back argument, is inefficient for compression.

For these reasons, in this paper we will tackle the problem of compression and efficiency in neural networks by adopting a full Bayesian treatment and inferring a posterior distribution over the parameters under a scale mixture prior. We will consider two choices for

prior over the scales $p(z)$; the hyperparameter free log-uniform prior [48, 89] and the half-Cauchy prior, which results into a horseshoe [28] distribution. Both of these distributions correspond to a continuous relaxation of the spike-and-slab prior and we provide a brief discussion on their shrinkage properties at Appendix C.

6.4.1 *Reparametrizing variational dropout for group sparsity*

One potential choice for $p(z)$ is the improper log-uniform prior: $p(z) \propto |z|^{-1}$. It turns out that we can recover the log-uniform prior over the weights w if we marginalize over the scales z:

$$p(w) \propto \int \frac{1}{|z|}\mathcal{N}(w|0, z^2)dz = \frac{1}{|w|} \tag{76}$$

This alternative parametrization of the the log uniform prior is known in the statistics literature as the normal-Jeffreys prior and has been introduced by [48]. This formulation allows to “couple" the scales of weights that belong to the same group (e.g. neuron or feature map), by simply sharing the corresponding scale variable z in the joint prior:

$$p(\mathbf{W}, \mathbf{z}) \propto \prod_i^A \frac{1}{|z_i|} \prod_{ij}^{A,B} \mathcal{N}(w_{ij}|0, z_i^2) \tag{77}$$

where $\mathbf{W}$ is the weight matrix of a fully connected neural network layer with A being the dimensionality of the input and B the dimensionality of the output. Now consider performing variational inference with a joint posterior parametrized as follows:

$$q_\phi(\mathbf{W}, \mathbf{z}) = \prod_{i=1}^A \mathcal{N}(z_i|\mu_{z_i}, \mu_{z_i}^2\alpha_i) \prod_{i,j}^{A,B} \mathcal{N}(w_{ij}|z_i\mu_{ij}, z_i^2\sigma_{ij}^2) \tag{78}$$

where α_i is the dropout rate [89, 130, 174] of the given group. As explained at [89, 130], the multiplicative parametrization of the approximate posterior over $\mathbf{z}$ suffers from high variance gradients; therefore we will follow [130] and re-parametrize it in terms of $\sigma_{z_i}^2 = \mu_{z_i}^2\alpha_i$, hence optimize w.r.t. $\sigma_{z_i}^2$. The lower bound under this prior and posterior becomes:

$$\begin{aligned}\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{z})q_\phi(\mathbf{W}|\mathbf{z})}[\log p(\mathcal{D}|\mathbf{W})] - \mathbb{E}_{q_\phi(\mathbf{z})}[KL(q_\phi(\mathbf{W}|\mathbf{z})||p(\mathbf{W}|\mathbf{z}))] - \\ KL(q_\phi(\mathbf{z})||p(\mathbf{z}))\end{aligned} \tag{79}$$

Under this particular posterior parametrization the KL-divergence between the conditional prior $p(\mathbf{W}|\mathbf{z})$ and posterior $q_\phi(\mathbf{W}|\mathbf{z})$ is independent of $\mathbf{z}$:

$$KL(q_\phi(\mathbf{W}|\mathbf{z})||p(\mathbf{W}|\mathbf{z})) = \frac{1}{2}\sum_{i,j}^{A,B}\left(\log \frac{\cancel{z_i^2}}{\cancel{z_i^2}\sigma_{ij}^2} + \frac{\cancel{z_i^2}\sigma_{ij}^2}{\cancel{z_i^2}} + \frac{\cancel{z_i^2}\mu_{ij}^2}{\cancel{z_i^2}} - 1\right) \tag{80}$$

This independence can be better understood if we consider a non-centered parametrization of the prior [142]. More specifically, consider reparametrizing the weights as $\tilde{w}_{ij} = \frac{w_{ij}}{z_i}$;

this will then result into $p(\mathbf{W}|\mathbf{z})p(\mathbf{z}) = p(\tilde{\mathbf{W}})p(\mathbf{z})$, where $p(\tilde{\mathbf{W}}) = \prod_{i,j} \mathcal{N}(w_{ij}|0,1)$ and $\mathbf{W} = \text{diag}(\mathbf{z})\tilde{\mathbf{W}}$. Now if we perform variational inference under the $p(\tilde{\mathbf{W}})p(\mathbf{z})$ prior with a posterior of the form $q_\phi(\tilde{\mathbf{W}}, \mathbf{z}) = q_\phi(\tilde{\mathbf{W}})q_\phi(\mathbf{z})$, with $q_\phi(\tilde{\mathbf{W}}) = \mathcal{N}(\tilde{w}_{ij}|\mu_{ij}, \sigma^2_{ij})$, then we see that we arrive at the same expressions for the KL-divergence between the prior and the posterior. Finally, the KL-divergence between the normal-Jeffreys scale prior $p(\mathbf{z})$ and Gaussian posterior $q_\phi(\mathbf{z})$ depends only on the "implied" dropout rate, $\alpha_i = \sigma^2_{z_i}/\mu^2_{z_i}$, and takes the following form [130]:

$$KL(q_\phi(\mathbf{z})||p(\mathbf{z})) \approx \sum_i^A \left(k_1\sigma(k_2 + k_3 \log \alpha_i) - 0.5m(-\log \alpha_i) - k_1\right) \tag{81}$$

where $\sigma(\cdot)$, $m(\cdot)$ are the sigmoid and softplus functions respectively[3] and $k_1 = 0.63576$, $k_2 = 1.87320$, $k_3 = 1.48695$. We can now prune entire groups of parameters by simply specifying a threshold for the variational dropout rate of the corresponding group, e.g. $\log \alpha_i = (\log \sigma^2_{z_i} - \log \mu^2_{z_i}) \geqslant t$. It should be mentioned that this prior parametrization readily allows for a more flexible marginal posterior over the weights as we now have a compound distribution, $q_\phi(\mathbf{W}) = \int q_\phi(\mathbf{W}|\mathbf{z})q_\phi(\mathbf{z})d\mathbf{z}$; this is in contrast to the original parametrization and the Gaussian approximations employed by [89, 130]. At test time, in order to have a single feedforward pass we replace the distribution over $\mathbf{W}$ at each layer with a single weight matrix, the masked posterior mean:

$$\hat{\mathbf{W}} = \text{diag}(\mathbf{m}) \odot \mathbb{E}_{q(\mathbf{z})q(\tilde{\mathbf{W}})}[\text{diag}(\mathbf{z})\tilde{\mathbf{W}}] = \text{diag}\left(\mathbf{m} \odot \boldsymbol{\mu}_z\right)\mathbf{M}_W \tag{82}$$

where $\mathbf{m}$ is a binary mask determined according to the group variational dropout rate and $\mathbf{M}_W$ are the means of $q_\phi(\tilde{\mathbf{W}})$.

6.4.2 *Group horseshoe with half-Cauchy scale priors*

Another choice for $p(z)$ is a, proper, half-Cauchy: $\mathcal{C}^+(0,s) = 2(s\pi(1 + (z/s)^2))^{-1}$; it induces a horseshoe prior [28] distribution over the weights, which is a well known sparsity inducing prior in the statistics literature. More formally, the prior hierarchy over the weights is expressed as (in a non-centered parametrization):

$$s \sim \mathcal{C}^+(0,\tau_0); \quad \tilde{z}_i \sim \mathcal{C}^+(0,1); \quad \tilde{w}_{ij} \sim \mathcal{N}(0,1); \quad w_{ij} = \tilde{w}_{ij}\tilde{z}_i s \tag{83}$$

where τ_0 is the free parameter that can be tuned for specific desiderata. The idea behind the horseshoe is that of the "global-local" shrinkage; the global scale variable s pulls all of the variables towards zero whereas the heavy tailed local variables z_i can compensate and allow for some weights to escape. Instead of directly working with the half-Cauchy priors we will employ a decomposition of the half-Cauchy that relies upon (inverse) gamma distributions [136] as this will allow us to compute the KL-divergence between the scale prior $p(\mathbf{z})$ and a log-normal scale posterior $q_\phi(\mathbf{z})$ in closed form (the derivation is given

3 $\sigma(x) = (1 + \exp(-x))^{-1}$, $m(x) = \log(1 + \exp(x))$

in Appendix D). More specifically, we have that the half-Cauchy prior can be expressed in a non-centered parametrization as:

$$p(\tilde{\beta}) = \mathcal{IG}(0.5, 1); \qquad p(\tilde{\alpha}) = \mathcal{G}(0.5, k^2); \qquad z^2 = \tilde{\alpha}\tilde{\beta} \tag{84}$$

where $\mathcal{IG}(), \mathcal{G}()$ correspond to the inverse Gamma, Gamma distributions according to the scale parametrization, and z follows a half-Cauchy distribution with scale k. Therefore we will re-express the whole hierarchy as:

$$\begin{aligned} s_b \sim \mathcal{IG}(0.5, 1); \quad s_a \sim \mathcal{G}(0.5, \tau_0^2); \quad \tilde{\beta}_i \sim \mathcal{IG}(0.5, 1); \quad \tilde{\alpha}_i \sim \mathcal{G}(0.5, 1); \\ \tilde{w}_{ij} \sim \mathcal{N}(0, 1); \qquad w_{ij} = \tilde{w}_{ij}\sqrt{s_a s_b \tilde{\alpha}_i \tilde{\beta}_i} \end{aligned} \tag{85}$$

It should be mentioned that the improper log-uniform prior is the limiting case of the horseshoe prior when the shapes of the (inverse) Gamma hyperpriors on $\tilde{\alpha}_i, \tilde{\beta}_i$ go to zero [28]. In fact, several well known shrinkage priors can be expressed in this form by altering the shapes of the (inverse) Gamma hyperpriors [8]. For the variational posterior we will employ the following mean field approximation:

$$q_\phi(s_b, s_a, \tilde{\boldsymbol{\beta}}) = \mathcal{LN}(s_b|\mu_{s_b}, \sigma^2_{s_b})\mathcal{LN}(s_a|\mu_{s_a}, \sigma^2_{s_a}) \prod_i^A \mathcal{LN}(\tilde{\beta}_i|\mu_{\tilde{\beta}_i}, \sigma^2_{\tilde{\beta}_i}) \tag{86}$$

$$q_\phi(\tilde{\boldsymbol{\alpha}}, \tilde{\mathbf{W}}) = \prod_i^A \mathcal{LN}(\tilde{\alpha}_i|\mu_{\tilde{\alpha}_i}, \sigma^2_{\tilde{\alpha}_i}) \prod_{i,j}^{A,B} \mathcal{N}(\tilde{w}_{ij}|\mu_{\tilde{w}_{ij}}, \sigma^2_{\tilde{w}_{ij}}) \tag{87}$$

where $\mathcal{LN}(\cdot, \cdot)$ is a log-normal distribution. Furthermore, notice that we can also apply local reparametrizations [89] when we are sampling $\sqrt{\tilde{\alpha}_i\tilde{\beta}_i}$ and $\sqrt{s_a s_b}$ by exploiting properties of the log-normal distribution[4] and thus forming the implied:

$$\tilde{z}_i = \sqrt{\tilde{\alpha}_i\tilde{\beta}_i} \sim \mathcal{LN}(\mu_{\tilde{z}_i}, \sigma^2_{\tilde{z}_i}); \qquad s = \sqrt{s_a s_b} \sim \mathcal{LN}(\mu_s, \sigma^2_s) \tag{88}$$

$$\mu_{\tilde{z}_i} = \frac{1}{2}(\mu_{\tilde{\alpha}_i} + \mu_{\tilde{\beta}_i}); \qquad \sigma^2_{\tilde{z}_i} = \frac{1}{4}(\sigma^2_{\tilde{\alpha}_i} + \sigma^2_{\tilde{\beta}_i}); \tag{89}$$

$$\mu_s = \frac{1}{2}(\mu_{s_a} + \mu_{s_b}); \qquad \sigma^2_s = \frac{1}{4}(\sigma^2_{s_a} + \sigma^2_{s_b}) \tag{90}$$

As a threshold rule for group pruning we will use the negative log-mode[5] of the local log-normal r.v. $z_i = s\tilde{z}_i$:

$$p(z_i) = \mathcal{LN}(z_i|\mu_{z_i}, \sigma^2_{z_i}); \qquad \mu_{z_i} = \mu_{\tilde{z}_i} + \mu_s; \qquad \sigma^2_{z_i} = \sigma^2_{\tilde{z}_i} + \sigma^2_s \tag{91}$$

i.e. prune when $(\sigma^2_{z_i} - \mu_{z_i}) \geqslant t$. This ignores dependencies among the z_i elements induced by the common scale s, but nonetheless we found that it works well in practice. Similarly

4 The product of log-normal r.v.s is another log-normal and a power of a log-normal r.v. is another log-normal.

5 We empirically found that it has similar behavior to the negative log-mean, $-(\mu_{z_i} + \frac{1}{2}\sigma^2_{z_i})$, albeit slightly better separating the scales.

with the group normal-Jeffreys prior, we will replace the distribution over $\mathbf{W}$ at each layer with the masked posterior mean during test time:

$$\begin{aligned}\hat{\mathbf{W}} &= \text{diag}(\mathbf{m}) \odot \mathbb{E}_{q(\mathbf{z})q(\tilde{\mathbf{W}})}[\text{diag}(\mathbf{z})\tilde{\mathbf{W}}] \\ &= \text{diag}\left(\mathbf{m} \odot \exp(\boldsymbol{\mu}_z + \frac{1}{2}\boldsymbol{\sigma}_z^2)\right)\mathbf{M}_W \end{aligned} \tag{92}$$

where $\mathbf{m}$ is a binary mask determined according to the aforementioned threshold, $\mathbf{M}_W$ are the means of $q(\tilde{\mathbf{W}})$ and $\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2$ are the means and variances of the local log-normals over z_i.

6.5 EXPERIMENTS

For our experimental evaluation, we measure the compression achieved by our methods along with the speed-up that they provide on several neural network architectures and datasets; the LeNet-300-100 [99], LeNet-5-Caffe[6] on MNIST [100], along with VGG [166][7] on CIFAR 10 [95]. We realized the group of variables by tying the scale variables for each output channel for convolutional layers and for each input neuron for the fully connected layers. We provide the algorithms for the forward pass in the appendix. As for other hyperparameters; we set the scale of the global half-Cauchy prior τ_0 to a small value, i.e., $\tau_0 = 1e-5$, in order to increase the prior mass at zero and further encourage sparse models. Furthermore, we constrained the posterior variance [114] and used "warm-up" [170] in order to facilitate for better optimization that avoids the bad local optima of the variational objective. We provide in the appendix further details about the rest of the experimental setup along with a sample visualization of the pruning thresholds. In the latter it is easy to see that two well separated clusters are formed characterizing the signal and noise, thus making easy the determination of the appropriate cut-off point.

6.5.1 *Architecture learning & bit precisions*

We will first demonstrate the group sparsity capabilities of our methods by illustrating the learned architectures at Table 8. We also provide the inferred bit precision per layer for the Bayesian methods, which we obtain by using the marginal posterior variances of the weights as a proxy[8]:

$$\mathbb{V}(w_{ij})_{\text{NJ}} = \sigma_{z_i}^2\left(\sigma_{ij}^2 + \mu_{ij}^2\right) + \sigma_{ij}^2\mu_{z_i}^2 \tag{93}$$

$$\begin{aligned}\mathbb{V}(w_{ij})_{\text{HS}} = (\exp(\sigma_{z_i}^2) - 1)\exp(2\mu_{z_i} + \sigma_{z_i}^2)\left(\sigma_{ij}^2 + \mu_{ij}^2\right) + \\ \sigma_{ij}^2\exp(2\mu_{z_i} + \sigma_{z_i}^2)\end{aligned} \tag{94}$$

6 https://github.com/BVLC/caffe/tree/master/examples/mnist

7 We use the adaptation for CIFAR 10 described at http://torch.ch/blog/2015/07/30/cifar.html.

8 Due to the non-centered parametrization it is easy to compute: $\mathbb{V}(w_{ij}) = \mathbb{V}(z_i\tilde{w}_{ij}) = \mathbb{V}(z_i)\left(\mathbb{E}[\tilde{w}_{ij}]^2 + \mathbb{V}(\tilde{w}_{ij})\right) + \mathbb{V}(\tilde{w}_{ij})\mathbb{E}[z_i]^2$.

We used the mean variance across a layer to compute the unit round off necessary to represent weights. This method will give us the amount of significant bits, we furthermore add 3 exponent and 1 sign bit to compute the final bit precision per layer. As we can observe, our methods infer significantly smaller architectures for the LeNet-300-100 and LeNet-5-Caffe, compared to Sparse Variational Dropout (VD) [130], Generalized Dropout (GD) [171] and Group Lasso (GL) [192]. Interestingly, we observe that for the VGG network almost all of big 512 feature map layers are drastically reduced to around 10 feature maps whereas the initial layers are mostly kept intact. Furthermore, all of the Bayesian methods considered require far fewer than the standard 32 bits per-layer to represent the weights, sometimes even allowing for 5 bit precisions.

Table 8: Learned architectures with Sparse VD [130], Generalized Dropout (GD) [171] and Group Lasso (GL) [192]. Bayesian Compression (BC) with group normal-Jeffreys (GNJ) and group horseshoe (GHS) priors correspond to the proposed models. We show the amount of neurons left after pruning along with the average bit precisions for the weights at each layer.

Network & size	Method	Pruned architecture	Bit-precision
LeNet-300-100	Sparse VD	512-114-72	8-11-14
784-300-100	BC-GNJ	278-98-13	8-9-14
	BC-GHS	311-86-14	13-11-10
LeNet-5-Caffe	Sparse VD	14-19-242-131	13-10-8-12
	GD	7-13-208-16	-
20-50-800-500	GL	3-12-192-500	-
	BC-GNJ	8-13-88-13	18-10-7-9
	BC-GHS	5-10-76-16	10-10-14-13
VGG	BC-GNJ	63-64-128-128-245-155-63--26-24-20-14-12-11-11-15	10-10-10-10-8-8-8--5-5-5-5-5-6-7-11
(2× 64)-(2× 128)-(3×256)-(8× 512)	BC-GHS	51-62-125-128-228-129-38--13-9-6-5-6-6-6-20	11-12-9-14-10-8-5--5-6-6-6-8-11-17-10

6.5.2 *Compression Rates*

In order to measure the compression rates obtained by our algorithms we consider three different scenarios. For the first scenario, we measure the compression achieved by only removing the groups of weights that have been pruned and keeping everything else on full precision. For baselines that performed unstructured instead of group pruning, we measure the compression under a Compressed Sparse Column (CSC) format to store the parameters. This scenario is interesting, in that it can directly be applied to existing frameworks such as Tensorflow [1]. For the second scenario, which we name as "fast prediction", in addition

to pruning, we also reduce the bit precision of the weights for each layer independently. This can simultaneously decrease the size of the network to be stored and improve its computational efficiency. This scenario is relevant, since upcoming hardware can facilitate for mixed precision computation [62, 105]. For the third and final scenario, which we name as "maximum compression", we consider the scheme described at [65], by performing K-means with a K=32 on the neural network weights and thus storing only the centroids and centroid assignment for each weight. While this scheme leads to the best compression, it is not as practical since one needs to decompress the weight matrix at each layer when making predictions.

Table 9: Compression results for our methods. "DC" corresponds to Deep Compression method introduced at [65], "DNS" to the method of [59] and "SWS" to the Soft-Weight Sharing of [185]. Numbers marked with * are best case guesses.

			Compression Rates (Error %)		
Model Original Error %	Method	$\frac{\lvert\mathbf{w}\neq 0\rvert}{\lvert\mathbf{w}\rvert}\%$	Pruning	Fast Prediction	Maximum Compression
LeNet-300-100	DC	8.0	6 (1.6)	-	40 (1.6)
	DNS	1.8	28* (2.0)	-	-
1.6	SWS	4.3	12* (1.9)	-	64(1.9)
	Sparse VD	2.2	21(1.8)	84(1.8)	113 (1.8)
	BC-GNJ	10.8	9(1.8)	36(1.8)	58(1.8)
	BC-GHS	10.6	9(1.8)	23(1.9)	59(2.0)
LeNet-5-Caffe	DC	8.0	6*(0.7)	-	39(0.7)
	DNS	0.9	55*(0.9)	-	108(0.9)
0.9	SWS	0.5	100*(1.0)	-	162(1.0)
	Sparse VD	0.7	63(1.0)	228(1.0)	365(1.0)
	BC-GNJ	0.9	108(1.0)	361(1.0)	573(1.0)
	BC-GHS	0.6	156(1.0)	419(1.0)	771(1.0)
VGG	BC-GNJ	6.7	14(8.6)	56(8.8)	95(8.6)
8.4	BC-GHS	5.5	18(9.0)	59(9.0)	116(9.2)

The compression results can be seen at Table 9. Overall, both BC-GNJ and BC-GHS offer competitive accuracy vs compression trade-offs compared to the state-of-the-art. It should be mentioned that group sparsity and unstructured sparsity are not mutually-exclusive, so in principle we could combine our method with weight sparsity and further improve our results on, e.g., LeNet-300-100. To obtain the VGG results, we started by initializing the posterior means from a "pretrained" network, similarly to what was performed at [130], as training from a random initialization resulted into overall lower accu-

racy (∼1%-2% less). After this initialization step, we trained the network normally for 200 epochs with the Adam optimizer and its default hyperparameters. We observe that for this network, deterministic predictions yield slightly higher error compared to sampling the posterior distribution (∼0.2%-0.4% higher) and averaging, but we report the performance with the former for consistency.

6.5.3 *Speed and energy consumption*

In order to further show that group pruning from our methods results into networks that are faster and more energy efficient, we measure the time and energy consumption of the pruned LeNet-5-Caffe network when performing a forward pass with a large batch of 8192 examples. The measurements we report are averages over 10^4 forward passes and were run with Tensorflow 1.0.1, Cuda 8.0 and its respective cuDNN. We report results on both CPU and a Titan X GPU. We compare the network obtained from our methods with those obtained by performing group Lasso, denoted as GL, at [192]. As we can see at figure 17, moving from a CPU to a GPU yields the largest speedup offset (as expected), and applying our method on top can yield speed-up factors of ∼8×, while also reducing the energy consumption by 3×. This effect can be even more dramatic for larger networks; for the VGG network with a batch size of 256 we observe a speed-up factor of 51× compared to running the original network on CPU.

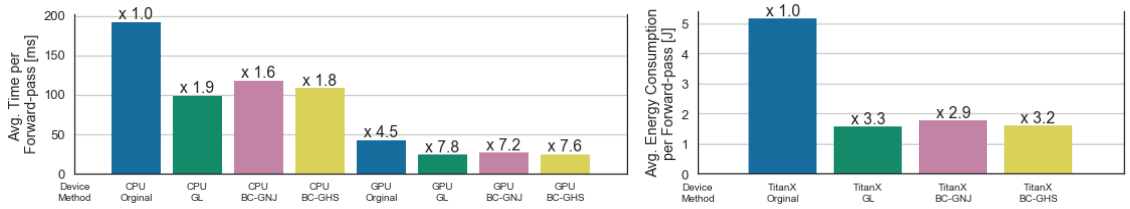


Figure 17: **Left:** Avg. Time a batch of 8192 samples takes to pass through LeNet-5-Caffe. Numbers on top of the bars represent speed-up factor relative to the CPU implementation of the original network. **Right:** Energy consumption of the GPU of the same process (when run on GPU).

6.6 CONCLUSION

We introduced Bayesian compression, a way to tackle efficiency and compression in deep neural networks in a unified and principled way. Our proposed methods allow for theoretically principled compression of neural networks, improved energy efficiency with reduced computation while naturally learning the bit precisions for each weight. This serves as a strong argument in favor of Bayesian methods for neural networks, when we are concerned with compression and speed up.

7

LEARNING SPARSE NEURAL NETWORKS THROUGH L_0 REGULARIZATION

In the previous chapter we showed how through specific choices of the prior distributions over the parameters, Bayesian inference can jointly provide sparse and quantized models. In this chapter, we decouple those two objectives and take a closer look on sparsity alone, in a way that allows for more fine grained and direct control compared to the Bayesian equivalent. For this reason, we propose a practical method for L_0 norm regularization for neural networks: pruning the network during training by encouraging weights to become exactly zero. Such regularization is interesting since (1) it can greatly speed up training and inference, and (2) it can improve generalization. AIC and BIC, well-known model selection criteria, are special cases of L_0 regularization. However, since the L_0 norm of weights is non-differentiable, we cannot incorporate it directly as a regularization term in the objective function. We propose a solution through the inclusion of a collection of non-negative stochastic gates, which collectively determine which weights to set to zero. We show that, somewhat surprisingly, for certain distributions over the gates, the expected L_0 regularized objective is differentiable with respect to the distribution parameters. We further propose the *hard concrete* distribution for the gates, which is obtained by "stretching" a binary concrete distribution and then transforming its samples with a hard-sigmoid. The parameters of the distribution over the gates can then be jointly optimized with the original network parameters. As a result our method allows for straightforward and efficient learning of model structures with stochastic gradient descent and allows for conditional computation in a principled way. We perform various experiments to demonstrate the effectiveness of the resulting approach and regularizer. [1]

7.1 INTRODUCTION

Deep neural networks are flexible function approximators that have been very successful in a broad range of tasks. They can easily scale to millions of parameters while allowing for tractable optimization with mini-batch stochastic gradient descent (SGD), graphical processing units (GPUs) and parallel computation. Nevertheless they do have drawbacks. Firstly, it has been shown in recent works [65, 130, 185] that they are greatly overparametrized as they can be pruned significantly without any loss in accuracy; this exhibits unnecessary computation and resources. Secondly, they can easily overfit and even mem-

1 This chapter has been adapted from our publication [115].

orize random patterns in the data [203], if not properly regularized. This overfitting can lead to poor generalization in practice.

A way to address both of these issues is by employing model compression and sparsification techniques. By sparsifying the model, we can avoid unnecessary computation and resources, since irrelevant degrees of freedom are pruned away and do not need to be computed. Furthermore, we reduce its complexity, thus penalizing memorization and alleviating overfitting.

A conceptually attractive approach is the L_0 norm regularization of (blocks of) parameters; this explicitly penalizes parameters for being different than zero with no further restrictions. However, the combinatorial nature of this problem makes for an intractable optimization for large models.

In this paper we propose a general framework for surrogate L_0 regularized objectives. It is realized by smoothing the *expected* L_0 regularized objective with continuous distributions in a way that can maintain the *exact* zeros in the parameters while still allowing for efficient gradient based optimization. This is achieved by transforming continuous random variables (r.v.s) with a hard nonlinearity, the hard-sigmoid. We further propose and employ a novel distribution obtained by this procedure; the hard concrete. It is obtained by "stretching" a binary concrete random variable [85, 120] and then passing its samples through a hard-sigmoid. We demonstrate the effectiveness of this simple procedure in various experiments.

7.2 MINIMIZING THE L_0 NORM OF PARAMETRIC MODELS

One way to sparsify parametric models, such as deep neural networks, with the least assumptions about the parameters is the following; let $\mathcal{D}$ be a dataset consisting of N i.i.d. input output pairs $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and consider a regularized empirical risk minimization procedure with an L_0 regularization on the parameters θ of a hypothesis (e.g. a neural network) $h(\cdot; \theta)$[2]:

$$\mathcal{R}(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\big(h(\mathbf{x}_i; \theta), \mathbf{y}_i\big)\right) + \lambda\|\theta\|_0, \qquad \|\theta\|_0 = \sum_{j=1}^{|\theta|} \mathbb{I}[\theta_j \neq 0], \tag{95}$$

$$\theta^* = \arg\min_{\theta}\{\mathcal{R}(\theta)\},$$

where $|\theta|$ is the dimensionality of the parameters, λ is a weighting factor for the regularization and $\mathcal{L}(\cdot)$ corresponds to a loss function, e.g. cross-entropy loss for classification or mean-squared error for regression. The L_0 norm penalizes the number of non-zero entries of the parameter vector and thus encourages sparsity in the final estimates θ^*. The Akaike Information Criterion (AIC) [5] and the Bayesian Information Criterion (BIC) [159], well-known model selection criteria, correspond to specific choices of λ. Notice that the L_0 norm induces no shrinkage on the actual values of the parameters θ; this is in contrast to

2 This assumption is just for ease of explanation; our proposed framework can be applied to any objective function involving parameters.

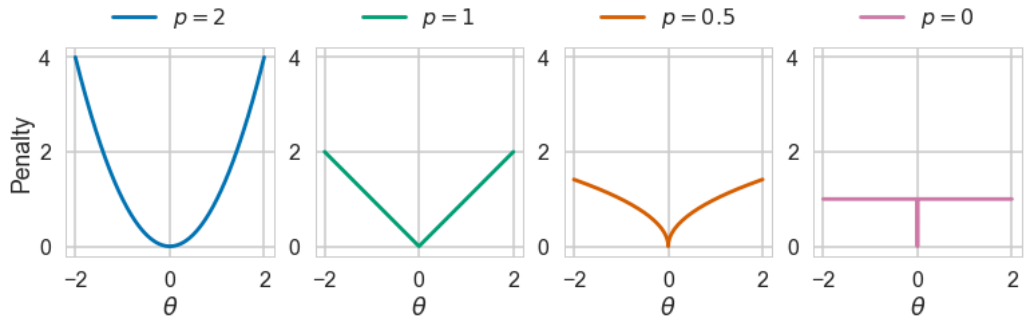


Figure 18: L_p norm penalties for a parameter θ according to different values of p. It is easily observed that both weight decay and Lasso, $p = 2$ and $p = 1$ respectively, impose shrinkage for large values of θ. By gradually allowing $p < 1$ we observe that the shrinkage is reduced and at the limit of $p = 0$ we observe that the penalty is a constant for $\theta \neq 0$.

e.g. L_1 regularization and the Lasso [181], where the sparsity is due to shrinking the actual values of θ. We provide a visualization of this effect in Figure 18.

Unfortunately, optimization under this penalty is computationally intractable due to the non-differentiability and combinatorial nature of $2^{|\theta|}$ possible states of the parameter vector θ. How can we relax the discrete nature of the L_0 penalty such that we allow for efficient continuous optimization of Eq. 95, while allowing for exact zeros in the parameters? This section will present the necessary details of our approach.

7.2.1 *A general recipe for efficiently minimizing* L_0 *norms*

Consider the L_0 norm under a simple re-parametrization of θ:

$$\theta_j = \tilde{\theta}_j z_j, \qquad z_j \in \{0, 1\}, \qquad \tilde{\theta}_j \neq 0, \qquad \|\theta\|_0 = \sum_{j=1}^{|\theta|} z_j, \tag{96}$$

where the z_j correspond to binary "gates" that denote whether a parameter is present and the L_0 norm corresponds to the amount of gates being "on". By letting $q(z_j|\pi_j) = \text{Bern}(\pi_j)$ be a Bernoulli distribution over each gate z_j we can reformulate the minimization of Eq. 95 as penalizing the number of parameters being used, on average, as follows:

$$\mathcal{R}(\tilde{\theta}, \boldsymbol{\pi}) = \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\pi})}\left[\frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\left(h(\mathbf{x}_i; \tilde{\theta} \odot \mathbf{z}), \mathbf{y}_i\right)\right)\right] + \lambda \sum_{j=1}^{|\theta|} \pi_j, \tag{97}$$

$$\tilde{\theta}^*, \boldsymbol{\pi}^* = \arg\min_{\tilde{\theta}, \boldsymbol{\pi}} \{\mathcal{R}(\tilde{\theta}, \boldsymbol{\pi})\},$$

where $\odot$ corresponds to the elementwise product. The objective described in Eq. 97 is in fact a special case of a variational bound over the parameters involving spike and slab [127] priors and approximate posteriors; we refer interested readers to appendix 10.5.1.

Now the second term of the r.h.s. of Eq. 97 is straightforward to minimize however the first term is problematic for $\boldsymbol{\pi}$ due to the discrete nature of $\mathbf{z}$, which does not allow for efficient gradient based optimization. While in principle a gradient estimator such

as the REINFORCE [195] could be employed, it suffers from high variance and control variates [128, 129, 184], that require auxiliary models or multiple evaluations of the network, have to be employed. Two simpler alternatives would be to use either the straight-through [19] estimator as done at [172] or the concrete distribution as e.g. at [51]. Unfortunately both of these approach have drawbacks; the first one provides biased gradients due to ignoring the Heaviside function in the likelihood during the gradient evaluation whereas the second one does not allow for the gates (and hence parameters) to be exactly zero during optimization, thus precluding the benefits of conditional computation [19].

Fortunately, there is a simple alternative way to smooth the objective such that we allow for efficient gradient based optimization of the expected L_0 norm along with zeros in the parameters θ. Let $\mathbf{s}$ be a continuous random variable with a distribution $q(\mathbf{s})$ that has parameters $\boldsymbol{\phi}$. We can now let the gates $\mathbf{z}$ be given by a hard-sigmoid rectification of $\mathbf{s}$[3], as follows:

$$\mathbf{s} \sim q(\mathbf{s}|\boldsymbol{\phi}) \tag{98}$$

$$\mathbf{z} = \min(\mathbf{1}, \max(\mathbf{0}, \mathbf{s})). \tag{99}$$

This would then allow the gate to be *exactly* zero and, due to the underlying continuous random variable $\mathbf{s}$, we can still compute the probability of the gate being non-zero (active). This is easily obtained by the cumulative distribution function (CDF) $Q(\cdot)$ of $\mathbf{s}$:

$$q(\mathbf{z} \neq 0|\boldsymbol{\phi}) = 1 - Q(\mathbf{s} \leqslant 0|\boldsymbol{\phi}), \tag{100}$$

i.e. it is the probability of the $\mathbf{s}$ variable being positive. We can thus smooth the binary Bernoulli gates $\mathbf{z}$ appearing in Eq. 97 by employing continuous distributions in the aforementioned way:

$$\begin{aligned} \mathcal{R}(\tilde{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q(\mathbf{s}|\boldsymbol{\phi})}\left[\frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\left(h(\mathbf{x}_i; \tilde{\theta} \odot g(\mathbf{s})), \mathbf{y}_i\right)\right)\right] + \\ \lambda \sum_{j=1}^{|\theta|} \left(1 - Q(s_j \leqslant 0|\phi_j)\right), \\ \tilde{\theta}^*, \boldsymbol{\phi}* = \arg\min_{\tilde{\theta}, \boldsymbol{\phi}} \{\mathcal{R}(\tilde{\theta}, \boldsymbol{\phi})\}, \quad g(\cdot) = \min(1, \max(0, \cdot)). \end{aligned} \tag{101}$$

Notice that this is a close surrogate to the original objective function in Eq. 97, as we similarly have a cost that explicitly penalizes the probability of a gate being different from zero. Now for continuous distributions $q(\mathbf{s})$ that allow for the reparameterization trick [91, 151] we can express the objective in Eq. 101 as an expectation over a parameter free

3 We chose to employ a hard-sigmoid instead of a rectifier, $g(\cdot) = \max(0, \cdot)$, so as to have the variable z better mimic a binary gate (rather than a scale variable).

noise distribution $p(\boldsymbol{\epsilon})$ and a deterministic and differentiable transformation $f(\cdot)$ of the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\epsilon}$:

$$\mathcal{R}(\tilde{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\left(h(\mathbf{x}_i; \tilde{\theta} \odot g(f(\boldsymbol{\phi}, \boldsymbol{\epsilon}))), \mathbf{y}_i\right)\right)\right] + \lambda \sum_{j=1}^{|\theta|} \left(1 - Q(s_j \leqslant 0 | \phi_j)\right), \tag{102}$$

which allows us to make the following Monte Carlo approximation to the (generally) intractable expectation over the noise distribution $p(\boldsymbol{\epsilon})$:

$$\begin{aligned}\hat{\mathcal{R}}(\tilde{\theta}, \boldsymbol{\phi}) &= \frac{1}{L}\sum_{l=1}^{L}\left(\frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\left(h(\mathbf{x}_i; \tilde{\theta} \odot \mathbf{z}^{(l)}), \mathbf{y}_i\right)\right)\right) + \lambda \sum_{j=1}^{|\theta|} \left(1 - Q(s_j \leqslant 0 | \phi_j)\right) \\ &= \mathcal{L}_E(\tilde{\theta}, \boldsymbol{\phi}) + \lambda \mathcal{L}_C(\boldsymbol{\phi}), \\ &\text{where } \mathbf{z}^{(l)} = g(f(\boldsymbol{\phi}, \boldsymbol{\epsilon}^{(l)})) \text{ and } \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon}).\end{aligned} \tag{103}$$

$\mathcal{L}_E$ corresponds to the *error loss* that measures how well the model is fitting the current dataset whereas $\mathcal{L}_C$ refers to the *complexity loss* that measures the flexibility of the model. Crucially, the total cost in Eq. 103 is now differentiable w.r.t. $\boldsymbol{\phi}$, thus enabling for efficient stochastic gradient based optimization, while still allowing for exact zeros at the parameters. One price we pay is that now the gradient of the log-likelihood w.r.t. the parameters $\boldsymbol{\phi}$ of $q(\mathbf{s})$ is sparse due to the rectifications; nevertheless this should not pose an issue considering the prevalence of rectified linear units in neural networks. Furthermore, due to the stochasticity at $\mathbf{s}$ the hard-sigmoid gate $\mathbf{z}$ is smoothed to a soft version on average, thus allowing for gradient based optimization to succeed, even when the mean of $\mathbf{s}$ is negative or larger than one. An example visualization can be seen in Figure 19b. It should be noted that a similar argument was also shown at [19], where with logistic noise a rectifier nonlinearity was smoothed to a softplus[4] on average.

7.2.2 *The hard concrete distribution*

The framework described in Section 7.2.1 gives us the freedom to choose an appropriate smoothing distribution $q(\mathbf{s})$. A choice that seems to work well in practice is the following; assume that we have a binary concrete [85, 120] random variable s distributed in the $(0, 1)$ interval with probability density $q_s(s|\phi)$ and cumulative density $Q_s(s|\phi)$. The parameters of the distribution are $\phi = (\log \alpha, \beta)$, where $\log \alpha$ is the location and β is the temperature. We can "stretch" this distribution to the (γ, ζ) interval, with $\gamma < 0$ and $\zeta > 1$, and then apply a hard-sigmoid on its random samples:

4 $f(x) = \log(1 + \exp(x))$.

$$u \sim \mathcal{U}(0,1), \quad s = \text{Sigmoid}\big((\log u - \log(1-u) + \log \alpha)/\beta\big), \tag{104}$$

$$\bar{s} = s(\zeta - \gamma) + \gamma, \quad z = \min(1, \max(0, \bar{s})). \tag{105}$$

This would then induce a distribution where the probability mass of $q_{\bar{s}}(\bar{s}|\phi)$ on the negative values, $Q_{\bar{s}}(0|\phi)$, is "folded" to a delta peak at zero, the probability mass on values larger than one, $1 - Q_{\bar{s}}(1|\phi)$, is "folded" to a delta peak at one and the original distribution $q_{\bar{s}}(\bar{s}|\phi)$ is truncated to the (0, 1) range. We provide more information and the density of the resulting distribution at the appendix.

Notice that a similar behavior would have been obtained even if we passed samples from any other distribution over the real line through a hard-sigmoid. The only requirement of the approach is that we can evaluate the CDF of $\bar{s}$ at 0 and 1. The main reason for picking the binary concrete is its close ties with Bernoulli r.v.s. It was originally proposed at [85, 120] as a smooth approximation to Bernoulli r.vs, a fact that allows for gradient based optimization of its parameters through the reparametrization trick. The temperature β controls the degree of approximation, as with $\beta = 0$ we can recover the original Bernoulli r.v. (but lose the differentiable properties) whereas with $0 < \beta < 1$ we obtain a probability density that concentrates its mass near the endpoints (e.g. as shown in Figure 19a). As a result, the hard concrete also inherits the same theoretical properties w.r.t. the Bernoulli distribution. Furthermore, it can serve as a better approximation of the discrete nature, since it includes $\{0, 1\}$ in its support, while still allowing for (sub)gradient optimization of its parameters due to the continuous probability mass that connects those two values. We can also view this distribution as a "rounded" version of the original binary concrete, where values larger than $\frac{1-\gamma}{\zeta-\gamma}$ are rounded to one whereas values smaller than $\frac{-\gamma}{\zeta-\gamma}$ are rounded to zero. We provide an example visualization of the hard concrete distribution in Figure 19a.

The L_0 complexity loss of the objective in Eq. 103 under the hard concrete r.v. is conveniently expressed as follows:

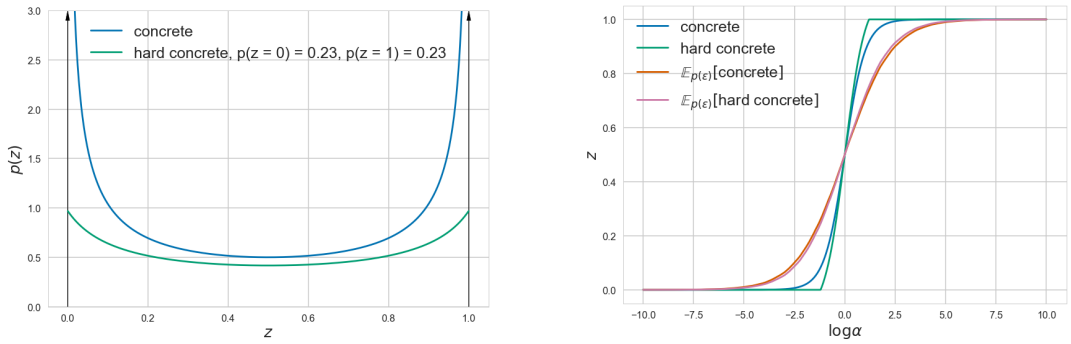
$$\mathcal{L}_C = \sum_{j=1}^{|\theta|} \big(1 - Q_{\bar{s}_j}(0|\phi)\big) = \sum_{j=1}^{|\theta|} \text{Sigmoid}\big(\log \alpha_j - \beta \log \frac{-\gamma}{\zeta}\big). \tag{106}$$

At test time we use the following estimator for the final parameters θ^* under a hard concrete gate:

$$\hat{\mathbf{z}} = \min(\mathbf{1}, \max(\mathbf{0}, \text{Sigmoid}(\log \boldsymbol{\alpha})(\zeta - \gamma) + \gamma)), \qquad \theta^* = \tilde{\theta}^* \odot \hat{\mathbf{z}}. \tag{107}$$

7.2.3 *Combining the* L_0 *norm with other norms*

While the L_0 norm leads to sparse estimates without imposing any shrinkage on θ it might still be desirable to impose some form of prior assumptions on the values of θ with alternative norms, e.g. impose smoothness with the L_2 norm (i.e. weight decay). In the following



(a) (b)

Figure 19: **(a)** The binary concrete distribution with location $\log \alpha = 0$ and temperature $\beta = 0.5$ and the hard concrete equivalent distribution obtained by stretching the concrete distribution to $(\gamma = -0.1, \zeta = 1.1)$ and then applying a hard-sigmoid. Under this specification the hard concrete distribution assigns, roughly, half of its mass to $\{0, 1\}$ and the rest to $(0, 1)$. **(b)** The expected value of the afforementioned concrete and hard concrete gate as a function of the location $\log \alpha$, obtained by averaging 10000 samples. We also added the value of the gates obtained by removing the noise entirely. We can see that the noise smooths the hard-sigmoid to a sigmoid on average.

we will show how this combination is feasible for the L_2 norm. The expected L_2 norm under the Bernoulli gating mechanism can be conveniently expressed as:

$$\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\pi})}\left[\|\theta\|_2^2\right] = \sum_{j=1}^{|\theta|} \mathbb{E}_{q(z_j|\pi_j)}\left[z_j^2 \tilde{\theta}_j^2\right] = \sum_{j=1}^{|\theta|} \pi_j \tilde{\theta}_j^2, \tag{108}$$

where π_j corresponds to the success probability of the Bernoulli gate z_j. To maintain a similar expression with our smoothing mechanism, and avoid extra shrinkage for the gates z_j, we can take into account that the standard L_2 norm penalty is proportional to the negative log density of a zero mean Gaussian prior with a standard deviation of $\sigma = 1$. We will then assume that the σ for each θ is governed by z in a way that when $z = 0$ we have that $\sigma = 1$ and when $z > 0$ we have that $\sigma = z$. As a result, we can obtain the following expression for the L_2 penalty (where $\hat{\theta} = \frac{\theta}{\sigma}$):

$$\begin{aligned}
\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})}\left[\|\hat{\theta}\|_2^2\right] &= \sum_{j=1}^{|\theta|} \Bigg(Q_{\bar{s}_j}(0|\phi_j)\frac{0}{1} + \\
&\qquad \left(1 - Q_{\bar{s}_j}(0|\phi_j)\right) \mathbb{E}_{q(z_j|\phi_j, \bar{s}_j > 0)}\left[\frac{\tilde{\theta}_j^2 z_j^2}{z_j^2}\right] \Bigg) \\
&= \sum_{j=1}^{|\theta|} \left(1 - Q_{\bar{s}_j}(0|\phi_j)\right) \tilde{\theta}_j^2.
\end{aligned} \tag{109}$$

7.2.4 *Group sparsity under an* L_0 *norm*

For reasons of computational efficiency it is usually desirable to perform group sparsity instead of parameter sparsity, as this can allow for practical computation savings. For example, in neural networks speedups can be obtained by employing a dropout [174] like procedure with neuron sparsity in fully connected layers or feature map sparsity for convolutional layers [112, 135, 192]. This is straightforward to do with hard concrete gates; simply share the gate between all of the members of the group. The expected L_0 and, according to section 7.2.3, L_2 penalties in this scenario can be rewritten as:

$$\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\Phi})}\Big[\|\theta\|_0\Big] = \sum_{g=1}^{|G|} |g| \Big(1 - Q(s_g \leqslant 0|\phi_g)\Big) \tag{110}$$

$$\mathbb{E}_{q(\mathbf{z}|\boldsymbol{\Phi})}\Big[\|\hat{\theta}\|_2^2\Big] = \sum_{g=1}^{|G|} \Big(\big(1 - Q(s_g \leqslant 0|\phi_g)\big) \sum_{j=1}^{|g|} \tilde{\theta}_j^2\Big). \tag{111}$$

where $|G|$ corresponds to the number of groups and $|g|$ corresponds to the number of parameters of group g. For all of our subsequent experiments we employed neuron sparsity, where we introduced a gate per input neuron for fully connected layers and a gate per output feature map for convolutional layers. Notice that in the interpretation we adopt the gate is shared across all locations of the feature map for convolutional layers, akin to spatial dropout [183]. This can lead to practical computation savings while training, a benefit which is not possible with the commonly used independent dropout masks per spatial location (e.g. as at [202]).

7.3 RELATED WORK

Compression and sparsification of neural networks has recently gained much traction in the deep learning community. The most common and straightforward technique is parameter / neuron pruning [101] according to some criterion. Whereas weight pruning [65, 130, 185] is in general inefficient for saving computation time, neuron pruning [112, 135, 192] can lead to computation savings. Unfortunately, all of the aforementioned methods require training the original dense network thus precluding the benefits we can obtain by having exact sparsity on the computation during training. This is in contrast to our approach where sparsification happens during training, thus theoretically allowing conditional computation to speed-up training [18, 19].

Emulating binary r.v.s with rectifications of continuous r.v.s is not a new concept and has been previously done with Gaussian distributions in the context of generative modelling [67, 75, 155] and with logistic distributions at [19] in the context of conditional computation. These distributions can similarly represent the value of exact zero, while still maintaining the tractability of continuous optimization. Nevertheless, they are sub-optimal when we require approximations to binary r.v.s (as is the case for the L_0 penalty); we cannot represent the bimodal behavior of a Bernoulli r.v. due to the fact that the underlying distribution is unimodal. Another technique that allows for gradient based optimization of

discrete r.v.s are the smoothing transformations proposed by [154]. There the core idea is that if a model has binary latent variables, then we can smooth them with continuous noise in a way that allows for reparametrization gradients. There are two main differences with the hard concrete distribution we employ here; firstly, the double rectification of the hard concrete r.v.s allows us to represent the values of exact zero and one (instead of just zero) and, secondly, due to the underlying concrete distribution the random samples from the hard concrete will better emulate binary r.v.s.

7.4 EXPERIMENTS

We validate the effectiveness of our method on two tasks. The first corresponds to the toy classification task of MNIST using a simple multilayer perceptron (MLP) with two hidden layers of size 300 and 100 [99], and a simple convolutional network, the LeNet-5-Caffe[5]. The second corresponds to the more modern task of CIFAR 10 and CIFAR 100 classification using Wide Residual Networks [202]. For all of our experiments we set $\gamma = -0.1$, $\zeta = 1.1$ and, following the recommendations from [120], set $\beta = 2/3$ for the concrete distributions. We initialized the locations $\log \alpha$ by sampling from a normal distribution with a standard deviation of 0.01 and a mean that yields $\frac{\alpha}{\alpha+1}$ to be approximately equal to the original dropout rate employed at each of the networks. We used a single sample of the gate $\mathbf{z}$ for each minibatch of datapoints during the optimization, even though this can lead to larger variance in the gradients [89]. In this way we show that we can obtain the speedups in training with practical implementations, without actually hurting the overall performance of the network. We have made the code publicly available at https://github.com/AMLab-Amsterdam/L0_regularization.

7.4.1 *MNIST classification and sparsification*

For these experiments we did no further regularization besides the L_0 norm and optimization was done with Adam [88] using the default hyper-parameters and temporal averaging. We can see at Table 10 that our approach is competitive with other methods that tackle neural network compression. However, it is worth noting that all of these approaches prune the network post-training using thresholds while requiring training the full network. We can further see that our approach minimizes the amount of parameters more at layers where the gates affect a larger part of the cost; for the MLP this corresponds to the input layer whereas for the LeNet5 this corresponds to the first fully connected layer. In contrast, the methods with sparsity inducing priors [112, 135] sparsify parameters irrespective of that extra cost (since they are only encouraged by the prior to move parameters to zero) and as a result they achieve similar sparsity on all of the layers. Nonetheless, it should be mentioned that we can in principle increase the sparsification on specific layers simply by specifying a separate λ for each layer, e.g. by increasing the λ for gates that affect less parameters. We provide such results at the "λ sep." rows.

5 https://github.com/BVLC/caffe/tree/master/examples/mnist

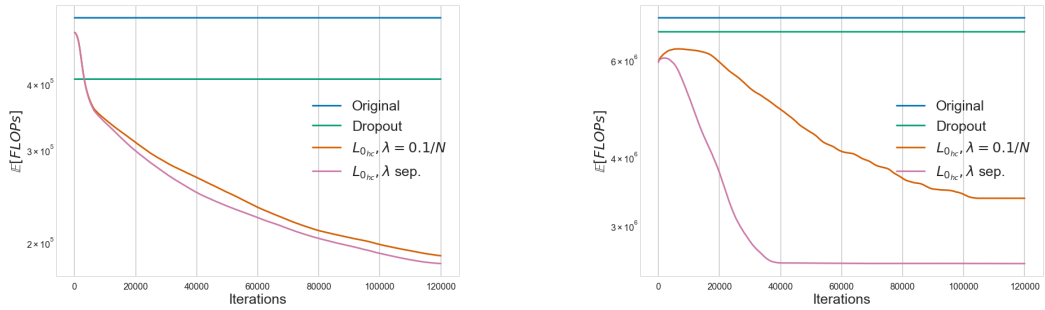
Table 10: Comparison of the learned architectures and performance of the baselines from [112] and the proposed L_0 minimization under $L_{0_{hc}}$. We show the amount of neurons left after pruning with the estimator in Eq. 107 along with the error in the test set after 200 epochs. N denotes the number of training datapoints.

Network & size	Method	Pruned architecture	Error (%)
MLP	Sparse VD [130]	512-114-72	1.8
784-300-100	BC-GNJ [112]	278-98-13	1.8
	BC-GHS [112]	311-86-14	1.8
	$L_{0_{hc}}, \lambda = 0.1/N$	219-214-100	1.4
	$L_{0_{hc}}, \lambda$ sep.	266-88-33	1.8
LeNet-5-Caffe	Sparse VD [130]	14-19-242-131	1.0
20-50-800-500	GL [192]	3-12-192-500	1.0
	GD [171]	7-13-208-16	1.1
	SBP [135]	3-18-284-283	0.9
	BC-GNJ [112]	8-13-88-13	1.0
	BC-GHS [112]	5-10-76-16	1.0
	$L_{0_{hc}}, \lambda = 0.1/N$	20-25-45-462	0.9
	$L_{0_{hc}}, \lambda$ sep.	9-18-65-25	1.0

To get a better idea about the potential speedup we can obtain in training we plot in Figure 20 the expected, under the probability of the gate being active, floating point operations (FLOPs) as a function of the training iterations. We also included the theoretical speedup we can obtain by using dropout [174] networks. As we can observe, our L_0 minimization procedure that is targeted towards neuron sparsity can potentially yield significant computational benefits compared to the original or dropout architectures, with minimal or no loss in performance. We further observe that there is a significant difference in the flop count for the LeNet model between the $\lambda = 0.1/N$ and λ sep. settings. This is because we employed larger values for λ ($10/N$ and $0.5/N$) for the convolutional layers (which contribute the most to the computation) in the λ sep. setting. As a result, this setting is more preferable when we are concerned with speedup, rather than network compression (which is affected only by the number of parameters).

7.4.2 *CIFAR classification*

For WideResNets we apply L_0 regularization on the weights of the hidden layer of the residual blocks, i.e. where dropout is usually employed. We also employed an L_2 regularization term as described in Section 7.2.3 with the weight decay coefficient used in [202]. For the layers with the hard concrete gates we divided the weight decay coefficient by 0.7



(a) Expected FLOPs at the MLP.

(b) Expected FLOPs at LeNet5.

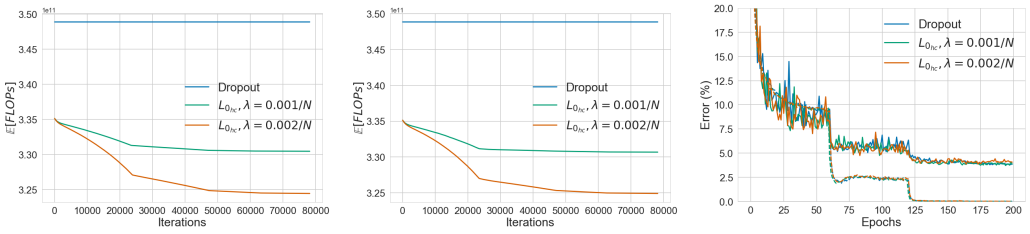
Figure 20: Expected number of floating point operations (FLOPs) during training for the original, dropout and L_0 regularized networks. These were computed by assuming one flop for multiplication and one flop for addition.

to ensure that a-priori we assume the same length-scale as the 0.3 dropout equivalent network. For optimization we employed the procedure described in [202] with a minibatch of 128 datapoints, which was split between two GPUs, and used a single sample for the gates for each GPU.

Table 11: Results on the benchmark classification tasks of CIFAR 10 and CIFAR 100. All of the baseline results are taken from [202]. For the L_0 regularized WRN we report the median of the error on the test set after 200 epochs over 5 runs.

Network	CIFAR-10	CIFAR-100
original-ResNet-110 [69]	6.43	25.16
pre-act-ResNet-110 [70]	6.37	-
WRN-28-10 [202]	4.00	21.18
WRN-28-10-dropout [202]	3.89	18.85
WRN-28-10-$L_{0_{hc}}, \lambda = 0.001/N$	**3.83**	**18.75**
WRN-28-10-$L_{0_{hc}}, \lambda = 0.002/N$	3.93	19.04

As we can observe at Table 11, with a λ of $0.001/N$ the L_0 regularized wide residual network improves upon the accuracy of the dropout equivalent network on both CIFAR 10 and CIFAR 100. Furthermore, it simultaneously allows for potential training time speedup due to gradually decreasing the number of FLOPs, as we can see in Figures 21a, 21b. This sparsity is also obtained without any "lag" in convergence speed, as at Figure 21c we observe a behaviour that is similar to the dropout network. Finally, we observe that by further increasing λ we obtain a model that has a slight error increase but can allow for a larger speedup.



(a) (b) (c)

Figure 21: (a, b) Expected number of FLOPs during training for the dropout and L_0 regularized WRNs for CIFAR 10 (a) and CIFAR 100 (b). The original WRN is not shown as it has the same practical FLOPs as the dropout equivalent network. (c) Train (dashed) and test (solid) error as a function of the training epochs for dropout and L_0 WRNs at CIFAR 10.

7.5 DISCUSSION

We have described a general recipe that allows for optimizing the L_0 norm of parametric models in a principled and effective manner. The method is based on smoothing the combinatorial problem with continuous distributions followed by a hard-sigmoid. To this end, we also proposed a novel distribution which we coin as the hard concrete; it is a "stretched" binary concrete distribution, the samples of which are transformed by a hard-sigmoid. This in turn better mimics the binary nature of Bernoulli distributions while still allowing for efficient gradient based optimization. In experiments we have shown that the proposed L_0 minimization process leads to neural network sparsification that is competitive with current approaches while theoretically allowing for speedup in training. We have further shown that this process can provide a good inductive bias and regularizer, as on the CIFAR experiments with wide residual networks we improved upon dropout.

As for future work; better harnessing the power of conditional computation for efficiently training very large neural networks with learned sparsity patterns is a potential research direction. It would be also interesting to adopt a full Bayesian treatment over the parameters θ, such as the one employed at [112, 130]. This would then allow for further speedup and compression due to the ability of automatically learning the bit precision of each weight. Finally, it would be interesting to explore the behavior of hard concrete r.v.s at binary latent variable models, since they can be used as a drop in replacement that allow us to maintain both the discrete nature as well as the efficient reparametrization gradient optimization.

8

RELAXED QUANTIZATION FOR DISCRETIZED NEURAL NETWORKS

In the previous chapter we showed how we can prune neural networks through the golden standard for sparsity, the L_0 norm. In this chapter, we address the second part of the neural network compression recipe, quantization of the weights and activations of the neural network. In order to train networks that can be effectively discretized without loss of performance, we introduce a differentiable quantization procedure. Differentiability can be achieved by transforming continuous distributions over the weights and activations of the network to categorical distributions over the quantization grid. These are subsequently relaxed to continuous surrogates that can allow for efficient gradient-based optimization. We further show that stochastic rounding can be seen as a special case of the proposed approach and that under this formulation the quantization grid itself can also be optimized with gradient descent. We experimentally validate the performance of our method on MNIST, CIFAR 10 and Imagenet classification.[1]

8.1 INTRODUCTION

Neural networks excel in a variety of large scale problems due to their highly flexible parametric nature. However, deploying big models on resource constrained devices, such as mobile phones, drones or IoT devices is still challenging because they require a large amount of power, memory and computation. Neural network compression is a means to tackle this issue and has therefore become an important research topic.

Neural network compression can be, roughly, divided into two not mutually exclusive categories: pruning and quantization. While pruning [65, 101] aims to make the model "smaller" by altering the architecture, quantization aims to reduce the precision of the arithmetic operations in the network. In this paper we focus on the latter. Most network quantization methods either simulate or enforce discretization of the network during training, e.g. via rounding of the weights and activations. Although seemingly straighforward, the discontinuity of the discretization makes the gradient-based optimization infeasible. The reason is that there is no gradient of the loss with respect to the parameters. A workaround to the discontinuity are the "pseudo-gradients" according to the straight-through estimator [19], which have been successfully used for training low-bit width architectures at e.g. [80, 208].

1 This chapter is adapted from our publication [108].

The purpose of this work is to introduce a novel quantization procedure, Relaxed Quantization (RQ). RQ can bypass the non-differentiability of the quantization operation during training by smoothing it appropriately. The contributions of this paper are four-fold: **First**, we show how to make the set of quantization targets part of the training process such that we can optimize them with gradient descent. **Second**, we introduce a way to discretize the network by converting distributions over the weights and activations to categorical distributions over the quantization grid. **Third**, we show that we can obtain a "smooth" quantization procedure by replacing the categorical distributions with concrete [85, 120] equivalents. **Finally** we show that stochastic rounding [61], one of the most popular quantization techniques, can be seen as a special case of the proposed framework. We present the details of our approach in Section 8.2, discuss related work in Section 8.3 and experimentally validate it in Section 8.4. Finally we conclude and provide fruitful directions for future research in Section 8.5.

8.2 RELAXED QUANTIZATION FOR DISCRETIZING NEURAL NETWORKS

The central element for the discretization of weights and activations of a neural network is a quantizer $q(\cdot)$. The quantizer receives a (usually) continous signal as input and discretizes it to a countable set of values. This process is inherently lossy and non-invertible: given the output of the quantizer, it is impossible to determine the exact value of the input. One of the simplest quantizers is the rounding function:

$$q(x) = \alpha \left\lfloor \frac{x}{\alpha} + \frac{1}{2} \right\rfloor,$$

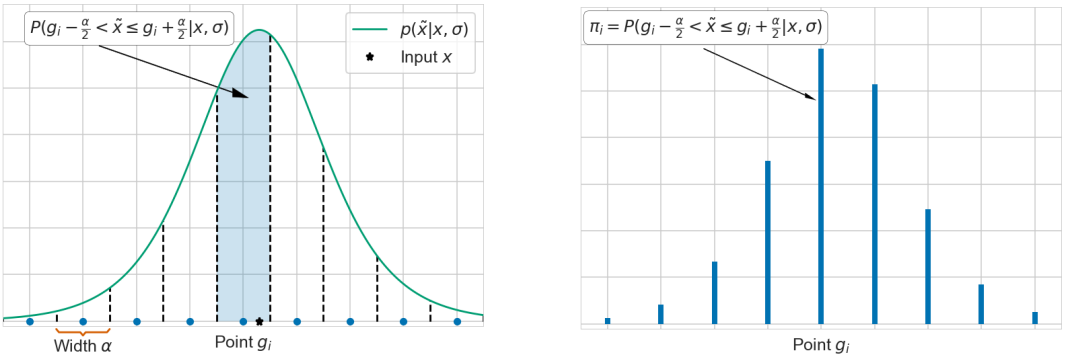
where α corresponds to the step size of the quantizer. With $\alpha = 1$, the quantizer rounds x to its nearest integer number.

Unfortunately, we cannot simply apply the rounding quantizer to discretize the weights and activations of a neural network. Because of the quantizers' lossy and non-invertible nature, important information might be destroyed and lead to a decrease in accuracy. To this end, it is preferable to train the neural network while simulating the effects of quantization during the training procedure. This encourages the weights and activations to be robust to quantization and therefore decreases the performance gap between a full-precision neural network and its discretized version.

However, the aforementioned rounding process is non-differentiable. As a result, we cannot directly optimize the discretized network with stochastic gradient descent, the workhorse of neural network optimization. In this work, we posit a "smooth" quantizer as a possible way for enabling gradient based optimization.

8.2.1 *Learning (fixed point) quantizers via gradient descent*

The proposed quantizer comprises four elements: a vocabulary, its noise model and the resulting discretization procedure, as well as a final relaxation step to enable gradient based optimization.



(a) (b)

Figure 22: The proposed discretization process. **(a)** Given a distribution $p(\tilde{x})$ over the real line we partition it into K intervals of width α where the center of each of the intervals is a grid point g_i. The shaded area corresponds to the probability of $\tilde{x}$ falling inside the interval containing that specific g_i. **(b)** Categorical distribution over the grid obtained after discretization. The probability of each of the grid points g_i is equal to the probability of $\tilde{x}$ falling inside their respective intervals.

The first element of the quantizer is the vocabulary: it is the set of (countable) output values that the quantizer can produce. In our case, this vocabulary has an inherent structure, as it is a grid of ordered scalars. For fixed point quantization the grid $\mathcal{G}$ is defined as

$$\mathcal{G} = \left[-2^{b-1}, \ldots, 0, \ldots, 2^{b-1} - 1\right], \tag{112}$$

where b is the number of available bits that allow for $K = 2^b$ possible integer values. By construction this grid of values is agnostic to the input signal x and hence suboptimal; to allow for the grid to adapt to x we introduce two free parameters, a scale α and an offset β. This leads to a learnable grid via $\hat{\mathcal{G}} = \alpha\mathcal{G} + \beta$ that can adapt to the range and location of the input signal.

The second element of the quantizer is the assumption about the input noise ϵ; it determines how probable it is for a specific value of the input signal to move to each grid point. Adding noise to x will result in a quantizer that is, on average, a smooth function of its input. In essense, this is an application of variational optimization [175] to the non-differentiable rounding function, which enables us to do gradient based optimization.

We model this form of noise as acting additively to the input signal x and being governed by a distribution $p(\epsilon)$. This process induces a distribution $p(\tilde{x})$ where $\tilde{x} = x + \epsilon$. In the next step of the quantization procedure, we discretize $p(\tilde{x})$ according to the quantization grid $\hat{\mathcal{G}}$; this neccesitates the evaluation of the cumulative distribution function (CDF). For this reason, we will assume that the noise is distributed according to a zero mean logistic distribution with a standard deviation σ, i.e. $L(0, \sigma)$, hence leading to $p(\tilde{x}) = L(x, \sigma)$. The CDF of the logistic distribution is the sigmoid function which is easy to evaluate and backpropagate through. Using Gaussian distributions proved to be less effective in preliminary experiments. Other distributions are conceivable and we will briefly discuss the choice of a uniform distribution in Section 8.2.3.

The third element is, given the aforementioned assumptions, how the quantizer determines an appropriate assignment for each realization of the input signal x. Due to the

stochastic nature of $\tilde{x}$, a deterministic round-to-nearest operation will result in a stochastic quantizer for x. Quantizing x in this manner corresponds to discretizing $p(\tilde{x})$ onto $\hat{\mathcal{G}}$ and then sampling grid points g_i from it. More specifically, we construct a categorical distribution over the grid by adopting intervals of width equal to α centered at each of the grid points. The probability of selecting that particular grid point will now be equal to the probability of $\tilde{x}$ falling inside those intervals:

$$p(\hat{x} = g_i|x, \sigma) = P(\tilde{x} \leqslant (g_i + \alpha/2)) - P(\tilde{x} < (g_i - \alpha/2))) \tag{113}$$

$$= \text{Sigmoid}((g_i + \alpha/2 - x)/\sigma) - \text{Sigmoid}((g_i - \alpha/2 - x)/\sigma), \tag{114}$$

where $\hat{x}$ corresponds to the quantized variable, $P(\cdot)$ corresponds to the CDF and the step from equation 113 to equation 114 is due to the logistic noise assumption. A visualization of the aforementioned process can be seen in Figure 22. For the first and last grid point we will assume that they reside within $(g_0 - \alpha/2, g_0 + \alpha/2]$ and $(g_K - \alpha/2, g_K + \alpha/2]$ respectively. Under this assumption we will have to truncate $p(\tilde{x})$ such that it only has support within $(g_0 - \alpha/2, g_K + \alpha/2]$. Fortunately this is easy to do, as it corresponds to just a simple modification of the CDF:

$$P(\tilde{x} \leqslant c|\tilde{x} \in (g_0 - \alpha/2, g_K + \alpha/2]) = \frac{P(\tilde{x} \leqslant c) - P(\tilde{x} < (g_0 - \alpha/2))}{P(\tilde{x} \leqslant (g_K + \alpha/2)) - P(\tilde{x} < (g_0 - \alpha/2))}. \tag{115}$$

Armed with this categorical distribution over the grid, the quantizer proceeds to assign a specific grid value to $\hat{x}$ by drawing a random sample. This procedure emulates quantization noise, which prevents the model from fitting the data. This noise can be reduced in two ways: by clustering the weights and activations around the points of the grid and by reducing the logistic noise σ. As $\sigma \to 0$, the CDF converges towards the step function, prohibiting gradient flow. On the other hand, if ϵ is too high, the optimization procedure is very noisy, prohibiting convergence. For this reason, during optimization we initialize σ in a sensible range, such that $L(x, \sigma)$ covers a significant portion of the grid. Please confer Appendix 10.6.1 for details. We then let σ be freely optimized via gradient descent such that the loss is minimized. Both effects reduce the gap between the function that the neural network computes during training time vs. test time. We illustrate this in Figure 23.

The fourth element of the procedure is the relaxation of the non differentiable categorical distribution sampling. This is achieved by replacing the categorical distribution with a concrete distribution [85, 120]. This relaxation procedure corresponds to adopting a "smooth" categorical distribution that can be seen as a "noisy" softmax. Let π_i be the categorical probability of sampling grid point i, i.e. $\pi_i = p(\hat{x} = g_i)$; the "smoothed" quantized value $\hat{x}$ can be obtained via:

$$u_i \sim \text{Gumbel}(0, 1), \quad z_i = \frac{\exp((\log \pi_i + u_i)/\lambda)}{\sum_j \exp((\log \pi_j + u_j)/\lambda)}, \quad \hat{x} = \sum_{i=1}^{K} z_i g_i, \tag{116}$$

where z_i is the random sample from the concrete distribution and λ is a temperature parameter that controls the degree of approximation, since as $\lambda \to 0$ the concrete distribution becomes a categorical.

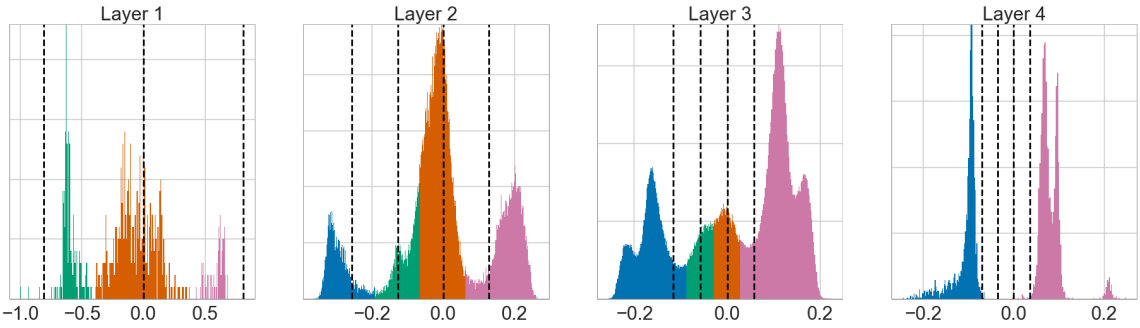


Figure 23: Best viewed in color. Illustration of the inductive bias obtained via training with the proposed quantizer; means of the logistic distribution over the weights for each layer of the LeNet-5 when trained with 2 bits per weight and activation. Each color corresponds to an assignment to a particular grid point and the vertical dashed lines correspond to the grid points ($\beta = 0$). We can clearly see that the real valued weights are naturally encouraged through training to cluster into multiple modes, one for each grid point. It should also be mentioned, that for the right and leftmost grid points the probability of selecting them is maximized by moving the corresponding weight furthest right or left respectively. Interestingly, we observe that the network converged to ternary weights for the input and (almost) binary weights for the output layer.

We have thus defined a fully differentiable "soft" quantization procedure that allows for stochastic gradients for both the quantizer parameters α, β, σ as well as the input signal x (e.g. the weights or the activations of a neural network). We refer to this alrogithm as Relaxed Quantization (RQ). We summarize its forward pass as performed during training in Algorithm 3. It is also worthwhile to notice that if there were no noise at the input x then the categorical distribution would have non-zero mass only at a single value, thus prohibiting gradient based optimization for x and σ.

One drawback of this approach is that the smoothed quantized values defined in equation 116 do not have to coincide with grid points, as $\mathbf{z}$ is not a one-hot vector. Instead, these values can lie anywhere between the smallest and largest grid point, something which is impossible with e.g. stochastic rounding [61]. In order to make sure that only grid-points are sampled, we propose an alternative algorithm RQ ST in which we use the variant of the straight-through (ST) estimator proposed in [85]. Here we sample the actual categorical distribution during the forward pass but assume a sample from the concrete distribution for the backward pass. While this gradient estimator is obviously biased, in practice it works as the "gradients" seem to point towards a valid direction. We perform experiments with both variants.

After convergence, we can obtain a "hard" quantization procedure, i.e. select points from the grid, at test time by either reverting to a categorical distribution (instead of the continuous surrogate) or by rounding to the nearest grid point. In this paper we chose the latter as it is more aligned with the low-resource environments in which quantized models will be deployed. Furthermore, with this goal in mind, we employ two quantization grids with their own learnable scalar α, σ (and potentially β) parameters for each layer; one for the weights and one for the activations.

Algorithm 3 Quantization during training.

Require: Input x, grid $\hat{g}$, scale of the grid α, scale of noise σ, temperature λ, fuzz param. ϵ

$\mathbf{r} = [\hat{g} - \alpha/2, g_K + \alpha/2]$ # interval points

$\mathbf{c} = \text{Sigmoid}((\mathbf{r} - x)/\sigma)$ # evaluate CDF

$\pi_i = \frac{c[i+1]-c[i]+\epsilon}{c[K+1]-c[1]+K\epsilon}$ # categorical distr.

$\mathbf{z} \sim \text{Concrete}(\boldsymbol{\pi}, \lambda)$

return $\sum_i z_i g_i$

8.2.2 *Scalable quantization via a local grid*

Sampling $\hat{x}$ based on drawing K random numbers for the concrete distribution as described in equation 116 can be very expensive for larger values of K. Firstly, drawing K random numbers for every individual weight and activation in a neural network drastically increases the number of operations required in the forward pass. Secondly, it also requires keeping many more numbers in memory for gradient computations during the backward pass. Compared to a standard neural network or stochastic rounding approaches, the proposed procedure can thus be infeasible for larger models and datasets.

Fortunately, we can make sampling $\hat{x}$ independent of the grid size by assuming zero probability for grid-points that lie far away from the signal x. Specifically, by only considering grid points that are within δ standard deviations away from x, we truncate $p(\tilde{x})$ such that it lies within a "localized" grid around x.

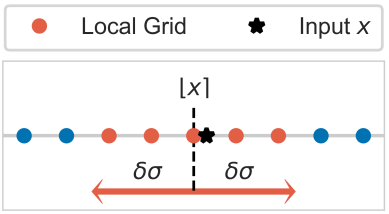


Figure 24: Local grid construction

To simplify the computation required for determining the local grid elements, we choose the grid point closest to x, $\lfloor x \rceil$, as the center of the local grid (Figure 24). Since σ is shared between all elements of the weight matrix or activation, the local grid has the same width for every element. The computation of the probabilities over the localized grid is similar to the truncation happening in equation 115 and the smoothed quantized value is obtained via a manner similar to equation 116:

$$P(\tilde{x} \leqslant c | \tilde{x} \in (\lfloor x \rceil - \delta\sigma, \lfloor x \rceil + \delta\sigma]) = \frac{P(\tilde{x} \leqslant c) - P(\tilde{x} < \lfloor x \rceil - \delta\sigma)}{P(\tilde{x} \leqslant \lfloor x \rceil + \delta\sigma) - P(\tilde{x} < \lfloor x \rceil - \delta\sigma)} \tag{117}$$

$$\hat{x} = \sum_{g_i \in (\lfloor x \rceil - \delta\sigma, \lfloor x \rceil + \delta\sigma]} z_i g_i \tag{118}$$

Algorithm 4 Quantization during testing.

Require: Input x, scale and offset of the grid α, β, minimum and maximum values g_0, g_K

$y = \alpha \cdot \text{round}((x - \beta)/\alpha) + \beta$
return $\min(g_K, \max(g_0, y))$

8.2.3 *Relation to Stochastic Rounding*

One of the pioneering works in neural network quantization has been the work of [61]; it introduced stochastic rounding, a technique that is one of the most popular approaches for training neural networks with reduced numerical precision. Instead of rounding to the nearest representable value, the stochastic rounding procedure selects one of the two closest grid points with probability depending on the distance of the high precision input from these grid points. In fact, we can view stochastic rounding as a special case of RQ where $p(\tilde{x}) = U(x - \frac{\alpha}{2}, x + \frac{\alpha}{2})$. This uniform distribution centered at x of width equal to the grid width α generally has support only for the closest grid point. Discretizing this distribution to a categorical over the quantization grid however assigns probabilities to the two closest grid points as in stochastic rounding, following equation 113:

$$\begin{aligned} p(\hat{x} = \left\lfloor \frac{x}{\alpha} \right\rfloor \alpha | x) &= P(\tilde{x} \leqslant (\left\lfloor \frac{x}{\alpha} \right\rfloor \alpha + \alpha/2)) - P(\tilde{x} < (\left\lfloor \frac{x}{\alpha} \right\rfloor \alpha - \alpha/2)) \\ &= \left\lceil \frac{x}{\alpha} \right\rceil - \frac{x}{\alpha}. \end{aligned} \tag{119}$$

Stochastic rounding has proven to be a very powerful quantization scheme, even though it relies on biased gradient estimates for the rounding procedure. On the one hand, RQ provides a way to circumvent this estimator at the cost of optimizing a surrogate objective. On the other hand, RQ ST makes use of the unreasonably effective straight-through estimator as used in [85] to avoid optimizing a surrogate objective, at the cost of biased gradients. Compared to stochastic rounding, RQ ST further allows sampling of not only the two closest grid points, but also has support for more distant ones depending on the estimated input noise σ. Intuitively, this allows for larger steps in the input space without first having to decrease variance at the traversion between grid sections.

8.3 RELATED WORK

In this work we focus on hardware oriented quantization approaches. As opposed to methods that focus only on weight quantization and network compression for a reduced memory footprint, quantizing all operations within the network aims to additionally provide reduced execution speeds. Within the body of work that considers quantizing weights and activations fall papers using stochastic rounding [61, 63, 80, 198]. [198] also consider quantized backpropagation, which is out-of-scope for this work.

Furthermore, another line of work considers binarizing [36, 206] or ternarizing [103, 206] weights and activations [80, 149, 207] via the straight-through gradient estimator [19]; these allow for fast implementations of convolutions using only bit-shift operations. In a similar vein, the straight through estimator has also been used in [27, 47, 83, 125, 205] for quantizing neural networks to arbitrary bit-precision. In these approaches, the full precision weights that are updated during training correspond to the means of the logistic distributions that are used in RQ. Furthermore, [83] maintains moving averages for the minimum and maximum observed values for activations while parameterises the network's weights' grids via their minimum and maximum values directly. This fixed-point grid is therefore learned during training, however without gradient descent; unlike the proposed RQ. Alternatively, instead of discretizing real valued weights, [161] directly optimize discrete distributions over them. While providing promising results, this approach does not generalize straightforwardly to activation quantization.

Another line of work quantizes networks through regularization. [112] formulate a variational approach that allows for heuristically determining the required bit-width precision for each weight of the model. Improving upon this work, [2] proposed a quantizing prior that encourages ternary weights during training. Similarly to RQ, this method also allows for optimizing the scale of the ternary grid. In contrast to RQ, this is only done implicitly via the regularization term. One drawback of these approaches is that the strength of the regularization decays with the amount of training data, thus potentially reducing their effectiveness on large datasets.

Weights in a neural network are usually not distributed uniformly within a layer. As a result, performing non-uniform quantization is usually more effective. [15] employ a stochastic quantizer by first uniformizing the weight or activation distribution through a non-linear transformation and then injecting uniform noise into this transformed space. [146] propose a version of their method in which the quantizer's code book is learned by gradient descent, resulting in a non-uniformly spaced grid. Another line of works quantizes by clustering and therefore falls into this category; [65, 185] represent each of the weights by the centroid of its closest cluster. While such non-uniform techniques can be indeed effective, they do not allow for efficient implementations on todays hardware.

Within the liteterature on quantizing neural networks there are many approaches that are orthogonal to our work and could potentially be combined for additional improvements. [125, 146] use knowledge distrillation techniques to good effect, whereas works such as [126] modify the architecture to compensate for lower precision computations. [15, 205, 206] perform quantization in an step-by-step manner going from input layer to output, thus allowing the later layers to more easily adapt to the rounding errors introduced. [47, 146] further employ "bucketing", where small groups of weights share a grid, instead of one grid per layer. As an example from [146], a bucket size of 256 weights per grid on Resnet-18 translates to $\sim$ 45.7k separate weight quantization grids as opposed to 22 in RQ.

8.4 EXPERIMENTS

For the subsequent experiments RQ will correspond to the proposed procedure that has concrete sampling and RQ ST will correspond to the proposed procedure that uses the Gumbel-softmax straight-through estimator [85] for the gradient. We did not optimize an offset for the grids in order to be able to represent the number zero exactly, which allows for sparcity and is required for zero-padding. Furthermore we assumed a grid that starts from zero when quantizing the outputs of ReLU. We provide further details on the experimental settings at Appendix 10.6.1. We will also provide results of our own implementation of stochastic rounding [61] with the dynamic fixed point format [63] (SR+DR). Here we used the same hyperparameters as for RQ. All experiments were implemented with TensorFlow [1], using the Keras library [34].

8.4.1 *LeNet-5 on MNIST and VGG7 on CIFAR 10*

For the first task we considered the toy LeNet-5 network trained on MNIST with the 32C5 - MP2 - 64C5 - MP2 - 512FC - Softmax architecture and the VGG 2x(128C3) - MP2 - 2x(256C3) - MP2 - 2x(512C3) - MP2 - 1024FC - Softmax architecture on the CIFAR 10 dataset. Details about the hyperparameter settings can be found in Appendix 10.6.1.

By observing the results in Table 12, we see that our method can achieve competitive results that improve upon several recent works on neural network quantization. Considering that we achieve lower test error for 8 bit quantization than the high-precision models, we can see how RQ has a regularizing effect. Generally speaking we found that the gradient variance for low bit-widths (i.e. 2-4 bits) in RQ needs to be kept in check through appropriate learning rates.

8.4.2 *Resnet-18 on Imagenet*

In order to demonstrate the effectiveness of our proposed approach on large scale tasks we considered the task of quantizing a Resnet-18 [69] trained on the Imagenet (ILSVRC2012) dataset. We started from a pre-trained full precision model that was trained for 90 epochs. We quantized the weights of all layers, post ReLU activations and average pooling layer for various bit-widths via fine-tuning for ten epochs. Further details can be found in Appendix 10.6.2.

Some of the existing quantization works do not quantize the first (and sometimes) last layer. Doing so simplifies the problem but it can, depending on the model and input dimensions, significantly increase the amount of computation required. We therefore make use of the bit operations per second (BOPs) metric [15], which can be seen as a proxy for the execution speed on appropriate hardware. In BOPs, the impact of not quantizing the first layer in, for example, the Resnet-18 model on Imagenet, becomes apparent: keeping the first layer in full precision requires roughly 2.5 times as many BOPs for one forward pass through the whole network compared to quantizing all weights and activations to 5 bits.

Table 12: Test error (%) on MNIST and CIFAR 10 using LeNet5-Caffe and VGG-7 respectively. Two and four bit for VGG with SR+DR resulted in a big gap between training and validation accuracy, so we omit those results.

Method	# Bits weights/act.	MNIST	CIFAR 10
Original	32/32	0.64	6.95
SR+DR	8/8	0.58	7.06
[61, 63]	4/4	0.66	-
	2/2	1.03	-
Deep Comp. [65]	(5-8)/32	0.74	-
TWN [103]	2/32	0.65[a]	7.44
BWN [149]	1/32	-	9.88
XNOR-net [149]	1/1	-	10.17
SWS [185]	3/32	0.97	-
Bayesian Comp. [112]	(7-18)/32	1.00	-
VNQ [2]	2/32	0.73	-
WAGE [198]	2/8	0.40	6.78
LR Net [161][b]	1/32	0.53[a]	6.82
	2/32	0.50[a]	6.74
RQ (ours)	8/8	0.55	6.70
	4/4	0.58	8.43
	2/2	0.76	11.75
RQ ST (ours)	8/8	0.56	6.72
	4/4	0.61	7.96
	2/2	0.63	9.08

a With batch normalization after convolution
b Last layer in full precision

Figure 25 compares a wide range of methods in terms of accuracy and BOPs. We choose to compare only against methods that employ fixed-point quantization on Resnet-18 hence do not compare with non-uniform quantization techniques, such as the one described at [15]. In addition to our own implementation of [61] with the dynamic fixed point format [63], we also report results of "rounding". This corresponds to simply rounding the pre-trained high-precision model followed by re-estimation of the batchnorm statistics. The grid in this case is defined as the initial grid used for fine-tuning with RQ. For batchnorm re-estimation and grid initialization, please confer Appendix 10.6.1.

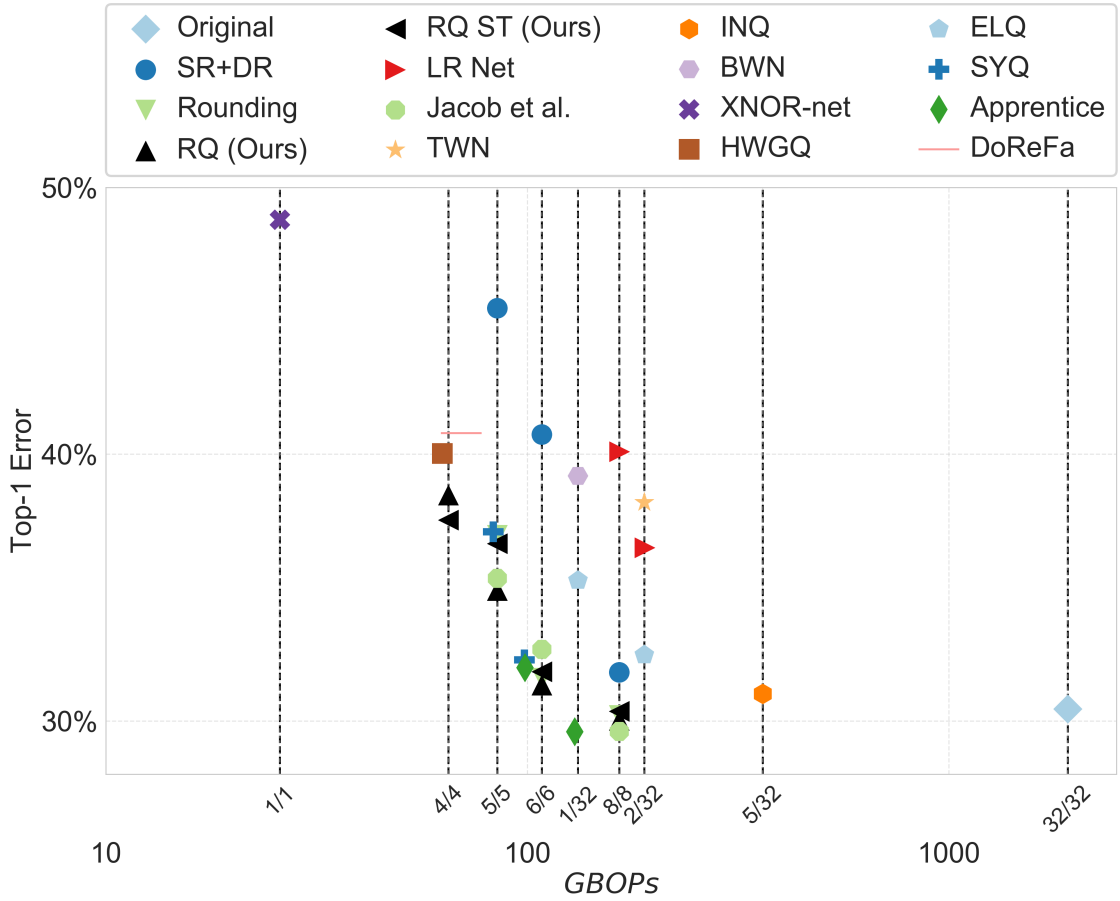


Figure 25: Best viewed in color. Comparison of various methods on Resnet-18 according to top-1 error (on the y-axis) and bit operations per second (on the x-axis) computed according to the formula described in [15]. Each dashed line corresponds to employing a specific bit configuration for every layer's weights and activations. Values for top-1 and top-5 errors are given in Table 16 in the Appendix. We compare against multiple works that employ fixed-point quantization: SR+DR [61, 63], LR Net [161], [83], TWN [103], INQ [205], BWN [149], XNOR-net [149], HWGQ [27], ELQ [206], SYQ [47], Apprentice [125] and rounding.

We observe that on ResNet-18 the RQ variants form the "Pareto frontier" in the trade-off between accuracy and efficiency, along with SYQ, Apprentice and [83]. SYQ, however, employs "bucketing" and Apprentice uses distillation, both of which can be combined with RQ and improve performance. [83] does better than RQ with 8 bits, however RQ improved w.r.t. to its pretrained model, whereas [83] decreased slightly. For experimental details with [83], please confer Appendix 10.6.2.1. SR+DR underperforms in this setting and is worse than simple rounding for 5 to 8 bits.

8.5 DISCUSSION

We have introduced Relaxed Quantization (RQ), a powerful and versatile algorithm for learning low-bit neural networks using a uniform quantization scheme. As such, the models trained by this method can be easily transferred and executed on low-bit fixed point

chipsets. We have extensively evaluated RQ on various image classification benchmarks and have shown that it allows for the better trade-offs between accuracy and bit operations per second.

Future hardware might enable us to cheaply do non-uniform quantization, for which this method can be easily extended. [96, 139] for example, show the benefits of low-bit floating point weights that can be efficiently implemented in hardware. The floating point quantization grid can be easily learned with RQ by redefining $\hat{\mathcal{G}}$. General non-uniform quantization, as described for example in [15], is a natural extension to RQ, whose exploration we leave to future work. Currently, the bit-width of every quantizer is determined beforehand, but in future work we will explore learning the required bit precision within this framework. In our experiments, batch normalization was implemented as a sequence of convolution, batch normalization and quantization. On a low-precision chip, however, batch normalization would be ”folded" [83] into the kernel and bias of the convolution, the result of which is then rounded to low precision. In order to accurately reflect this folding at test time, future work on the proposed algorithm will emulate folded batchnorm at training time and learn the corresponding quantization grid of the modified kernel and bias. For fast model evaluation on low-precision hardware, quantization goes hand-in-hand with network pruning. The proposed method is orthogonal to pruning methods such as, for example, L_0 regularization [115], which allows for group sparsity and pruning of hidden units.

9

CONCLUSION

In this thesis we aimed to improve deep learning, in hopes of making it a robust and effective tool that we can use to improve our lives. We identified two key research questions that involved 1) enhancing neural networks with the ability to robustly represent their uncertainty and 2) making neural networks smaller and faster while retaining their good predictive capabilities.

For the first question we believe that Bayesian reasoning provide us with an elegant and principled way to obtain effective uncertainty estimates for neural networks. In Chapters 3, 4 we explored the straightforward application of such reasoning to neural networks and we verified that this is indeed the case. As a follow up work, we presented in Chapter 5 the functional neural process (FNP), a model that combines the power of deep learning and the Bayesian machinery of stochastic processes. We showed that such models allow for more intuitive priors (compared to priors over the weights of the neural network), a fact that allowed us to obtain more robust uncertainty estimates than Bayesian neural networks while retaining the same predictive accuracy. As a result, these three chapters provided a positive answer to the first research question.

Nevertheless, while we do believe that the research of this work is a step towards a good direction, there is still more work to be done in order to obtain neural networks that can robustly represent their uncertainty in real world tasks. First of all, there are significant optimization challenges for variational BNNs that are yet to be resolved; the Monte Carlo sampling that is necessary for sampling the variational distribution over the weights, e.g. in Chapters 3, 4, 6, can lead to large variance in the gradients, a fact that prohibits convergence to high accuracy optima. For this reason, we had to resort to ad-hoc ways to mitigate this effect by e.g. clamping the variance of the approximate posterior in Chapters 4, 6. While lower variance gradient estimators such as [89, 193] exist, we believe that they are only a part of the final solution and optimizers that are better suited for such kind of tasks, along with better prior / variational posterior pairs can provide more principled solutions to this phenomenon. Secondly, the priors usually adopted are overly generic or chosen for computational convenience; for varational BNNs the standard Gaussian is prevalent in the literature [23, 50, 113, 114], whereas for models that posit distributions over functions such as the functional variational BNNs [176] or the FNPs [110] generic Gaussian process priors or priors based on locality in the latent space are employed. We believe that we, as a community, should collectively work towards allowing a broader range of prior knowledge, for example informative group structure priors that respect the symmetries in the data, thus expanding the toolbox of a practitioner. Finally, we believe

that proper benchmarks for this particular problem need to be established as they can greatly help in advancing the state of the art, akin to how Imagenet[1] guided the research in neural networks for computer vision. For this reason, works such as the one described at https://github.com/OATML/bdl-benchmarks or [141] are important for this field.

For the second question we showed in Chapter 6 that through Bayesian inference and sparsity inducing priors over the weights of the network we can also provide neural networks that are fast and compact via joint pruning and quantization. We further elaborated on pruning and quantization separately in Chapters 7, 8. In Chapter 7 we showed how the golden standard for sparsity and pruning but non-differentiable L_0 norm can be optimized with stochastic gradient descent via the inclusion of a collection of non-negative stochastic gates. In Chapter 8 we showed how we can effectively optimize quantized neural networks where both their weights and activations are constrained to obey a fixed point quantization grid with an-apriori defined bit-width budget. The answer to the second research question has thus been similarly positive.

Similarly to the first research question, there is still work to be done in order to push towards even smaller and accurate neural networks. We believe that promising research directions are towards formalizing objectives that can perform joint pruning and quantization, akin to how it was performed (albeit at an implicit way) in Chapter 6. This will allow the given optimizer to find better trade-offs between the computational complexity and accuracy of the model. Furthermore, we need to define proper metrics for the performance of the model on a given device, such as a mobile phone. The amount of FLOPs is only loosely correlated with the actual performance on-device, as we have to also take into account e.g. memory and data transfer. Finally, most modern architectures have been defined with the goal of best performance in mind, something which does not always correlate with efficiency. For this reason, we believe that neural architecture search methods, something which was not explored in this thesis, coupled with hardware simulators are well suited to tackle this task.

1 Imagenet

BIBLIOGRAPHY

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. "Variational Network Quantization." In: *International Conference on Learning Representations*. 2018.

[3] Felix V Agakov and David Barber. "An auxiliary variational method." In: *International Conference on Neural Information Processing*. Springer. 2004, pp. 561–566.

[4] Sungjin Ahn, Anoop Korattikara Balan, and Max Welling. "Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring." In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. 2012.

[5] Hirotogu Akaike. "Information theory and an extension of the maximum likelihood principle." In: *Selected Papers of Hirotugu Akaike*. Springer, 1998, pp. 199–213.

[6] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. "Deep variational information bottleneck." In: *arXiv preprint arXiv:1612.00410* (2016).

[7] David F Andrews and Colin L Mallows. "Scale mixtures of normal distributions." In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1974), pp. 99–102.

[8] Artin Armagan, Merlise Clyde, and David B Dunson. "Generalized beta mixtures of Gaussians." In: *Advances in neural information processing systems*. 2011, pp. 523–531.

[9] Arthur Asuncion and David Newman. *UCI machine learning repository*. 2007.

[10] Andrei Atanov, Arsenii Ashukha, Kirill Struminsky, Dmitry Vetrov, and Max Welling. "The Deep Weight Prior. Modeling a prior distribution for CNNs using generative models." In: *arXiv preprint arXiv:1810.06943* (2018).

[11] Erfan Azarkhish, Davide Rossi, Igor Loi, and Luca Benini. "Neurostream: Scalable and Energy Efficient Deep Learning with Smart Memory Cubes." In: *arXiv preprint arXiv:1701.06420* (2017).

[12] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in neural information processing systems* (2014), pp. 2654–2662.

[13] Juhan Bae, Guodong Zhang, and Roger Grosse. "Eigenvalue corrected noisy natural gradient." In: *arXiv preprint arXiv:1811.12565* (2018).

[14] Sergey Bartunov and Dmitry Vetrov. "Few-shot generative modelling with generative matching networks." In: *International Conference on Artificial Intelligence and Statistics*. 2018, pp. 670–678.

[15] Chaim Baskin, Eli Schwartz, Evgenii Zheltonozhskii, Natan Liss, Raja Giryes, Alex M Bronstein, and Avi Mendelson. "UNIQ: Uniform Noise Injection for the Quantization of Neural Networks." In: *arXiv preprint arXiv:1804.10969* (2018).

[16] Matthew James Beal. *Variational algorithms for approximate Bayesian inference*. 2003.

[17] EML Beale, CL Mallows, et al. "Scale mixing of symmetric distributions with zero means." In: *The Annals of Mathematical Statistics* 30.4 (1959), pp. 1145–1151.

[18] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. "Conditional computation in neural networks for faster models." In: *arXiv preprint arXiv:1511.06297* (2015).

[19] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." In: *arXiv preprint arXiv:1308.3432* (2013).

[20] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. "Theano: a CPU and GPU math expression compiler." In: *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. Austin, TX. 2010, p. 3.

[21] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[22] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. "Variational inference: A review for statisticians." In: *Journal of the American Statistical Association* (2017).

[23] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. "Weight Uncertainty in Neural Networks." In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (2015).

[24] Edwin V Bonilla, Felix V Agakov, and Christopher Williams. "Kernel multi-task learning using task-specific features." In: *International Conference on Artificial Intelligence and Statistics*. 2007, pp. 43–50.

[25] Edwin V Bonilla, Kian M Chai, and Christopher Williams. "Multi-task Gaussian process prediction." In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[26] Diana Cai, Nathanael Ackerman, Cameron Freer, et al. "Priors on exchangeable directed graphs." In: *Electronic Journal of Statistics* 10.2 (2016), pp. 3490–3515.

[27] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. "Deep learning with low precision by half-wave gaussian quantization." In: *arXiv preprint arXiv:1702.00953* (2017).

[28] Carlos M Carvalho, Nicholas G Polson, and James G Scott. "The horseshoe estimator for sparse signals." In: *Biometrika* 97.2 (2010), pp. 465–480.

[29] Sek Chai, Aswin Raghavan, David Zhang, Mohamed Amer, and Tim Shields. "Low Precision Neural Networks using Subband Decomposition." In: *arXiv preprint arXiv:1703.08595* (2017).

[30] Tianqi Chen, Emily B Fox, and Carlos Guestrin. "Stochastic Gradient Hamiltonian Monte Carlo." In.

[31] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. "Compressing convolutional neural networks." In: *arXiv preprint arXiv:1506.04449* (2015).

[32] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Variational lossy autoencoder." In: *arXiv preprint arXiv:1611.02731* (2016).

[33] Hyunsun Choi and Eric Jang. "Generative ensembles for robust anomaly detection." In: *arXiv preprint arXiv:1810.01392* (2018).

[34] François Chollet. *Keras*. https://github.com/fchollet/keras. 2015.

[35] Matthieu Courbariaux and Yoshua Bengio. "Binarynet: Training deep neural networks with weights and activations constrained to +1 or −1." In: *arXiv preprint arXiv:1602.02830* (2016).

[36] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations." In: *Advances in Neural Information Processing Systems*. 2015, pp. 3105–3113.

[37] Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. "Training deep neural networks with low precision multiplications." In: *arXiv preprint arXiv:1412.7024* (2014).

[38] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[39] Kurt Cutajar, Edwin V Bonilla, Pietro Michiardi, and Maurizio Filippone. "Random feature expansions for deep Gaussian processes." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 884–893.

[40] Andreas C. Damianou and Neil D. Lawrence. "Deep Gaussian Processes." In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*. 2013, pp. 207–215.

[41] Abhirup Datta, Sudipto Banerjee, Andrew O Finley, and Alan E Gelfand. "Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets." In: *Journal of the American Statistical Association* 111.514 (2016), pp. 800–812.

[42] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. "Predicting parameters in deep learning." In: *Advances in Neural Information Processing Systems*. 2013, pp. 2148–2156.

[43] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP." In: *arXiv preprint arXiv:1605.08803* (2016).

[44] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. "More is Less: A More Complicated Network with Less Inference Complexity." In: *arXiv preprint arXiv:1703.08651* (2017).

[45] David K. Duvenaud, Oren Rippel, Ryan P. Adams, and Zoubin Ghahramani. "Avoiding pathologies in very deep networks." In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*. 2014, pp. 202–210.

[46] Luca Falorsi, Pim de Haan, Tim R Davidson, and Patrick Forré. "Reparameterizing Distributions on Lie Groups." In: *arXiv preprint arXiv:1903.02958* (2019).

[47] Julian Faraone, Nicholas Fraser, Michaela Blott, and Philip HW Leong. "SYQ: Learning Symmetric Quantization For Efficient Deep Neural Networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4300–4309.

[48] Mario AT Figueiredo. "Adaptive sparseness using Jeffreys' prior." In: *Advances in neural information processing systems* 1 (2002), pp. 697–704.

[49] Yarin Gal and Zoubin Ghahramani. "Bayesian convolutional neural networks with Bernoulli approximate variational inference." In: *arXiv preprint arXiv:1506.02158* (2015).

[50] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." In: *international conference on machine learning*. 2016, pp. 1050–1059.

[51] Yarin Gal, Jiri Hron, and Alex Kendall. "Concrete Dropout." In: *arXiv preprint arXiv:1705.07832* (2017).

[52] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. "Neural processes." In: *arXiv preprint arXiv:1807.01622* (2018).

[53] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. "Compressing deep convolutional networks using vector quantization." In: *ICLR* (2015).

[54] Ian Goodfello, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[55] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." In: *arXiv preprint arXiv:1412.6572* (2014).

[56] Alex Graves. "Practical variational inference for neural networks." In: *Advances in Neural Information Processing Systems*. 2011, pp. 2348–2356.

[57] Alex Graves, Jacob Menick, and Aaron van den Oord. "Associative compression networks." In: *arXiv preprint arXiv:1804.02476* (2018).

[58] Peter D Grünwald. *The minimum description length principle*. 2007.

[59] Yiwen Guo, Anbang Yao, and Yurong Chen. "Dynamic Network Surgery for Efficient DNNs." In: *Advances In Neural Information Processing Systems*. 2016, pp. 1379–1387.

[60] Arjun K Gupta and Daya K Nagar. *Matrix variate distributions*. Vol. 104. CRC Press, 1999.

[61] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep learning with limited numerical precision." In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.

[62] Philipp Gysel. "Ristretto: Hardware-oriented approximation of convolutional neural networks." In: *Master's thesis, University of California* (2016).

[63] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks." In: *IEEE Transactions on Neural Networks and Learning Systems* (2018). DOI: 10.1109/TNNLS.2018.2808319.

[64] Danijar Hafner, Dustin Tran, Alex Irpan, Timothy Lillicrap, and James Davidson. "Reliable uncertainty estimates in deep neural networks using noise contrastive priors." In: *arXiv preprint arXiv:1807.09289* (2018).

[65] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *ICLR* (2016).

[66] Song Han, Jeff Pool, John Tran, and William Dally. "Learning both weights and connections for efficient neural networks." In: *Advances in Neural Information Processing Systems*. 2015, pp. 1135–1143.

[67] Markus Harva and Ata Kabán. "Variational learning for rectified factor analysis." In: *Signal Processing* 87.3 (2007), pp. 509–527.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1026–1034.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Identity Mappings in Deep Residual Networks." In: *arXiv preprint arXiv:1603.05027* (2016).

[71] James Hensman, Nicolas Durrande, Arno Solin, et al. "Variational Fourier Features for Gaussian Processes." In: *Journal of Machine Learning Research* 18 (2017), pp. 151–1.

[72] José Miguel Hernández-Lobato and Ryan Adams. "Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks." In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 2015, pp. 1861–1869.

[73] José Miguel Hernández-Lobato, Yingzhen Li, Daniel Hernández-Lobato, Thang Bui, and Richard E Turner. "Black-box α-divergence Minimization." In: *arXiv preprint arXiv:1511.03243* (2015).

[74] John R Hershey and Peder A Olsen. "Approximating the Kullback Leibler divergence between Gaussian mixture models." In: *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*. Vol. 4. IEEE. 2007, pp. IV–317.

[75] Geoffrey E Hinton and Zoubin Ghahramani. "Generative models for discovering sparse distributed representations." In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 352.1358 (1997), pp. 1177–1190.

[76] Geoffrey E Hinton and Drew Van Camp. "Keeping the neural networks simple by minimizing the description length of the weights." In: *Proceedings of the sixth annual conference on Computational learning theory*. ACM. 1993, pp. 5–13.

[77] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." In: *arXiv preprint arXiv:1503.02531* (2015).

[78] Antti Honkela and Harri Valpola. "Variational learning and bits-back coding: an information-theoretic view to Bayesian learning." In: *IEEE Transactions on Neural Networks* (2004).

[79] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." In: *arXiv preprint arXiv:1704.04861* (2017).

[80] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized neural networks: Training neural networks with low precision weights and activations." In: *arXiv preprint arXiv:1609.07061* (2016).

[81] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." In: *ICLR* (2017).

[82] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *arXiv preprint arXiv:1502.03167* (2015).

[83] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." In: *arXiv preprint arXiv:1712.05877* (2017).

[84] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural tangent kernel: Convergence and generalization in neural networks." In: *Advances in neural information processing systems*. 2018.

[85] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax." In: *arXiv preprint arXiv:1611.01144* (2016).

[86] Theofanis Karaletsos and Gunnar Rätsch. "Automatic Relevance Determination For Deep Generative Models." In: *arXiv preprint arXiv:1505.07765* (2015).

[87] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. "Attentive neural processes." In: *arXiv preprint arXiv:1901.05761* (2019).

[88] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[89] Diederik P Kingma, Tim Salimans, and Max Welling. "Variational dropout and the local reparametrization trick." In: *Advances in Neural Information Processing Systems* (2015), pp. 2575–2583.

[90] Diederik P Kingma, Tim Salimans, and Max Welling. "Improving variational inference with inverse autoregressive flow." In: *arXiv preprint arXiv:1606.04934* (2016).

[91] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." In: *International Conference on Learning Representations (ICLR)* (2014).

[92] Achim Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.

[93] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." In: *ICML deep learning workshop*. 2015.

[94] Anoop Korattikara, Vivek Rathod, Kevin Murphy, and Max Welling. "Bayesian dark knowledge." In: *arXiv preprint arXiv:1506.04416* (2015).

[95] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. 2009.

[96] Liangzhen Lai, Naveen Suda, and Vikas Chandra. "Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations." In: *arXiv preprint arXiv:1703.03073* (2017).

[97] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles." In: *arXiv preprint arXiv:1612.01474* (2016).

[98] Neil D Lawrence. "Note Relevance Determination." In: *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 128–133.

[99] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[100] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998.

[101] Yann LeCun, John S Denker, and Sara A Solla. "Optimal Brain Damage." In: *Advances in neural information processing systems 2, NIPS 1989*. Vol. 2. Morgan-Kaufmann Publishers. 1990, pp. 598–605.

[102] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Jascha Sohl-Dickstein, and Jeffrey Pennington. "Wide neural networks of any depth evolve as linear models under gradient descent." In: *arXiv preprint arXiv:1902.06720* (2019).

[103] Fengfu Li, Bo Zhang, and Bin Liu. "Ternary weight networks." In: *arXiv preprint arXiv:1605.04711* (2016).

[104] Shiyu Liang, Yixuan Li, and R Srikant. "Enhancing the reliability of out-of-distribution image detection in neural networks." In: *arXiv preprint arXiv:1706.02690* (2017).

[105] Darryl D Lin and Sachin S Talathi. "Overcoming challenges in fixed point training of deep convolutional networks." In: *Workshop ICML* (2016).

[106] Darryl D Lin, Sachin S Talathi, and V Sreekanth Annapureddy. "Fixed Point Quantization of Deep Convolutional Networks." In: *arXiv preprint arXiv:1511.06393* (2015).

[107] Christos Louizos. "Smart Regularization of Deep Architectures." In: *Master's thesis, University of Amsterdam* (2015).

[108] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. "Relaxed quantization for discretized neural networks." In: *Proceedings of the 7th International Conference on Learning Representations (ICLR)* (2019).

[109] Christos Louizos, Uri Shalit, Joris M Mooij, David Sontag, Richard Zemel, and Max Welling. "Causal effect inference with deep latent-variable models." In: *Advances in Neural Information Processing Systems*. 2017, pp. 6446–6456.

[110] Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. "The Functional Neural Process." In: *Arxiv* (2019).

[111] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. "The variational fair autoencoder." In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)* (2016).

[112] Christos Louizos, Karen Ullrich, and Max Welling. "Bayesian compression for deep learning." In: *Advances in Neural Information Processing Systems*. 2017, pp. 3288–3298.

[113] Christos Louizos and Max Welling. "Structured and efficient variational deep learning with matrix gaussian posteriors." In: *International Conference on Machine Learning*. 2016, pp. 1708–1716.

[114] Christos Louizos and Max Welling. "Multiplicative normalizing flows for variational bayesian neural networks." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2218–2227.

[115] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization." In: *Proceedings of the 6th International Conference on Learning Representations (ICLR)* (2018).

[116] Chao Ma, Yingzhen Li, and José Miguel Hernández-Lobato. "Variational implicit processes." In: *arXiv preprint arXiv:1806.02390* (2018).

[117] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. "Auxiliary Deep Generative Models." In: *arXiv preprint arXiv:1602.05473* (2016).

[118] David JC MacKay. "A practical Bayesian framework for backpropagation networks." In: *Neural computation* 4.3 (1992), pp. 448–472.

[119] David JC MacKay. "Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks." In: *Network: Computation in Neural Systems* 6.3 (1995), pp. 469–505.

[120] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables." In: *arXiv preprint arXiv:1611.00712* (2016).

[121] César Lincoln C Mattos, Zhenwen Dai, Andreas Damianou, Jeremy Forth, Guilherme A Barreto, and Neil D Lawrence. "Recurrent gaussian processes." In: *arXiv preprint arXiv:1511.06644* (2015).

[122] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. "Ternary Neural Networks with Fine-Grained Quantization." In: *arXiv preprint arXiv:1705.01462* (2017).

[123] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. "Deep neural networks are robust to weight binarization and other non-linear distortions." In: *arXiv preprint arXiv:1606.01981* (2016).

[124] Thomas P Minka. "Expectation propagation for approximate Bayesian inference." In: *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2001, pp. 362–369.

[125] Asit Mishra and Debbie Marr. "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy." In: *arXiv preprint arXiv:1711.05852* (2017).

[126] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. "WRPN: wide reduced-precision networks." In: *arXiv preprint arXiv:1709.01134* (2017).

[127] Toby J Mitchell and John J Beauchamp. "Bayesian variable selection in linear regression." In: *Journal of the American Statistical Association* 83.404 (1988), pp. 1023–1032.

[128] Andriy Mnih and Karol Gregor. "Neural variational inference and learning in belief networks." In: *arXiv preprint arXiv:1402.0030* (2014).

[129] Andriy Mnih and Danilo Rezende. "Variational inference for Monte Carlo objectives." In: *International Conference on Machine Learning*. 2016, pp. 2188–2196.

[130] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. "Variational Dropout Sparsifies Deep Neural Networks." In: *arXiv preprint arXiv:1701.05369* (2017).

[131] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814.

[132] Eric Nalisnick, Anima Anandkumar, and Padhraic Smyth. "A scale mixture perspective of multiplicative noise in neural networks." In: *arXiv preprint arXiv:1506.03208* (2015).

[133] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. "Do Deep Generative Models Know What They Don't Know?" In: *arXiv preprint arXiv:1810.09136* (2018).

[134] Radford M Neal. "Bayesian learning for neural networks." PhD thesis. Citeseer, 1995.

[135] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. "Structured Bayesian Pruning via Log-Normal Multiplicative Noise." In: *arXiv preprint arXiv:1705.07283* (2017).

[136] Sarah E Neville, John T Ormerod, MP Wand, et al. "Mean field variational Bayes for continuous sparse signal shrinkage: pitfalls and remedies." In: *Electronic Journal of Statistics* 8.1 (2014), pp. 1113–1151.

[137] Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. "Bayesian deep convolutional networks with many channels are gaussian processes." In: (2018).

[138] Peter Orbanz and Daniel M Roy. "Bayesian models of graphs, arrays and other exchangeable random structures." In: *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2015), pp. 437–461.

[139] Marc Ortiz, Adrián Cristal, Eduard Ayguadé, and Marc Casas. "Low-Precision Floating-Point Schemes for Neural Network Training." In: *arXiv preprint arXiv:1804.05267* (2018).

[140] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep exploration via bootstrapped DQN." In: *Advances in neural information processing systems*. 2016, pp. 4026–4034.

[141] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift." In: *arXiv preprint arXiv:1906.02530* (2019).

[142] Omiros Papaspiliopoulos, Gareth O Roberts, and Martin Sköld. "A general framework for the parametrization of hierarchical models." In: *Statistical Science* (2007), pp. 59–73.

[143] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. "cleverhans v1.0.0: an adversarial machine learning library." In: *arXiv preprint arXiv:1610.00768* (2016).

[144] Jorn WT Peters and Max Welling. "Probabilistic Binary Neural Networks." In: *arXiv preprint arXiv:1809.03368* (2018).

[145] Carsten Peterson. "A mean field theory learning algorithm for neural networks." In: *Complex systems* 1 (1987), pp. 995–1019.

[146] Antonio Polino, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization." In: *arXiv preprint arXiv:1802.05668* (2018).

[147] Rajesh Ranganath, Dustin Tran, and David M Blei. "Hierarchical Variational Models." In: *arXiv preprint arXiv:1511.02386* (2015).

[148] Carl Edward Rasmussen. "Gaussian processes in machine learning." In: *Summer School on Machine Learning*. Springer. 2003, pp. 63–71.

[149] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[150] Danilo Jimenez Rezende and Shakir Mohamed. "Variational inference with normalizing flows." In: *arXiv preprint arXiv:1505.05770* (2015).

[151] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models." In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. 2014, pp. 1278–1286.

[152] Jorma Rissanen. "Modelling by shortest data description." In: *Automatica* (1978).

[153] Jorma Rissanen. "Stochastic complexity and modelling." In: *The annals of statistics* (1986).

[154] Jason Tyler Rolfe. "Discrete Variational Autoencoders." In: *arXiv preprint arXiv:1609.02200* (2016).

[155] Tim Salimans. "A Structured Variational Auto-encoder for Learning Deep Hierarchies of Sparse Features." In: *arXiv preprint arXiv:1602.08734* (2016).

[156] Tim Salimans, Diederik P Kingma, Max Welling, et al. "Markov chain Monte Carlo and variational inference: Bridging the gap." In: *International Conference on Machine Learning*. 2015, pp. 1218–1226.

[157] Tim Salimans, David A Knowles, et al. "Fixed-form variational posterior approximation through stochastic linear regression." In: *Bayesian Analysis* 8.4 (2013), pp. 837–882.

[158] Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. "Group Sparse Regularization for Deep Neural Networks." In: *arXiv preprint arXiv:1607.00485* (2016).

[159] Gideon Schwarz et al. “Estimating the dimension of a model.” In: *The annals of statistics* 6.2 (1978), pp. 461–464.

[160] Moshe Shaked and J George Shanthikumar. *Stochastic orders*. Springer Science & Business Media, 2007.

[161] Oran Shayer, Dan Levi, and Ethan Fetaya. “Learning Discrete Weights Using the Local Reparameterization Trick.” In: *International Conference on Learning Representations*. 2018.

[162] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. “A Quantization-Friendly Separable Convolution for MobileNets.” In: (2018).

[163] Jiaxin Shi, Shengyang Sun, and Jun Zhu. “Kernel implicit variational inference.” In: *arXiv preprint arXiv:1705.10119* (2017).

[164] Shaohuai Shi and Xiaowen Chu. “Speeding up Convolutional Neural Networks By Exploiting the Sparsity of Rectifier Units.” In: *arXiv preprint arXiv:1704.07724* (2017).

[165] Patrice Y Simard, Dave Steinkraus, and John C Platt. “Best practices for convolutional neural networks applied to visual document analysis.” In: IEEE. 2003, p. 958.

[166] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition.” In: *ICLR* (2015).

[167] More Sites. “Ieee standard for floating-point arithmetic.” In: (2008).

[168] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical networks for few-shot learning.” In: *NIPS*. 2017.

[169] Edward Snelson and Zoubin Ghahramani. “Sparse Gaussian processes using pseudo-inputs.” In: *Advances in neural information processing systems*. 2006, pp. 1257–1264.

[170] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. “Ladder Variational Autoencoders.” In: *arXiv preprint arXiv:1602.02282* (2016).

[171] Suraj Srinivas and R Venkatesh Babu. “Generalized dropout.” In: *arXiv preprint arXiv:1611.06791* (2016).

[172] Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. “Training sparse neural networks.” In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE. 2017, pp. 455–462.

[173] Nitish Srivastava. *Improving neural networks with dropout*. 2013.

[174] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A simple way to prevent neural networks from overfitting.” In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[175] Joe Staines and David Barber. "Variational optimization." In: *arXiv preprint arXiv:1212.4507* (2012).

[176] Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. "Functional Variational Bayesian Neural Networks." In: *arXiv preprint arXiv:1903.05779* (2019).

[177] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. "Learning to compare: Relation network for few-shot learning." In: 2018.

[178] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." In: *arXiv preprint arXiv:1703.09039* (2017).

[179] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks." In: *arXiv preprint arXiv:1312.6199* (2013).

[180] Da Tang, Dawen Liang, Tony Jebara, and Nicholas Ruozzi. "Correlated Variational Auto-Encoders." In: *arXiv preprint arXiv:1905.05335* (2019).

[181] Robert Tibshirani. "Regression shrinkage and selection via the lasso." In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288.

[182] Michalis Titsias. "Variational learning of inducing variables in sparse Gaussian processes." In: *Artificial Intelligence and Statistics*. 2009, pp. 567–574.

[183] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. "Efficient object localization using convolutional networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 648–656.

[184] George Tucker, Andriy Mnih, Chris J Maddison, and Jascha Sohl-Dickstein. "REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models." In: *arXiv preprint arXiv:1703.07370* (2017).

[185] Karen Ullrich, Edward Meeds, and Max Welling. "Soft Weight-Sharing for Neural Network Compression." In: *ICLR* (2017).

[186] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. "Accelerating Deep Convolutional Networks using low-precision and sparsity." In: *arXiv preprint arXiv:1610.00324* (2016).

[187] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. "Matching networks for one shot learning." In: *Advances in neural information processing systems*. 2016, pp. 3630–3638.

[188] Chris S Wallace. "Classification by minimum-message-length inference." In: *International Conference on Computing and Information* (1990).

[189] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. "Regularization of neural networks using dropconnect." In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 1058–1066.

[190] Sida Wang and Christopher Manning. “Fast dropout training.” In: *Proceedings of The 30th International Conference on Machine Learning*. 2013, pp. 118–126.

[191] Max Welling and Yee W Teh. “Bayesian learning via stochastic gradient Langevin dynamics.” In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 681–688.

[192] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. “Learning structured sparsity in deep neural networks.” In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[193] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. “Flipout: Efficient pseudo-independent weight perturbations on mini-batches.” In: *arXiv preprint arXiv:1803.04386* (2018).

[194] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. “Convolutional gaussian processes.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 2849–2858.

[195] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning.” In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[196] Andrew G Wilson, Zhiting Hu, Ruslan R Salakhutdinov, and Eric P Xing. “Stochastic variational deep kernel learning.” In: *Advances in Neural Information Processing Systems*. 2016, pp. 2586–2594.

[197] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. “Deep kernel learning.” In: *Artificial Intelligence and Statistics*. 2016, pp. 370–378.

[198] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. “Training and Inference with Integers in Deep Neural Networks.” In: *International Conference on Learning Representations*. 2018.

[199] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. “Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning.” In: *CVPR* (2017).

[200] Kai Yu, Wei Chu, Shipeng Yu, Volker Tresp, and Zhao Xu. “Stochastic relational models for discriminative link prediction.” In: *Advances in neural information processing systems*. 2006, pp. 1553–1560.

[201] Ming Yuang and Yi Lin. “Model selection and estimation in regression with grouped variables.” In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* (2006).

[202] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks.” In: *arXiv preprint arXiv:1605.07146* (2016).

[203] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. “Understanding deep learning requires rethinking generalization.” In: *arXiv preprint arXiv:1611.03530* (2016).

[204] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. "Noisy natural gradient as variational inference." In: *arXiv preprint arXiv:1712.02390* (2017).

[205] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. "Incremental network quantization: Towards lossless cnns with low-precision weights." In: *arXiv preprint arXiv:1702.03044* (2017).

[206] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. "Explicit Loss-Error-Aware Quantization for Low-Bit Deep Neural Networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9426–9435.

[207] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients." In: *arXiv preprint arXiv:1606.06160* (2016).

[208] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. "Trained Ternary Quantization." In: *ICLR* (2017).

10

APPENDICES

10.1 NOTATION

Symbol	Description
$\mathbf{A} - \mathbf{Z}$	Capital bold letters denote matrices or higher order tensors.
$\mathbf{a} - \mathbf{z}$	Small bold letters denote vectors.
$a - z$	Regular letters denote scalars.
$p(\cdot), q(\cdot)$	Both are used in order to denote either continuous or discrete probability distributions. With $p(\cdot)$ we denote distributions that are exact, whereas with $q(\cdot)$ we denote distributions that approximate intractable distributions $p(\cdot)$.
$P(\cdot), Q(\cdot)$	Both are used in order to denote the cumulative distribution functions (CDFs) of $p(\cdot), q(\cdot)$ respectively. The CDF is defined as $P(x \leqslant a) = \int_{-\infty}^{a} p(t)dt$.
$p_\theta(\cdot), q_\phi(\cdot)$	Both are used in order to denote probability distributions $p(\cdot), q(\cdot)$ with parameters θ, ϕ respectively.
KL	The KL-divergence from a distribution $p(x)$ to $q(x)$, $KL(p(x)\|\|q(x)) = \int p(x) \log \frac{p(x)}{q(x)} dx$.
$\mathcal{H}(q(x))$ or $\mathcal{H}_q$	The entropy of a given distribution, in this case $q(x)$, being either $-\sum_x q(x) \log q(x)$ for when $q(x)$ is discrete or $-\int q(x) \log q(x) dx$ when $q(x)$ is continuous.

10.2 APPENDIX OF CHAPTER 2

10.2.1 *KL divergence between matrix variate Gaussian prior and posterior*

Let $\mathcal{MN}_0(\mathbf{M}_0, \mathbf{U}_0, \mathbf{V}_0)$ and $\mathcal{MN}_1(\mathbf{M}_1, \mathbf{U}_1, \mathbf{V}_1)$ be two matrix variate Gaussian distributions for random matrices of size $n \times p$. We can use the fact that the matrix variate Gaussian is a multivariate Gaussian if we flatten the matrix, i.e. $\mathcal{MN}_0(\mathbf{M}_0, \mathbf{U}_0, \mathbf{V}_0) =$

$\mathcal{N}_0(\text{vec}(\mathbf{M}_0), \mathbf{V}_0 \otimes \mathbf{U}_0)$, and as a result use the KL-divergence between two multivariate Gaussians:

$$\begin{aligned}
\text{KL}(\mathcal{N}_0||\mathcal{N}_1) &= \frac{1}{2}\Bigg(\text{tr}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - K + \log \frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|} \Bigg) \\
&= \frac{1}{2}\Bigg(\text{tr}\left((\mathbf{V}_1 \otimes \mathbf{U}_1)^{-1}(\mathbf{V}_0 \otimes \mathbf{U}_0)\right) + \\
&\quad + \left(\text{vec}(\mathbf{M}_1) - \text{vec}(\mathbf{M}_0)\right)^\top \left(\mathbf{V}_1 \otimes \mathbf{U}_1\right)^{-1} \left(\text{vec}(\mathbf{M}_1) - \text{vec}(\mathbf{M}_0)\right) - \\
&\quad - np + \log \frac{|\mathbf{V}_1 \otimes \mathbf{U}_1|}{|\mathbf{V}_0 \otimes \mathbf{U}_0|} \Bigg)
\end{aligned}$$

Now to compute each term in the KL efficiently we need to use some properties of the vectorization and Kronecker product:

$$\begin{aligned}
t_a &= \text{tr}\left((\mathbf{V}_1 \otimes \mathbf{U}_1)^{-1}(\mathbf{V}_0 \otimes \mathbf{U}_0)\right) = \text{tr}\left((\mathbf{V}_1^{-1} \otimes \mathbf{U}_1^{-1})(\mathbf{V}_0 \otimes \mathbf{U}_0)\right) \\
&= \text{tr}\left((\mathbf{V}_1^{-1}\mathbf{V}_0) \otimes (\mathbf{U}_1^{-1}\mathbf{U}_0)\right) = \text{tr}(\mathbf{U}_1^{-1}\mathbf{U}_0)\,\text{tr}(\mathbf{V}_1^{-1}\mathbf{V}_0)
\end{aligned} \tag{120}$$

$$\begin{aligned}
t_b &= \left(\text{vec}(\mathbf{M}_1) - \text{vec}(\mathbf{M}_0)\right)^\top \left(\mathbf{V}_1 \otimes \mathbf{U}_1\right)^{-1} \left(\text{vec}(\mathbf{M}_1) - \text{vec}(\mathbf{M}_0)\right) \\
&= \text{vec}(\mathbf{M}_1 - \mathbf{M}_0)^\top (\mathbf{V}_1^{-1} \otimes \mathbf{U}_1^{-1})\,\text{vec}(\mathbf{M}_1 - \mathbf{M}_0) \\
&= \text{vec}(\mathbf{M}_1 - \mathbf{M}_0)^\top \,\text{vec}(\mathbf{U}_1^{-1}(\mathbf{M}_1 - \mathbf{M}_0)\mathbf{V}_1^{-1}) \\
&= \text{tr}\left((\mathbf{M}_1 - \mathbf{M}_0)^\top \mathbf{U}_1^{-1}(\mathbf{M}_1 - \mathbf{M}_0)\mathbf{V}_1^{-1}\right)
\end{aligned} \tag{121}$$

$$\begin{aligned}
t_c &= \log \frac{|\mathbf{V}_1 \otimes \mathbf{U}_1|}{|\mathbf{V}_0 \otimes \mathbf{U}_0|} = \log \frac{|\mathbf{U}_1|^p |\mathbf{V}_1|^n}{|\mathbf{U}_0|^p |\mathbf{V}_0|^n} \\
&= p \log |\mathbf{U}_1| + n \log |\mathbf{V}_1| - p \log |\mathbf{U}_0| - n \log |\mathbf{V}_0|
\end{aligned} \tag{122}$$

So putting everything together we have that:

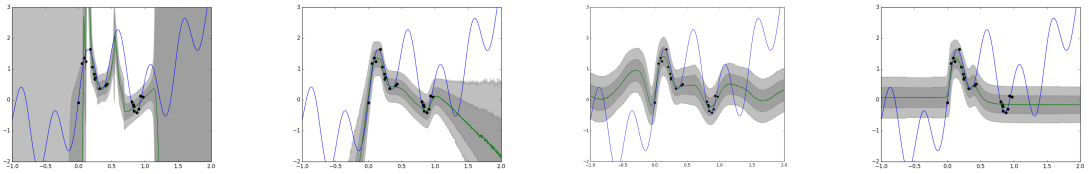
$$\begin{aligned}
\text{KL}(\mathcal{MN}_0, \mathcal{MN}_1) = \frac{1}{2}\Bigg(& \text{tr}(\mathbf{U}_1^{-1}\mathbf{U}_0)\,\text{tr}(\mathbf{V}_1^{-1}\mathbf{V}_0) + \\
& + \text{tr}\left((\mathbf{M}_1 - \mathbf{M}_0)^\top \mathbf{U}_1^{-1}(\mathbf{M}_1 - \mathbf{M}_0)\mathbf{V}_1^{-1}\right) - np \\
& + p \log |\mathbf{U}_1| + n \log |\mathbf{V}_1| - p \log |\mathbf{U}_0| - n \log |\mathbf{V}_0| \Bigg)
\end{aligned} \tag{123}$$

10.2.2 *Different toy dataset*

We also performed an experiment with a different toy dataset that was employed in [140]. We generated 12 inputs from $U[0, 0.6]$ and 8 inputs from $U[0.8, 1]$. We then transform those inputs via:

$$y_i = x_i + \epsilon_i + \sin(4(x_i + \epsilon_i)) + \sin(13(x_i + \epsilon_i))$$

where $\epsilon_i \sim \mathcal{N}(0, 0.0009)$. We continued in fitting four neural networks that had two hidden-layers with 50 units each. The first was trained with probabilistic back-propagation [72], and the remaining three with our model while varying the nonlinearities among the layers: we used ReLU, cosine and hyperbolic tangent activations. For our model we set the upper bound of the variational dropout rate to 0.2 and we used 2 pseudo data pairs for the input layer and 4 for the rest. The resulting predictive distributions can be seen at Figure 26.



(a) PBP (b) MG ReLU (c) MG cosine (d) MG tanh

Figure 26: Predictive distributions for the toy dataset. Grey areas correspond to $\pm\{1, 2\}$ standard deviations around the mean function.

10.3 APPENDIX OF CHAPTER 4

10.3.1 *Experimental details*

Throughout the experiments, the architectures for the FNP and FNP$^+$ were constructed as follows. We used a neural network torso in order to obtain an intermediate hidden representation $\mathbf{h}$ of the inputs $\mathbf{x}$ and then parametrized two linear output layers, one that lead to the parameters of $p(\mathbf{u}|\mathbf{x})$ and one that lead to the parameters of $q(\mathbf{z}|\mathbf{x})$, both of which were fully factorized Gaussians. The function $g(\cdot, \cdot)$ for the Bernoulli probabilities was set to an RBF, i.e. $g(\mathbf{u}_i, \mathbf{u}_j) = \exp(-.5\tau\|\mathbf{u}_i - \mathbf{u}_j\|^2)$, where τ was optimized to maximize the lower bound. The temperature of the binary concrete / Gumbel-softmax relaxation was kept at 0.3 throughout training and we used the log CDF of a standard normal as the $\tau_k(\cdot)$ for $\mathbf{G}$. For the classifiers $p(y|\mathbf{z}), p(y|\mathbf{z}, \mathbf{u})$ we used a linear model that operated on top of $\text{ReLU}(\mathbf{z})$ or $\text{ReLU}([\mathbf{z}, \mathbf{u}])$ respectively. We used a single Monte Carlo sample for each batch during training in order to estimate the bound of FNPs. We similarly used a single sample for the NP, MC-dropout and the variationally trained Bayesian neural network (VI BNN). All of the models were implemented in PyTorch and were run across six Titan X (Pascal) GPUs (one GPU per model).

The NN, MC-dropout and VI BNN had the same torso and classifier as the FNPs. As the NP has not been previously employed in the settings we considered, we designed the architecture in a way that is similar to the FNP. More specifically, we used the same neural network torso to provide an intermediate representation $\mathbf{h}$ for the inputs $\mathbf{x}$. To obtain the global embedding $\mathbf{r}$ we concatenated the labels $\mathbf{y}$ to obtain $\tilde{\mathbf{h}} = [\mathbf{h}, \mathbf{y}]$, projected $\tilde{\mathbf{h}}$ to 256 dimensions with a linear layer and then computed the average of each dimension across the context. The parameters of the distribution over the global latent variables θ were then given by a linear layer acting on top of $\text{ReLU}(\mathbf{r})$. After sampling θ we then used a linear classifier that operated on top of $[\mathbf{h}, \text{ReLU}(\theta)]$.

In the regression experiment for the initial transformation of x we used 100 ReLUs for both NP and FNP models via a single layer MLP, whereas for the regressor we used a linear layer for NP (more capacity lead to overfitting and a decrease in predictive uncertainty) and a single hidden layer MLP of 100 ReLUs for the FNPs. For the MC-dropout network used a single hidden layer MLP of 100 units and we applied dropout with a rate of 0.5 at the hidden layer. In all of the neural networks models, the heteroscedastic noise was parametrized according to $\sigma = .1 + 0.9\log(1 + \exp(d))$, where d was a neural network output. For the GP, we optimized the kernel lengthscale according to the marginal likelihood. We also found it beneficial to apply a soft-free bits [32] modification of the bound to help with the optimization of $\mathbf{z}$, where we allowed 1 free bit on average across all dimensions and batch elements for the FNP and 4 for the FNP$^+$.

For the MNIST experiment, the model architecture was a 20C5 - MP2 - 50C5 - MP2 - 500FC - Softmax, where 20C5 corresponds to a convolutional layer of 20 output feature maps with a kernel size of 5, MP2 corresponds to max pooling with a size of 2, 500FC corresponds to fully connected layer of 500 output units and Softmax corresponds to the output layer. The initial representation of $\mathbf{x}$ for the NP and FNPs was provided by the penultimate layer of the network. For the MC-dropout network we applied 0.5 dropout to every layer, whereas for the VI BNN we performed variational inference over all of the parameters. The number of points in R was set to 300, a value that was determined from a range of $[50, 100, 200, 300, 500]$ by judging the performance of the NP and FNP models on the MNIST / notMNIST pair. For the FNP we used minibatches of 100 points from M, while we always appended the full R to each of those batches. For the NP, since we were using a random set of contexts every time, we used a batch size of 400 points, where, in order to be comparable to the FNP, we randomly selected up to 300 points from the current batch for the context points during training and used the same 300 points as FNP for evaluation. We set the upper bound of training epochs for the FNPs, NN, MC-dropout and VI BNN networks to 100 epochs, and 200 epochs for the NP as it did less parameter updates per epoch than the others. Optimization was done with Adam [88] using the default hyperparameters. We further did early stopping according to the accuracy on the validation set and no other regularization was employed. Finally, we also employed a soft-free bits [32] modification of the bound to help with the optimization; for the $\mathbf{z}$ of FNPs we allowed 1 free bit on average across all dimensions and batch elements throughout training whereas for the VI BNN we allowed 1 free bit on average across all of the parameters.

The architecture for the CIFAR 10 experiment was a 2x(128C3) - MP2 - 2x(256C3) - MP2 - 2x(512C3) - MP2 - 1024FC - Softmax along with batch normalization [82] employed after every layer (besides the output one). Similarly to the MNIST experiment, the initial representation of $\mathbf{x}$ for the NP and FNPs was provided by the penultimate layer of each of the networks. We didn't optimize any hyperparameters for these experiments and used the same number of reference points, free bits, amount of epochs, regularization and early stopping criteria we used at MNIST. For the MC-dropout network we applied dropout with a rate of 0.2 at the beginning of each stack of convolutional layers that shared the same output channels and with a rate of 0.5 before every fully connected layer. For the VI BNN, we performed variational inference over the layers were dropout was originally employed in the MC-dropout network, and we performed maximum likelihood for the rest.

Optimization was done with Adam with an initial learning rate of 0.001 that was decayed by a factor of 10 every thirty epochs for the NN, MC-Dropout, VI BNN and FNPs and every 60 epochs for the NP. We also performed data augmentation during training by doing random cropping with a padding of 4 pixels and random horizontal flips for both the reference and other points. We did not do any data augmentation during test time. The images were further normalized by subtracting the mean and by dividing with the standard deviation of each channel, computed across the training dataset.

10.3.2 *Ablation study on MNIST*

In this section we provide the additional results we obtained on MNIST during the ablation study. The discussion of the results can be found in the main text. We measured the sensitivity of NPs and FNPs to the size of the reference set R, the trade-offs we obtain by varying the dimensionalities of $\mathbf{u}, \mathbf{z}$ for the FNP$^+$, and finally the deviation of the scores for the FNPs after 5 replications, where on each one we select a different subset of size 300 (the size we used for the experiments in the main paper) of the training set as R. The results can be seen at Table 13, Table 14 and Table 15.

Table 13: Test error and uncertainty quality as a function of the size of the reference set R. For the o.o.d. entropy and AUCR we report the mean and standard error across all of the o.o.d. datasets.

(a) Error %

# R	NP	FNP	FNP$^+$
50	0.6	30.6	0.7
100	0.6	0.9	0.9
200	0.4	0.8	0.7
500	0.5	0.9	0.7

(b) o.o.d. entropy

# R	NP	FNP	FNP$^+$
50	1.0±0.2	2.1±0.0	1.6±0.1
100	1.4±0.3	1.8±0.1	2.0±0.1
200	0.9±0.2	1.8±0.1	1.6±0.1
500	0.8±0.1	1.7±0.1	1.3±0.1

(c) AUCR

# R	NP	FNP	FNP$^+$
50	99.4±0.3	80.0±0.3	99.7±0.2
100	99.7±0.2	99.6±0.1	99.9±0.1
200	99.5±0.2	99.8±0.1	99.8±0.1
500	99.5±0.2	99.8±0.1	99.4±0.3

Table 14: Test error and uncertainty quality as a function of the size of $\mathbf{u}, \mathbf{z}$ for the FNP$^+$. We used the same R as the one for the experiments in the main text.

$\mathbf{u}, \mathbf{z}$	Error %	o.o.d. entropy	AUCR
$32, 64$	0.7	1.8±0.1	99.9±0.1
$64, 32$	0.7	1.2±0.3	99.3±0.6
$16, 80$	0.7	2.0±0.1	99.6±0.4
$8, 88$	0.8	1.1±0.0	99.4±0.1
$2, 94$	4.7	0.4±0.0	91.4±2.2

Table 15: Average and standard error for the FNP models after 5 replications with different reference sets R of size 300.

	FNP	FNP$^+$
MNIST	0.02± 0.0 / 0.7±0.0	0.02±0.0 / 0.7±0.0
nMNIST	1.95±0.06 / 99.93±0.03	1.97±0.05 / 99.97± 0.02
fMNIST	1.69±0.05 / 99.43±0.10	1.63±0.04 / 99.58±0.07
Omniglot	1.88±0.04 / 99.86±0.04	1.85±0.06 / 99.90±0.03
Gaussian	1.95±0.14 / 99.81±0.16	2.07±0.02 / 99.98±0.02
Uniform	1.99±0.06 / 99.96±0.02	1.95±0.06 / 99.96±0.02
CIFAR10	0.17±0.01 / 7.5±0.08	0.08±0.01 / 7.3±0.04
SVHN	0.86±0.05 / 90.74±0.81	0.51±0.04 / 91.3±0.76
tImag32	1.22±0.02 / 94.49±0.29	0.69±0.02 / 92.6±0.39
iSUN	1.33±0.02 / 95.71±0.24	0.75±0.02 / 93.8±0.38
Gaussian	1.05±0.10 / 93.73±1.29	0.60±0.08 / 93.6±1.09
Uniform	0.85±0.16 / 89.43±4.20	0.61±0.11 / 93.4±1.89

10.3.3 *The Functional Neural Process is an exchangeable stochastic process*

Proposition. *The distributions defined in Eq.8, 9 define valid permutation invariant stochastic processes, hence they correspond to Bayesian models.*

Proof. In order to prove the proposition we will rely on de Finetti's and Kolmogorov Extension Theorems [92] and show that $p(\mathbf{y}_{\mathcal{D}}|\mathbf{X}_{\mathcal{D}})$ is permutation invariant and its marginal distributions are consistent under marginalization. We will focus on FNP as the proof for FNP$^+$ is analogous. As a reminder, we previously defined R to be a set of reference inputs $R = \{\mathbf{x}^r_1, \ldots, \mathbf{x}^r_K\}$, we defined $\mathcal{D}_X$ to be the set of observed inputs, and we also defined the auxiliary sets $M = \mathcal{D}_X \setminus R$, the set of all inputs in the observed dataset that are not a part of the reference set R, and $B = R \cup M$, the set of all points in the reference and observed dataset.

We will start with the permutation invariance. It will suffice to show that each of the individual probability densities described at Section 2.1 are permutation equivariant, as the products / sums will then make the overall probability permutation invariant. Without loss of generality we will assume that the elements in the set B are arranged as $B = \{\mathcal{D}_X, R \setminus \mathcal{D}_X\}$. Consider applying a permutation $\sigma(\cdot)$ over $\mathcal{D}$, $\tilde{\mathcal{D}} = \sigma(\mathcal{D})$; this will also induce the same permutation over $\mathcal{D}_X$, hence we will have that $\sigma(B) = \{\sigma(\mathcal{D}_X), R \setminus \mathcal{D}_X\}$. Now consider the fact that in the FNP each individual $\mathbf{u}_i$ is a function, let it be $f(\cdot)$, of the values of $\mathbf{x}_i$; as a result we will have that:

$$f(\sigma(B)) = \sigma(f(B)), \tag{124}$$

i.e. the latent variables $\mathbf{u}$ are permutation equivariant w.r.t. $\mathtt{B}$. Continuing to the latent adjacency matrices $\mathbf{G}, \mathbf{A}$; in the FNP each particular element i, j of these is a function of the values of the specific $\mathbf{u}_i, \mathbf{u}_j$. As a result, we will also have permutation equivariance for the rows / columns of $\mathbf{G}, \mathbf{A}$. Now since $\mathbf{G}, \mathbf{A}$ are essentially used as a way to factorize the joint distribution over the $\mathbf{z}_i$ in $\mathtt{B}$ and given the fact that the distribution of each $\mathbf{z}_i$ is invariant to the permutation of its parents, we will have that the permutation of $\mathtt{B}$ will result into the same re-ordering of the $\mathbf{z}_i$'s i.e.:

$$\sigma(\mathbf{Z}_{\mathtt{B}}) = g(\sigma(\mathtt{B})), \tag{125}$$

where $g(\cdot)$ is the function that maps $\mathtt{B}$ to $\mathbf{Z}_{\mathtt{B}}$. Finally, as each y_i is a function, let it be $h(\cdot)$, of the specific $\mathbf{z}_i$, we will similarly have that $\sigma(\mathbf{y}_{\mathtt{B}}) = h(\sigma(\mathtt{B}))$. We have thus described that all of the aforementioned random variables are permutation equivariant to $\mathtt{B}$ and as a result, due to the permutation invariant product / integral / summation operators, we will have that the FNP model is permutation invariant.

Continuing to the consistency under marginalization. Following [41] let us define $\tilde{\mathcal{D}} = \mathcal{D} \cup \{(\mathbf{x}_0, y_0)\}$ and consider two cases, one where the $\mathbf{x}_0$ belongs in $\mathtt{R}$ and one where it doesn't. We will show that in both cases $\int p(\mathbf{y}_{\tilde{\mathcal{D}}}|\mathbf{X}_{\tilde{\mathcal{D}}})dy_0 = p(\mathbf{y}_{\mathcal{D}}|\mathbf{X}_{\mathcal{D}})$. Lets consider the case when $\mathbf{x}_0 \in \mathtt{R}$. In this case we have that the $\mathcal{M}$ and $\mathtt{B}$ sets will be the same across $\mathcal{D}$ and $\tilde{\mathcal{D}}$. As a result we can proceed as

$$\begin{aligned}\int p(\mathbf{y}_{\tilde{\mathcal{D}}}|\mathbf{X}_{\tilde{\mathcal{D}}})dy_0 = \sum_{\mathbf{G},\mathbf{A}} \int & p_\theta(\mathbf{U}_{\mathtt{B}}|\mathbf{X}_{\mathtt{B}})p(\mathbf{G},\mathbf{A}|\mathbf{U}_{\mathtt{B}}) \\ & p_\theta(\mathbf{Z}_{\mathtt{B}}, \mathbf{y}_{\mathtt{B}}|\mathtt{R}, \mathbf{G}, \mathbf{A})d\mathbf{U}_{\mathtt{B}}d\mathbf{Z}_{\mathtt{B}}dy_{i\in\mathtt{R}\setminus\tilde{\mathcal{D}}_x}dy_0.\end{aligned} \tag{126}$$

Now we can notice that $y_{i\in\mathtt{R}\setminus\tilde{\mathcal{D}}_x} \cup y_0 = y_{i\in\mathtt{R}\setminus\mathcal{D}_x}$, hence the measure that we are integrating over above can be rewritten as

$$\begin{aligned}\int p(\mathbf{y}_{\tilde{\mathcal{D}}}|\mathbf{X}_{\tilde{\mathcal{D}}})dy_0 = \sum_{\mathbf{G},\mathbf{A}} \int & p_\theta(\mathbf{U}_{\mathtt{B}}|\mathbf{X}_{\mathtt{B}})p(\mathbf{G},\mathbf{A}|\mathbf{U}_{\mathtt{B}}) \\ & p_\theta(\mathbf{y}_{\mathtt{B}}, \mathbf{Z}_{\mathtt{B}}|\mathtt{R}, \mathbf{G}, \mathbf{A})d\mathbf{U}_{\mathtt{B}}d\mathbf{Z}_{\mathtt{B}}dy_{i\in\mathtt{R}\setminus\mathcal{D}_x},\end{aligned} \tag{127}$$

where it is easy to see that we arrived at the same expression as the one provided at Eq. 8. Now we will consider the case where $\mathbf{x}_0 \notin \mathtt{R}$. In this case we have that $\mathtt{R} \setminus \tilde{\mathcal{D}}_x = \mathtt{R} \setminus \mathcal{D}_x$ and thus

$$\begin{aligned}\int p(\mathbf{y}_{\tilde{\mathcal{D}}}|\mathbf{X}_{\tilde{\mathcal{D}}})dy_0 = \sum_{\mathbf{G},\mathbf{A},\mathbf{a}_0} \int & p_\theta(\mathbf{U}_{\mathtt{B}}|\mathbf{X}_{\mathtt{B}})p(\mathbf{G},\mathbf{A}|\mathbf{U}_{\mathtt{B}})p_\theta(\mathbf{y}_{\mathtt{B}}, \mathbf{Z}_{\mathtt{B}}|\mathtt{R}, \mathbf{G}, \mathbf{A}) \\ & p_\theta(\mathbf{u}_0|\mathbf{x}_0)p(\mathbf{a}_0|\mathbf{U}_{\mathtt{R}}, \mathbf{u}_0)p_\theta(\mathbf{z}_0|\mathrm{par}_{\mathbf{a}_0}(\mathtt{R}, \mathbf{y}_{\mathtt{R}})) \\ & p_\theta(y_0|\mathbf{z}_0)d\mathbf{U}_{\mathtt{B}}d\mathbf{Z}_{\mathtt{B}}d\mathbf{u}_0d\mathbf{z}_0dy_{i\in\mathtt{R}\setminus\mathcal{D}_x}dy_0.\end{aligned} \tag{128}$$

Notice that in this case the new point that is added is a leaf in the dependency graph, hence it doesn't affect any of the points in $\mathcal{D}$. As a result we can easily marginalize it out sequentially

$$\int p(\mathbf{y}_{\tilde{\mathcal{D}}}|\mathbf{X}_{\tilde{\mathcal{D}}})dy_0 = \sum_{\mathbf{G},\mathbf{A},\mathbf{a}_0} \int p_\theta(\mathbf{U}_B|\mathbf{X}_B)p(\mathbf{G},\mathbf{A}|\mathbf{U}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{G},\mathbf{A})$$
$$p_\theta(\mathbf{u}_0|\mathbf{x}_0)p(\mathbf{a}_0|\mathbf{U}_R,\mathbf{u}_0)p_\theta(\mathbf{z}_0|\mathrm{par}_{\mathbf{a}_0}(R,\mathbf{y}_R))$$
$$\cancelto{1}{\left(\int p_\theta(y_0|\mathbf{z}_0)dy_0\right)} d\mathbf{U}_B d\mathbf{Z}_B d\mathbf{u}_0 d\mathbf{z}_0 dy_{i\in R\setminus\mathcal{D}_x}. \tag{129}$$
$$= \sum_{\mathbf{G},\mathbf{A},\mathbf{a}_0} \int p_\theta(\mathbf{U}_B|\mathbf{X}_B)p(\mathbf{G},\mathbf{A}|\mathbf{U}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{G},\mathbf{A})$$
$$p_\theta(\mathbf{u}_0|\mathbf{x}_0)p(\mathbf{a}_0|\mathbf{U}_R,\mathbf{u}_0)$$
$$\cancelto{1}{\left(\int p_\theta(\mathbf{z}_0|\mathrm{par}_{\mathbf{a}_0}(R,\mathbf{y}_R))d\mathbf{z}_0\right)} d\mathbf{U}_B d\mathbf{Z}_B d\mathbf{u}_0 dy_{i\in R\setminus\mathcal{D}_x} \tag{130}$$
$$= \sum_{\mathbf{G},\mathbf{A}} \int p_\theta(\mathbf{U}_B|\mathbf{X}_B)p(\mathbf{G},\mathbf{A}|\mathbf{U}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{G},\mathbf{A})$$
$$p_\theta(\mathbf{u}_0|\mathbf{x}_0)\cancelto{1}{\left(\sum_{\mathbf{a}_0} p(\mathbf{a}_0|\mathbf{U}_R,\mathbf{u}_0)\right)} d\mathbf{U}_B d\mathbf{Z}_B d\mathbf{u}_0 dy_{i\in R\setminus\mathcal{D}_x} \tag{131}$$
$$= \sum_{\mathbf{G},\mathbf{A}} \int p_\theta(\mathbf{U}_B|\mathbf{X}_B)p(\mathbf{G},\mathbf{A}|\mathbf{U}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{G},\mathbf{A})$$
$$\cancelto{1}{\left(\int p(\mathbf{u}_0|\mathbf{x}_0)d\mathbf{u}_0\right)} d\mathbf{U}_B d\mathbf{Z}_B dy_{i\in R\setminus\mathcal{D}_x} \tag{132}$$
$$= \sum_{\mathbf{G},\mathbf{A}} \int p_\theta(\mathbf{U}_B|\mathbf{X}_B)p(\mathbf{G},\mathbf{A}|\mathbf{U}_B)p_\theta(\mathbf{y}_B,\mathbf{Z}_B|R,\mathbf{G},\mathbf{A})d\mathbf{U}_B d\mathbf{Z}_B dy_{i\in R\setminus\mathcal{D}_x} \tag{133}$$

where it is similarly easy to see that we arrived at Eq. 8. So we just showed that in both cases we have that $\int p(\mathbf{y}_{\tilde{\mathcal{D}}}|\mathbf{X}_{\tilde{\mathcal{D}}})dy_0 = p(\mathbf{y}_{\mathcal{D}}|\mathbf{X}_{\mathcal{D}})$, hence the model is consistent under marginalization. □

10.3.4 *Minibatch optimization of the bound of FNPs*

As we mentioned in the main text, the objective of FNPs is amenable to minibatching where the size of the batch scales according to the reference set R. We will only describe

the procedure for the FNP as the extension for FNP$^+$ is straightforward. Lets remind ourselves that the bound of FNPs can be expressed into two terms:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{q_\phi(\mathbf{Z}_R|\mathbf{X}_R)p_\theta(\mathbf{U}_R,\mathbf{G}|\mathbf{X}_R)}[\log p_\theta(\mathbf{y}_R,\mathbf{Z}_R|R,\mathbf{G}) - \log q_\phi(\mathbf{Z}_R|\mathbf{X}_R)] + \\ &+ \mathbb{E}_{p_\theta(\mathbf{U}_\mathcal{D},\mathbf{A}|\mathbf{X}_\mathcal{D})q_\phi(\mathbf{Z}_M|\mathbf{X}_M)}[\log p_\theta(\mathbf{y}_M|\mathbf{Z}_M) + \log p_\theta\left(\mathbf{Z}_M|\text{par}_\mathbf{A}(R,\mathbf{y}_R)\right) \\ &\qquad - \log q_\phi(\mathbf{Z}_M|\mathbf{X}_M)] \\ &= \mathcal{L}_R + \mathcal{L}_{M|R},\end{aligned} \tag{134}$$

where we have a term that corresponds to the variational bound on the datapoints in R, $\mathcal{L}_R$, and a second term that corresponds to the bound on the points in M when we condition on R, $\mathcal{L}_{M|R}$. While the $\mathcal{L}_R$ term of Eq. 134 cannot, in general, be decomposed to independent sums due to the DAG structure in R, the $\mathcal{L}_{M|R}$ term can; from the conditional i.i.d. nature of M and the structure of the variational posterior we can express it as M independent sums:

$$\mathcal{L}_{M|R} = \mathbb{E}_{p_\theta(\mathbf{U}_R|\mathbf{X}_R)}\left[\sum_{i=1}^{|M|}\mathbb{E}_{p_\theta(\mathbf{u}_i,\mathbf{A}_i|\mathbf{x}_i,\mathbf{U}_R)q_\phi(\mathbf{z}_i|\mathbf{x}_i)}\Big[\log p_\theta\left(\mathbf{y}_i,\mathbf{z}_i|\text{par}_{\mathbf{A}_i}(R,\mathbf{y}_R)\right) - \log q_\phi(\mathbf{z}_i|\mathbf{x}_i)\Big]\right]. \tag{135}$$

We can now easily use a minibatch $\hat{M}$ of points from M in order to approximate the inner sum and thus obtain unbiased estimates of the overall bound that depend on a minibatch $\{R,\hat{M}\}$:

$$\tilde{\mathcal{L}}_{M|R} = \mathbb{E}_{p_\theta(\mathbf{U}_R|\mathbf{X}_R)}\left[\frac{|M|}{|\hat{M}|}\sum_{i=1}^{|\hat{M}|}\mathbb{E}_{p_\theta(\mathbf{u}_i,\mathbf{A}_i|\mathbf{x}_i,\mathbf{U}_R)q_\phi(\mathbf{z}_i|\mathbf{x}_i)}\Big[\log p_\theta\left(\mathbf{y}_i,\mathbf{z}_i|\text{par}_{\mathbf{A}_i}(R,\mathbf{y}_R)\right) - \log q_\phi(\mathbf{z}_i|\mathbf{x}_i)\Big]\right], \tag{136}$$

thus obtain the following unbiased estimate of the overall bound that depends on a minibatch $\{S,\hat{M}\}$

$$\mathcal{L} \approx \mathcal{L}_R + \tilde{\mathcal{L}}_{M|R}. \tag{137}$$

In practice, this might limit us to use relatively small reference sets as training can become relatively expensive; in this case an alternative would be to subsample also the reference set and just reweigh appropriately $\mathcal{L}_R$. This provides a biased gradient estimator but, after a limited set of experiments, it seems that it can work reasonably well.

10.3.5 *Predictive distribution of FNPs*

Given the fact that the parameters of the model has been optimized, we are now seeking a way to do predictions for new unseen points. As we assumed that all of the reference points

are a part of the observed dataset $\mathcal{D}$, every new point $\mathbf{x}^*$ will be a part of O. Furthermore, we will have that $B = \mathcal{D}_X = \mathbf{X}_\mathcal{D}$. We will only provide the derivation for the FNP model, since the extension to FNP$^+$ is straightforward. To derive the predictive distribution for this point we will rely on Bayes theorem and thus have:

$$p_\theta(y^*|\mathbf{x}^*, \mathbf{X}_\mathcal{D}, \mathbf{y}_\mathcal{D}) = \frac{p_\theta(y^*, \mathbf{y}_\mathcal{D}|\mathbf{x}^*, \mathbf{X}_\mathcal{D})}{\int p_\theta(y^*, \mathbf{y}_\mathcal{D}|\mathbf{x}^*, \mathbf{X}_\mathcal{D}) dy^*}. \tag{138}$$

As we have established the consistency of FNP, we know that the denominator is $p_\theta(\mathbf{y}_\mathcal{D}|\mathbf{X}_\mathcal{D})$. Therefore we can expand the enumerator and rewrite Eq. 138 as

$$\begin{aligned} p_\theta(y^*|\mathbf{x}^*, \mathbf{X}_\mathcal{D}, \mathbf{y}_\mathcal{D}) = \sum_{\mathbf{G},\mathbf{A},\mathbf{a}^*} \int & \frac{p_\theta(\mathbf{U}_\mathcal{D}|\mathbf{X}_\mathcal{D}) p(\mathbf{G}, \mathbf{A}|\mathbf{U}_\mathcal{D}) p_\theta(\mathbf{Z}_\mathcal{D}, \mathbf{y}_\mathcal{D}|R, \mathbf{G}, \mathbf{A})}{p_\theta(\mathbf{y}_\mathcal{D}|\mathbf{X}_\mathcal{D})} \\ & p_\theta(\mathbf{u}^*|\mathbf{x}^*) p(\mathbf{a}^*|\mathbf{U}_R, \mathbf{u}^*) p_\theta(\mathbf{z}^*|\text{par}_{\mathbf{a}^*}(R, \mathbf{y}_R)) \\ & p_\theta(y^*|\mathbf{z}^*) d\mathbf{U}_\mathcal{D} d\mathbf{u}^* d\mathbf{Z}_\mathcal{D} d\mathbf{z}^*, \end{aligned} \tag{139}$$

where $\mathbf{a}^*$ is the binary vector that denotes which points from R are the parents of the new point. We can now see that the top part is the posterior distribution of the latent variables of the model when we condition on $\mathcal{D}$. We can thus replace it with its variational approximation

$$p_\theta(\mathbf{U}_\mathcal{D}|\mathbf{X}_\mathcal{D}) p(\mathbf{G}, \mathbf{A}|\mathbf{U}_\mathcal{D}) q_\phi(\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D})$$

and obtain

$$\begin{aligned} p_\theta(y^*|\mathbf{x}^*, \mathbf{X}_\mathcal{D}, \mathbf{y}_\mathcal{D}) &\approx \sum_{\mathbf{G},\mathbf{A},\mathbf{a}^*} \int p_\theta(\mathbf{U}_\mathcal{D}|\mathbf{X}_\mathcal{D}) p(\mathbf{G}, \mathbf{A}|\mathbf{U}_\mathcal{D}) q_\phi(\mathbf{Z}_\mathcal{D}|\mathbf{X}_\mathcal{D}) \\ &\quad p_\theta(\mathbf{u}^*|\mathbf{x}^*) p(\mathbf{a}^*|\mathbf{U}_R, \mathbf{u}^*) p_\theta(\mathbf{z}^*|\text{par}_{\mathbf{a}^*}(R, \mathbf{y}_R)) \\ &\quad p_\theta(y^*|\mathbf{z}^*) d\mathbf{U}_\mathcal{D} d\mathbf{u}^* d\mathbf{Z}_\mathcal{D} d\mathbf{z}^* \qquad (140) \\ &= \sum_{\mathbf{a}^*} \int p_\theta(\mathbf{U}_R, \mathbf{u}^*|\mathbf{X}_R, \mathbf{x}^*) p(\mathbf{a}^*|\mathbf{U}_R, \mathbf{u}^*) \\ &\quad p_\theta(\mathbf{z}^*|\text{par}_{\mathbf{a}^*}(R, \mathbf{y}_R)) \\ &\quad p_\theta(y^*|\mathbf{z}^*) d\mathbf{U}_R d\mathbf{u}^* d\mathbf{z}^* \qquad (141) \end{aligned}$$

after integrating / summing over the latent variables that do not affect the distributions that are specific to the new point.

10.4 APPENDIX OF CHAPTER 5

10.4.1 *Detailed experimental setup*

We implemented our methods in Tensorflow [1] and optimized the variational parameters using Adam [88] with the default hyperparameters. The means of the conditional Gaussian $q_\phi(\mathbf{W}|\mathbf{z})$ were initialized with the scheme proposed at [68], whereas the log of the standard deviations were initialized by sampling from $\mathcal{N}(-9, 1e-4)$. The parameters of $q_\phi(\mathbf{z})$

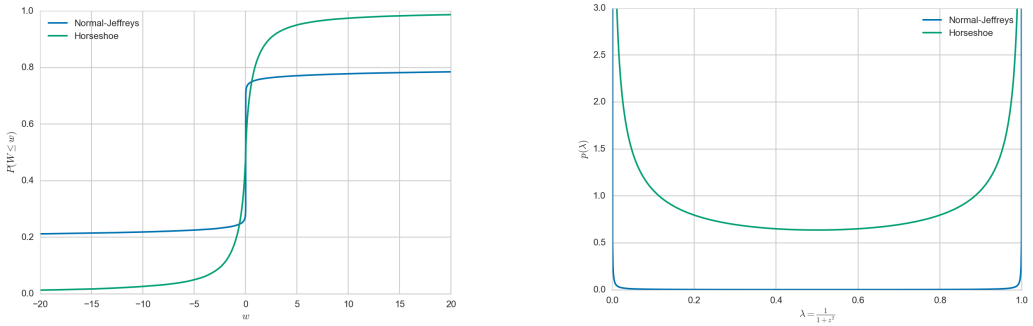
were initialized such that the overall mean of $\mathbf{z}$ is ≈ 1 and the overall variance is very low ($\approx 1e-8$); this ensures that all of the groups are active during the initial training iterations.

As for the standard deviation constraints; for the LeNet-300-100 architecture we constrained the standard deviation of the first layer to be $\leqslant 0.2$ whereas for the LeNet-5-Caffe we constrained the standard deviation of the first layer to be $\leqslant 0.5$. The remaining standard deviations were left unconstrained. For the VGG network we constrained the standard deviations of the 64 and 128 feature map layers to be $\leqslant 0.1$, the standard deviations of the 256 feature map layers to be $\leqslant 0.2$ and left the rest of the standard deviations unconstrained. We also found beneficial the incorporation of "warm-up" [170], i.e we annealed the negative KL-divergence between the approximate posterior and the prior with a linear schedule for the first 100 epochs. We initialized the means of the approximate posterior by the weights and biases obtained from a VGG network trained with batch normalization and dropout on CIFAR 10. For our method we disabled batch-normalization during training.

As for preprocessing the data; for MNIST the only preprocessing we did was to rescale the digits to lie at the $[-1, 1]$ range and for CIFAR 10 we used the preprocessed dataset provided by [202].

Furthermore, do note that by pruning a given filter at a particular convolutional layer we can also prune the parameters corresponding to that feature map for the next layer. This similarly holds for fully connected layers; if we drop a given input neuron then the weights corresponding to that node from the previous layer can also be pruned.

10.4.2 *Shrinkage properties of the normal-Jeffreys and horseshoe priors*



(a) Empirical CDF

(b) Prior on shrinkage coefficient

Figure 27: Comparison of the behavior of the log-uniform / normal-Jeffreys (NJ) prior and the horseshoe (HS) prior (where $s = 1$). Both priors behave similarly at zero but the normal-Jeffreys has an extremely heavy tail (thus making it non-normalizable).

In this section we will provide some insights about the behavior of each of the priors we employ by following the excellent analysis of [28]; we can perform a change of variables

and express the scale mixture distribution of eq.3 in the main paper in terms of a shrinkage coefficient, $\lambda = \frac{1}{1+z^2}$:

$$\lambda \sim p(\lambda); \qquad w \sim \mathcal{N}\left(0, \frac{1-\lambda}{\lambda}\right) \tag{142}$$

It is easy to observe that eq. 142 corresponds to a continuous relaxation of the spike-and-slab prior: when $\lambda = 0$ we have that $p(w|\lambda = 0) = \mathcal{U}(-\infty, \infty)$, i.e. no shrinkage/regularization for w, when $\lambda = 1$ we have that $p(w|\lambda = 1) = \delta(w = 0)$, i.e. w is exactly zero, and when $\lambda = \frac{1}{2}$ we have that $p(w|\lambda = \frac{1}{2}) = \mathcal{N}(0, 1)$. Now by examining the implied prior on the shrinkage coefficient λ for both the log-uniform and the horseshoe priors we can better study their behavior. As it is explained at [28], the half-Cauchy prior on z corresponds to a beta prior on the shrinkage coefficient, $p(\lambda) = \mathcal{B}(\frac{1}{2}, \frac{1}{2})$, whereas the normal-Jeffreys / log-uniform prior on z corresponds to $p(\lambda) = \mathcal{B}(\epsilon, \epsilon)$ with $\epsilon \approx 0$. The densities of both of these distributions can be seen at Figure 27b. As we can observe, the log-uniform prior posits a distribution that concentrates almost all of its mass at either $\lambda \approx 0$ or $\lambda \approx 1$, essentially either pruning the parameter or keeping it close to the maximum likelihood estimate due to $p(w|\lambda \approx 1) = \mathcal{U}(-\infty, \infty)$. In contrast the horseshoe prior maintains enough probability mass for the in-between values of λ and thus can, potentially, offer better regularization and generalization.

10.4.3 *Negative KL-divergences for log-normal approximating posteriors*

Let $q(z) = \mathcal{LN}(\mu, \sigma^2)$ be a log-normal approximating posterior. Here we will derive the negative KL-divergences between $q(z)$ and inverse gamma, gamma and half-normal distributions.

Let $p(z)$ be an inverse gamma distribution, i.e. $p(z) = \mathcal{IG}(\alpha, \beta)$. The negative KL-divergence can be expressed as follows:

$$-\mathrm{KL}(q(z)||p(z)) = \int q(z) \log p(z) dz - \int q(z) \log q(z) dz \tag{143}$$

The second term is the entropy of the log-normal distribution which has the following form:

$$\mathcal{H}_q = -\int q(z) \log q(z) dz = \frac{1}{2} \log \sigma^2 + \mu + \frac{1}{2} + \frac{1}{2} \log(2\pi) \tag{144}$$

The first term is the cross entropy between the log-normal posterior and the inverse-Gamma prior:

$$\mathcal{CE}_{qp} = \int q(z) \left(\alpha \log \beta - \log \Gamma(\alpha) - (\alpha + 1) \log z) - \frac{\beta}{z} \right) dz \tag{145}$$

$$= \alpha \log \beta - \log \Gamma(\alpha) - (\alpha + 1) \mathbb{E}_{q(z)}[\log z] - \beta \mathbb{E}_{q(z)}[z^{-1}] \tag{146}$$

Since the natural logarithm of a log-normal distribution $\mathcal{LN}(\mu, \sigma^2)$ follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$ we have that $\mathbb{E}_{q(z)}[\log z] = \mu$. Furthermore we have that if $x \sim \mathcal{LN}(\mu, \sigma^2)$

then $\frac{1}{x} \sim \mathcal{LN}(-\mu, \sigma^2)$, therefore $\mathbb{E}_{q(z)}[z^{-1}] = \exp(-\mu + \frac{\sigma^2}{2})$. Putting everything together we have that:

$$\mathcal{CE}_{qp} = \alpha \log \beta - \log \Gamma(\alpha) - (\alpha + 1)\mu - \beta \exp(-\mu + \frac{\sigma^2}{2}) \tag{147}$$

Therefore the negative KL-divergence is:

$$\begin{aligned} -\mathrm{KL}(q(z)||p(z)) = \alpha \log \beta - \log \Gamma(\alpha) - \alpha\mu - \beta \exp(-\mu + 0.5\sigma^2) + \\ + 0.5(\log \sigma^2 + 1 + \log(2\pi)) \end{aligned} \tag{148}$$

Now let $p(z)$ be a Gamma prior, i.e. $p(z) = \mathcal{G}(\alpha, \beta)$. We have that the cross-entropy changes to:

$$\mathcal{CE}_{qp} = \int q(z) \Big(- \alpha \log \beta - \log \Gamma(\alpha) - \frac{z}{\beta} + (\alpha - 1) \log z \Big) dz \tag{149}$$

$$= -\alpha \log \beta - \log \Gamma(\alpha) - \beta^{-1} \mathbb{E}_{q(z)}[z] + (\alpha - 1)\mathbb{E}_{q(z)}[\log z] \tag{150}$$

$$= -\alpha \log \beta - \log \Gamma(\alpha) - \beta^{-1} \exp(\mu + \frac{\sigma^2}{2}) + (\alpha - 1)\mu \tag{151}$$

Therefore the negative KL-divergence is:

$$\begin{aligned} -\mathrm{KL}(q(z)||p(z)) = -\alpha \log \beta - \log \Gamma(\alpha) + \alpha\mu - \beta^{-1} \exp(\mu + 0.5\sigma^2) + \\ + 0.5(\log \sigma^2 + 1 + \log(2\pi)) \end{aligned} \tag{152}$$

Now, with the above we can express the KL-divergence between the prior $p(s_a, s_b, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}})$ and the variational posterior $q_\phi(s_a, s_b, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}})$ as follows:

$$\begin{aligned} -\mathrm{KL}(q_\phi(s_a)||p(s_a)) = \log \tau_0 + \tau_0^{-1} \exp\big(\mu_{s_a} + \frac{1}{2}\sigma^2_{s_a}\big) + \\ \frac{1}{2}\big(\mu_{s_a} + \log \sigma^2_{s_a} + 1 + \log 2\big) \end{aligned} \tag{153}$$

$$\begin{aligned} -\mathrm{KL}(q_\phi(s_b)||p(s_b)) = -\exp\big(\frac{1}{2}\sigma^2_{s_b} - \mu_{s_b}\big) + \\ \frac{1}{2}\big(- \mu_{s_b} + \log \sigma^2_{s_b} + 1 + \log 2\big) \end{aligned} \tag{154}$$

$$\begin{aligned} -\mathrm{KL}(q_\phi(\tilde{\boldsymbol{\alpha}})||p(\tilde{\boldsymbol{\alpha}})) = \sum_i^A \Bigg(\exp\big(\mu_{\tilde{\alpha}_i} + \frac{1}{2}\sigma^2_{\tilde{\alpha}_i}\big) + \\ \frac{1}{2}\big(\mu_{\tilde{\alpha}_i} + \log \sigma^2_{\tilde{\alpha}_i} + 1 + \log 2\big)\Bigg) \end{aligned} \tag{155}$$

$$\begin{aligned} -\mathrm{KL}(q_\phi(\tilde{\boldsymbol{\beta}})||p(\tilde{\boldsymbol{\beta}})) = \sum_i^A \Bigg(-\exp\big(\frac{1}{2}\sigma^2_{\tilde{\beta}_i} - \mu_{\tilde{\beta}_i}\big) + \\ \frac{1}{2}\big(- \mu_{\tilde{\beta}_i} + \log \sigma^2_{\tilde{\beta}_i} + 1 + \log 2\big)\Bigg) \end{aligned} \tag{156}$$

with the KL-divergence for the weight distribution $q_\phi(\tilde{\mathbf{W}})$ given by eq.8 in the main paper.

10.4.4 *Visualizations*

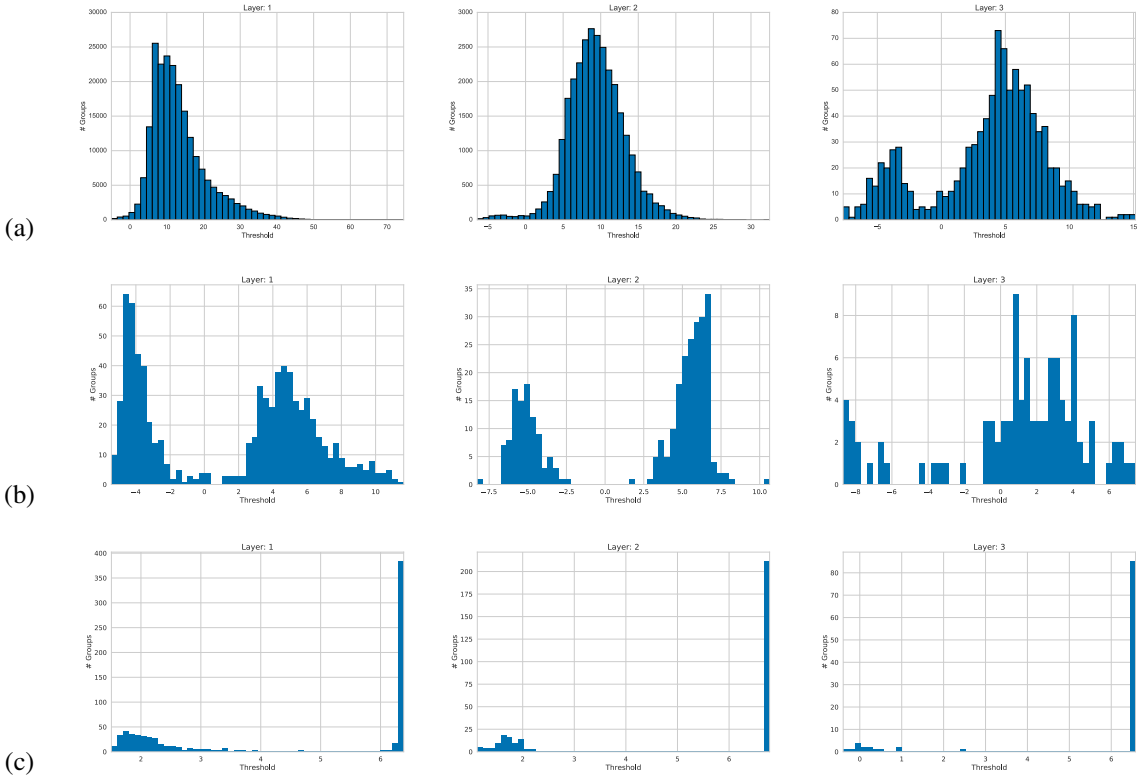


Figure 28: Distribution of the thresholds for the Sparse Variational Dropout 28a, Bayesian Compression with group normal-Jeffreys (BC-GNJ) 28b and group Horseshoe (BC-GHS) 28c priors for the three layer LeNet-300-100 architecture. It is easily observed that there are usually two well separable groups with BC-GNJ and BC-GHS, thus making the choice for the threshold easy. Smaller values indicate signal whereas larger values indicate noise (i.e. useless groups).

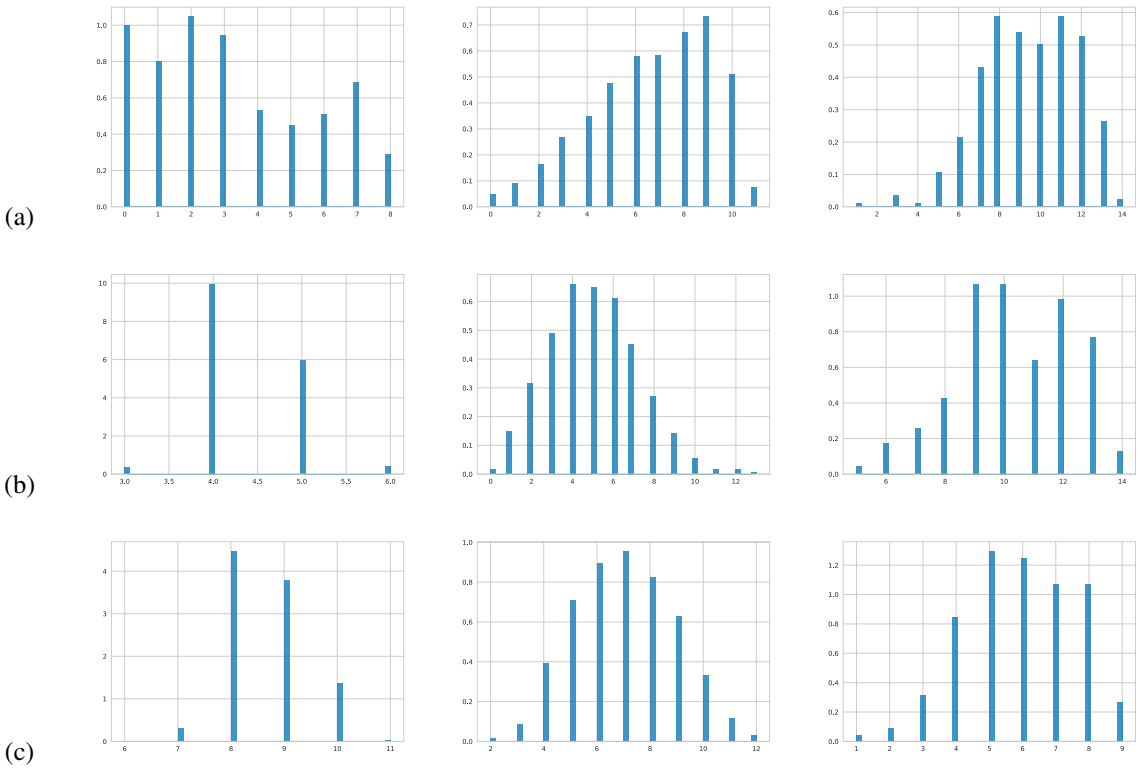


Figure 29: Distribution of the bit precisions for the Sparse Variational Dropout 29a, Bayesian Compression with group normal-Jeffreys (BC-GNJ) 29b and group Horseshoe (BC-GHS) 29c priors for the three layer LeNet-300-100 architecture. All of the methods usually require far fewer than 32bits for the weights.

10.4.5 *Algorithms for the feedforward pass*

Algorithms 5, 6, 7, 8 describe the forward pass using local reparametrizations for fully connected and convolutional layers with the approximate posteriors for the Bayesian Compression (BC) with group normal-Jeffreys (BC-GNJ) and group Horseshoe (BC-GHS) priors employed at the experiments. For the fully connected layers we coupled the scales for each input neuron whereas for the convolutional we couple the scales for each output feature map. $\mathbf{M}_W, \mathbf{\Sigma}_W$ are the means and variances of each layer, $\mathbf{H}$ is a minibatch of activations of size K. For the first layer we have that $\mathbf{H} = \mathbf{X}$ where $\mathbf{X}$ is the minibatch of inputs. For the convolutional layers N_f are the number of convolutional filters, $*$ is the convolution operator and we assume the [batch, height, width, feature maps] convention. rsh corresponds to a reshape operation.

Algorithm 5 Fully connected layer with BC-GNJ.

Require: $\mathbf{H}, \mathbf{M}_w, \mathbf{\Sigma}_w$

$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$
$\mathbf{Z} = \boldsymbol{\mu}_z + \boldsymbol{\sigma}_z \odot \hat{\mathbf{E}}$
$\hat{\mathbf{H}} = \mathbf{H} \odot \mathbf{Z}$
$\mathbf{M}_h = \hat{\mathbf{H}}\mathbf{M}_w$
$\mathbf{V}_h = \hat{\mathbf{H}}^2\mathbf{\Sigma}_w$
$\mathbf{E} \sim \mathcal{N}(0, 1)$
return $\mathbf{M}_h + \sqrt{\mathbf{V}_h} \odot \mathbf{E}$

Algorithm 6 Convolutional BC-GNJ layer.

Require: $\mathbf{H}, \mathbf{M}_w, \mathbf{\Sigma}_w$

$\mathbf{M}_h = \mathbf{H} * \mathbf{M}_w$
$\mathbf{V}_h = \mathbf{H}^2 * \mathbf{\Sigma}_w$
$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$
$\hat{\boldsymbol{\mu}}_z = \text{rsh}(\boldsymbol{\mu}_z, [K, 1, 1, N_f])$
$\hat{\boldsymbol{\sigma}}_z = \text{rsh}(\boldsymbol{\sigma}_z, [K, 1, 1, N_f])$
$\mathbf{Z} = \hat{\boldsymbol{\mu}}_z + \hat{\boldsymbol{\sigma}}_z \odot \hat{\mathbf{E}}$
$\mathbf{E} \sim \mathcal{N}(0, 1)$
return $\mathbf{M}_h \odot \mathbf{Z} + \sqrt{\mathbf{V}_h \odot \mathbf{Z}^2} \odot \mathbf{E}$

Algorithm 7 Fully connected layer with BC-GHS.

Require: $\mathbf{H}, \mathbf{M}_w, \mathbf{\Sigma}_w$

$\hat{\mathbf{e}} \sim \mathcal{N}(0, 1)$
$\mu_s = .5\mu_{s_a} + .5\mu_{s_b}$
$\sigma_s = \sqrt{.25\sigma_{s_a}^2 + .25\sigma_{s_b}^2}$
$\log \mathbf{s} = \mu_s + \sigma_s \odot \hat{\mathbf{e}}$
$\boldsymbol{\mu}_{\tilde{z}} = .5\boldsymbol{\mu}_{\tilde{\alpha}} + .5\boldsymbol{\mu}_{\tilde{\beta}} + \log \mathbf{s}$
$\boldsymbol{\sigma}_{\tilde{z}} = \sqrt{.25\boldsymbol{\sigma}_{\tilde{\alpha}}^2 + .25\boldsymbol{\sigma}_{\tilde{\beta}}^2}$
$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$
$\mathbf{Z} = \exp(\boldsymbol{\mu}_{\tilde{z}} + \boldsymbol{\sigma}_{\tilde{z}} \odot \hat{\mathbf{E}})$
$\hat{\mathbf{H}} = \mathbf{H} \odot \mathbf{Z}$
$\mathbf{M}_h = \hat{\mathbf{H}}\mathbf{M}_w$
$\mathbf{V}_h = \hat{\mathbf{H}}^2\mathbf{\Sigma}_w$
$\mathbf{E} \sim \mathcal{N}(0, 1)$
return $\mathbf{M}_h + \sqrt{\mathbf{V}_h} \odot \mathbf{E}$

Algorithm 8 Convolutional BC-GHS layer.

Require: $\mathbf{H}, \mathbf{M}_w, \mathbf{\Sigma}_w$

$\mathbf{M}_h = \mathbf{H} * \mathbf{M}_w$
$\mathbf{V}_h = \mathbf{H}^2 * \mathbf{\Sigma}_w$
$\hat{\mathbf{e}} \sim \mathcal{N}(0, 1)$
$\mu_s = .5\mu_{s_a} + .5\mu_{s_b}$
$\sigma_s = \sqrt{.25\sigma_{s_a}^2 + .25\sigma_{s_b}^2}$
$\log \mathbf{s} = \text{rsh}(\mu_s + \sigma_s \odot \hat{\mathbf{e}}, [K, 1, 1, 1])$
$\boldsymbol{\mu}_{\tilde{z}} = \text{rsh}(.5\boldsymbol{\mu}_{\tilde{\alpha}} + .5\boldsymbol{\mu}_{\tilde{\beta}}, [K, 1, 1, N_f])$
$\boldsymbol{\sigma}_{\tilde{z}} = \text{rsh}(\sqrt{.25(\boldsymbol{\sigma}_{\tilde{\alpha}}^2 + \boldsymbol{\sigma}_{\tilde{\beta}}^2)}, [K, 1, 1, N_f])$
$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$
$\mathbf{Z} = \exp(\boldsymbol{\mu}_{\tilde{z}} + \log \mathbf{s} + \boldsymbol{\sigma}_{\tilde{z}} \odot \hat{\mathbf{E}})$
$\mathbf{E} \sim \mathcal{N}(0, 1)$
return $\mathbf{M}_h \odot \mathbf{Z} + \sqrt{\mathbf{V}_h \odot \mathbf{Z}^2} \odot \mathbf{E}$

10.5 APPENDIX OF CHAPTER 6

10.5.1 *Relation to variational inference*

The objective function described in Eq. 97 is in fact a special case of a variational lower bound over the parameters of the network under a spike and slab [127] prior. The spike and slab distribution is the golden standard in sparsity as far as Bayesian inference is concerned and it is defined as a mixture of a delta spike at zero and a continuous distribution over the real line (e.g. a standard normal):

$$p(z) = \text{Bernoulli}(\pi), \qquad p(\theta|z = 0) = \delta(\theta), \qquad p(\theta|z = 1) = \mathcal{N}(\theta|0, 1). \tag{157}$$

Since the true posterior distribution over the parameters under this prior is intractable, we will use variational inference [16]. Let $q(\theta, z)$ be a spike and slab approximate posterior over the parameters θ and gate variables z, where we assume that it factorizes over the dimensionality of the parameters θ. It turns out that we can write the following variational free energy under the spike and slab prior and approximate posterior over a parameter vector θ:

$$\begin{aligned}\mathcal{F} = &-\mathbb{E}_{q(\mathbf{z})q(\theta|\mathbf{z})}[\log p(\mathcal{D}|\theta)] + \sum_{j=1}^{|\theta|} KL(q(z_j)||p(z_j)) + \\ &+ \sum_{j=1}^{|\theta|} \big(q(z_j = 1)KL(q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1)) + \\ &+ q(z_j = 0)KL(q(\theta_j|z_j = 0)||p(\theta_j|z_j = 0))\big) \qquad (158) \\ = &-\mathbb{E}_{q(\mathbf{z})q(\theta|\mathbf{z})}[\log p(\mathcal{D}|\theta)] + \sum_{j=1}^{|\theta|} KL(q(z_j)||p(z_j)) + \\ &+ \sum_{j=1}^{|\theta|} q(z_j = 1)KL(q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1)), \qquad (159)\end{aligned}$$

where the last step is due to $KL(q(\theta_j|z_j = 0)||p(\theta_j|z_j = 0)) = 0$[1]. The term that involves $KL(q(z_j)||p(z_j))$ corresponds to the KL-divergence from the Bernoulli prior $p(z_j)$ to the Bernoulli approximate posterior $q(z_j)$ and $KL(q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1))$ can be interpreted as the "code cost" or else the amount of information the parameter θ_j contains about the data $\mathcal{D}$, measured by the KL-divergence from the prior $p(\theta_j|z_j = 1)$.

Now consider making the assumption that we are optimizing, rather than integrating, over θ and further assuming that $KL(q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1)) = \lambda$. We can justify this assumption from an empirical Bayesian procedure: there is a hypothetical prior for each parameter $p(\theta_j|z_j = 1)$ that adapts to $q(\theta_j|z_j = 1)$ in a way that results into needing, approximately, λ nats to transform $p(\theta_j|z_j = 1)$ to that particular $q(\theta_j|z_j = 1)$. Those λ nats are thus the amount of information the $q(\theta_j|z_j = 1)$ can encode about the data had we used that $p(\theta_j|z_j = 1)$ as the prior. Notice that under this view we can consider λ as the amount of flexibility of that hypothetical prior; with $\lambda = 0$ we have a prior that is flexible enough to represent exactly $q(\theta_j|z_j = 1)$, thus resulting into no code cost and possible overfitting. Under this assumption the variational free energy can be re-written as:

$$\mathcal{F} = -\mathbb{E}_{q(\mathbf{z})}[\log p(\mathcal{D}|\tilde{\theta} \odot \mathbf{z})] + \sum_{j=1}^{|\theta|} KL(q(z_j)||p(z_j)) + \lambda \sum_{j=1}^{|\theta|} q(z_j = 1) \qquad (160)$$

$$\geqslant -\mathbb{E}_{q(\mathbf{z})}[\log p(\mathcal{D}|\tilde{\theta} \odot \mathbf{z})] + \lambda \sum_{j=1}^{|\theta|} \pi_j, \qquad (161)$$

1 We can see that this is indeed the case by taking the limit of $\sigma \to 0$ of the KL divergence of two Gaussians that have the same mean and variance.

where $\tilde{\theta}$ corresponds to the optimized θ and the last step is due to the positivity of the KL-divergence. Now by taking the negative log-probability of the data to be equal to the loss $\mathcal{L}(\cdot)$ of Eq. 95 we see that Eq. 161 is the same as Eq. 97. Note that in case that we are interested over the uncertainty of the gates $\mathbf{z}$, we should optimize Eq. 160, rather than Eq. 161, as this will properly penalize the entropy of $q(\mathbf{z})$. Furthermore, Eq. 160 also allows for the incorporation of prior information about the behavior of the gates (e.g. gates being active 10% of the time, on average). We have thus shown that the expected L_0 minimization procedure is in fact a close surrogate to a variational bound involving a spike and slab distribution over the parameters and a fixed coding cost for the parameters when the gates are active.

10.5.2 *The hard concrete distribution*

As mentioned in the main text, the hard concrete is a straightforward modification of the binary concrete [85, 120]; let $q_s(s|\phi)$ be the probability density function (pdf) and $Q_s(s|\phi)$ the cumulative distribution function (CDF) of a binary concrete random variable s:

$$q_s(s|\phi) = \frac{\beta\alpha s^{-\beta-1}(1-s)^{-\beta-1}}{(\alpha s^{-\beta}+(1-s)^{-\beta})^2}, \tag{162}$$

$$Q_s(s|\phi) = \text{Sigmoid}((\log s - \log(1-s))\beta - \log\alpha). \tag{163}$$

Now by stretching this distribution to the (γ, ζ) interval, with $\gamma < 0$ and $\zeta > 1$ we obtain $\bar{s} = s(\zeta-\gamma)+\gamma$ with the following pdf and CDF:

$$q_{\bar{s}}(\bar{s}|\phi) = \frac{1}{|\zeta-\gamma|} q_s\left(\frac{\bar{s}-\gamma}{\zeta-\gamma}\middle|\phi\right), \qquad Q_{\bar{s}}(\bar{s}|\phi) = Q_s\left(\frac{\bar{s}-\gamma}{\zeta-\gamma}\middle|\phi\right). \tag{164}$$

and by further rectifying $\bar{s}$ with the hard-sigmoid, $z = \min(1, \max(0, \bar{s}))$, we obtain the following distribution over z:

$$\begin{aligned} q(z|\phi) = {} & Q_{\bar{s}}(0|\phi)\delta(z) + (1 - Q_{\bar{s}}(1|\phi))\delta(z-1) + \\ & (Q_{\bar{s}}(1|\phi) - Q_{\bar{s}}(0|\phi))q_{\bar{s}}(z|\bar{s} \in (0,1), \phi), \end{aligned} \tag{165}$$

which is composed by a delta peak at zero with probability $Q_{\bar{s}}(0|\phi)$, a delta peak at one with probability $1 - Q_{\bar{s}}(1|\phi)$, and a truncated version of $q_{\bar{s}}(\bar{s}|\phi)$ in the (0, 1) range.

10.5.3 *Negative KL-divergence for hard concrete distributions*

In case th 160 is to be optimized with a hard concrete $q(z)$ then we have to compute the KL-divergence from a prior $p(z)$ to $q(z)$. It is necessary for the prior $p(z)$ to have the same support as $q(z)$ in order for the KL-divergence to be valid; as a result we can let the prior $p(z)$ similarly be a hard-sigmoid transformation of an arbitrary continuous distribution $p(\bar{s})$ with CDF $P_{\bar{s}}(\bar{s})$:

$$p(z) = P_{\bar{s}}(0)\delta(z) + (1 - P_{\bar{s}}(1))\delta(z-1) + (P_{\bar{s}}(1) - P_{\bar{s}}(0))p_{\bar{s}}(z|\bar{s} \in (0,1)) \tag{166}$$

Since both $q(z)$ and $p(z)$ are mixtures with the same number of components we can use the chain rule of relative entropy [38, 74] in order to compute the KL-divergence:

$$\mathrm{KL}(q(z)||p(z)) = Q_{\bar{s}}(0)\log\frac{Q_{\bar{s}}(0)}{P_{\bar{s}}(0)} + (1-Q_{\bar{s}}(1))\log\frac{1-Q_{\bar{s}}(1)}{1-P_{\bar{s}}(1)} + \\ + (Q_{\bar{s}}(1)-Q_{\bar{s}}(0))\mathbb{E}_{q_{\bar{s}}(z|\bar{s}\in(0,1))}\left[\log\frac{q_{\bar{s}}(z)}{p_{\bar{s}}(z)}\right], \tag{167}$$

where $\bar{s}$ corresponds to the the pre-rectified variable. Notice that in case that the integral under the truncated distribution $q(\bar{s}|\bar{s}\in(0,1))$ is not available in closed form we can still obtain a Monte Carlo estimate by sampling the truncated distribution, on e.g. a (γ,ζ) interval, via the inverse transform method:

$$u\sim\mathcal{U}(0,1), \qquad z = Q_{\bar{s}}^{-1}\big(Q_{\bar{s}}(\gamma)+u(Q_{\bar{s}}(\zeta)-Q_{\bar{s}}(\gamma))\big), \tag{168}$$

where $Q_{\bar{s}}^{-1}(\cdot)$ corresponds to the quantile function and $Q_{\bar{s}}(\cdot)$ to the CDF of the random variable $\bar{s}$. Furthermore, it should be mentioned that, since the rectifications are not invertible transformations, $\mathrm{KL}(q(z)||p(z)) \neq \mathrm{KL}(q(\bar{s})||p(\bar{s}))$.

10.6 APPENDIX OF CHAPTER 7

10.6.1 *Experimental details*

The grid width α of each grid was initialized according to the bit-width b and the maximum and minimum values of the input x to the quantizer[2]. Since the inputs $\tilde{x}$ in both cases for our approach are stochastic it makes sense to assume a width for the grid that is slightly larger than the standard width $t = (\max(x)-\min(x))/2^b$; for the activations, whenever $b > 4$, we initialize $\alpha = t + 3t/2^b$, for $4 \geqslant b > 2$ we used $\alpha = t + 3t/2^{b+1}$ and finally for $b = 2$ we used $\alpha = t$. Since with ReLU activations the magnitude can become quite large (thus leading to increased quantization noise for smaller bit widths), this scheme keeps the noise injected to the network in check. For the weights we always used an initial $\alpha = t + 3t/2^b$. The standard deviation of the logistic noise σ was initialized to be three times smaller than the width α, i.e. $\sigma = \alpha/3$. Under this specification, most of the probability mass of the logistic distribution is initially (roughly) in the bins containing the closest grid point and its' two neighbors.

The moving averages of layer statistics that are aggregated during the training phase for the batch normalization do not necessarily reflect the statistics of the quantized model accurately. Even though RQ aims to minimize the gap between training and testing phase, we found that the aggregated statistics in combination with the learned scale and shift parameters of batch normalization lead to decreased test performance. In order to avoid this drop in accuracy, we apply the insights from [144] and recompute the statistics of the quantized model before reporting the final test error rate. The final models were determined through early stopping using the validation loss computed with minibatch statistics, in case the model uses batch normalization.

2 For activations we computed the minimum and maximum on a random minibatch of inputs.

For the MNIST experiment we rescaled the input to the [-1, 1] range, employed no regularization and the network was trained with Adam [88] and a batch size of 128. We used a local grid whenever the bit width was larger than 2 for both, weights and biases (shared grid parameters), as well as for the ouputs of the ReLU, with $\delta = 3$. For the 8 and 4 bit networks we used a temperature λ of 2 whereas for the 2 bit models we used a temperature of 1 for RQ. We trained the 8 and 4 bit networks for 100 epochs using a learning rate of 1e-3 and the 2 bit networks for 200 epochs with a learning rate of 5e-4. In all of the cases the learning rate was annealed to zero during the last 50 epochs.

For the CIFAR 10 experiment, the hyperparameters were chosen identically to the LeNet-5 experiments except a few differences. We chose a learning rate ot 1e-4 instead of 1e-3 for 8 and 4 bit networks and trained for 300 epochs with a batch size of 100. We also included a weight decay term of 1e-4 for the 8 bit networks. For the 2 bit model we started with a learning rate of 1e-3. The VGG model contains a batch normalization layer after every convolutional layer, but preceeded by max pooling, if present.

10.6.2 *Imagenet details*

Each channel of the input images was preprocessed by subtracting the mean and dividing by the standard deviation of that channel across the training set. We then resized the images such that the shorter side is set to 256 and then applied random 224x224 crops and random horizontal flips for data augmentation. For evaluation we consider the center 224x224 crop of the images.

We trained the base Resnet-18 model with stochastic gradient descent, a batch size of 128, nesterov momentum of 0.9 and a learning rate of 0.1 which was multiplied by 0.1 at the 30th and 60th epoch. We also applied weight decay with a strength of 1e-4. For the quantized model fine-tuning phase, we used Adam with a learning rate of $5e^{-6}$, a batch size of 24 and a momentum of 0.99. We used a temperature of 2 for both RQ variants. Following the strategy in [83], we did not quantize the biases.

Table 16 contains the error rates for Resnet-18 on which Figure 22 is based on. Algorithm and architecture specific changes are mentioned explicitly through footnotes.

10.6.2.1 *[83] for Resnet18*

We used the code provided at `https://github.com/tensorflow/models/tree/master/official/resnet` and modified the construction of the computation graph by inserting quantization operations provided at `tensorflow.contrib.quantize`. In a first step, the unmodified code was used to train a high-precision Resnet18 model using the hyper-parameter settings for the learning rate scheduling that are provided in the github repository. More specifically, the model was trained for 90 epochs with a batch size of 128. The learning rate scheduling involved a "warm up" period in which the learning rate was annealed from zero to 0.64 over the first 50k steps, after which it was divided by 10 after epochs 30, 60 and 80 respectively. Gradients were modified using a momentum of 0.9. Final test performance under this procedure is 29.53% top-1 error and 10.44% top-5 error. From the high-precision model checkpoint, the final quantized model was then fine-

tuned for 10 epochs using a constant learning rate of $1e^{-4}$ and momentum of 0.9. We did not freeze the moving averages of the batch normalization layers. Finally, we found that re-estimating the batchnorm statistics was harmful for this algorithm. We hypothesise that this is due to the usage of folded batch normalization, which incorporates the statistics into the construction of the grid at training time.

Table 16: Top-1 and top-5 error (%) with Resnet18 on Imagenet

		Resnet18	
Method	# Bits weights/act.	Top-1	Top-5
Original	32/32	30.46	10.81
SR+DR	8/8	31.83	11.48
[61, 63]	6/6	40.75	16.90
	5/5	45.48	20.16
Rounding	8/8	30.22	10.60
	6/6	31.61	11.32
	5/5	36.97	14.95
	4/4	78.79	57.10
[83][a]	8/8	29.62	10.45
	6/6	32.69	12.46
	5/5	35.36	13.33
LR Net [161]	1/32[b]	40.10	17.70
	2/32[c]	36.50	15.20
TWN [103]	2/32	38.20	15.80
INQ [205]	5/32	31.02	10.90
BWN [149]	1/32	39.20	17.00
XNOR-net [149]	1/1	48.80	26.80
HWGQ [27][b]	1/2	40.4	17.8
ELQ [206]	1/32	35.28	13.96
	2/32	32.48	11.95
SYQ [47][d]	1/8	37.1	15.4
	2/8	32.3	12.2
Apprentice [125][b]	2/8	32	-
	4/8	29.6	-
RQ (ours)	8/8	30.03	10.56
	6/6	31.35	11.22
	5/5	34.90	13.43
	4/4	38.48	16.01
RQ ST (ours)	8/8	30.37	10.67
	6/6	31.85	11.62
	5/5	36.65	14.54
	4/4	37.54	15.22

a Includes folded batch normalization
b First and last layer not quantized
c First layer not quantized
d Weights of first and last layer not quantized

STATISTISCH REDENEREN OVER ONZEKERHEID EN COMPRESSIE IN DIEP LEREN: SAMENVATTING

Er bestaat een groot aantal verschillende taken waarbij de voorspellende capaciteit van neurale netwerken en diep leren excelleren. Dit heeft er toe geleid dat deze technologie wordt gebruikt voor verscheidene toepassingen die een belangrijke rol spelen in het dagelijks leven. Mede hierom is onderzoek naar potentiële verbeteringen van deze technieken een belangrijk onderwerp. In dit proefschrift werken we aan het verbeteren van twee belangrijke aspecten van neuraal netwerk modellen; het vermogen om naast voorspellingen ook de onzekerheden van die voorspellingen te leren, en de inherente noodzaak van grote hoeveelheden rekenkracht en andere computationele middelen.

Dit werk begint met een introductie waarin de twee hoofd onderzoeksvragen van dit proefschrift worden gesteld. Hierbij wordt ook de benodigde achtergrondkennis behandeld die zal worden gebruikt in de rest van het proefschrift. We beschrijven neurale netwerken en Bayesiaanse neurale netwerken. Dit laatste soort neurale netwerken heeft parameters (aka de weights en biases) die geen vaste waarde hebben maar stochastisch zijn en worden beschreven door een kansverdeling. Dit wordt gecombineerd met Bayesiaanse inferentie, een manier om de kansverdeling over de parameters aan te passen aan de hand van geobserveerde data. Tenslotte geven we ook een korte introductie over de compressie van neurale netwerken door middel van pruning en quantisatie. Bij pruning worden de niet relevante parameters en delen van het netwerk verwijderd door ze op nul te zetten. Bij quantisatie worden de numerieke waardes van de weights en tussenliggende representaties van het netwerk gerepresenteerd in een hardware-vriendelijk formaat (bijv. fixed point).

Het eerste gedeelte van dit proefschrift beschrijft drie contributies die het vermogen van neurale netwerken verbeteren om onzekerheden in voorspellingen in te schatten. De eerste twee contributies gaan over het verbeteren van de kwaliteit van voorspellings onzekerheden van variationele Bayesiaanse neurale netwerken door middel van betere benaderingen van de kansverdeling van de parameters bij het observeren van nieuwe data. We stellen een simpele manier voor om lineaire afhankelijkheid tussen de weights van een neuraal netwerk te introduceren door middel van matrix variate Gaussian distributies; dit zijn distributies over stochastische matrices die eenvoudig de correlaties kunnen modelleren tussen input en output neuronen in elke laag. We zullen aantonen dat dit leidt tot verbeterde prestaties. Vervolgens stellen we multiplicative normalizing flows voor, een algemeen kader waarbij niet-lineaire afhankelijkheden worden geïntroduceerd tussen de parameters van het netwerk. Dit wordt mogelijk gemaakt door middel van een combinatie van auxiliaire stochastische variabelen en geparametriseerde bijecties, op een manier die flexibele correlaties tussen de weights van elke laag mogelijk maakt zonder dat het te computationeel intensief wordt. Door middel van experimenten laten we zien dat de kwaliteit van de onzekerheden van de voorspellingen hierdoor is verbeterd ten opzichte van meer eenvoudige Gaussische benaderingen uit eerder werk. De laatste contributie van dit gedeelte

betreft functional neural processes, een model dat een ander standpunt aanneemt; in tegenstelling tot het maken van een aanname over de kansverdelingen en het uitvoeren van (variationele) inferentie over de weights van het neurale netwerk, wordt het raamwerk van stochastische processen gebruikt. Hierbij wordt een kansverdeling aangenomen en inferentie gedaan over de ruimte van functies die het neurale netwerk kan representeren. Hierbij is een bijkomend voordeel dat inferentie eenvoudiger is en de modelleer taak intuïtiever, doordat dit raamwerk het mogelijk maakt om te redeneren over de relaties tussen verschillende datapunten in een dataset door het introduceren van een referentieset van datapunten. Dit is in tegenstelling tot de moeilijk te interpreteren parameters van een neuraal netwerk. Aan de hand van experimenten laten we zien dat dit soort modellen betere onzekerheden verstrekken terwijl de voorspellingen van vergelijkbare kwaliteit blijven.

Het tweede gedeelte van dit proefschrift beschrijft drie nieuwe technieken voor compressie die de mogelijkheid bieden om neurale netwerken te leren die zowel kleiner als sneller zijn, zodat de noodzaak voor een grote hoeveelheid rekenkracht en computationele middelen verminderd wordt. De eerste contributie betreft Bayesian compression, een variationele Bayesiaanse inferentie procedure waarbij door middel van welgekozen kansverdelingen over de parameters van een netwerk goed presterende en computationeel efficiënte architecturen kunnen worden ontdekt met behulp van een combinatie van pruning en quantisatie. Hoewel deze benadering kan leiden tot zeer gecompresste architecturen, ontbreekt er een differentiatie van zowel pruning als quantisatie voor een specifieke taak of probleem. Om deze reden zijn de andere twee contributies apart gefocused op pruning en quantisatie. De tweede contributie betreft een nieuwe methode voor optimalisatie van de L_0 norm, de gouden standaard voor sparsiteit, van neurale netwerken. We stellen een algemene techniek voor waarbij een geschikt niveau van ruis optimalisatie aan de hand van gradiënten van de niet-differentieerbare L_0 norm mogelijk maakt. Aan de hand van empirische resultaten laten we zien dat deze aanpak leidt tot nauwkeurige modellen met een hoge sparsiteit, waarbij getraind kan worden met een spars model door middel van conditionele computatie en geschikte software. Het laatstgenoemde kan ook kortere trainingstijden faciliteren. Tenslotte bestaat de laatste contributie uit vergelijkbare ideeën waarbij we relaxed quantization introduceren; een optimalisatie procedure met behulp van gradiënten die het mogelijk maakt om neurale netwerken te leren met parameters en activaties die liggen op een (adaptief) gequantiseerd grid. We laten met empirische resultaten zien dat dit het mogelijk maakt om accurate neurale netwerken op trainen op taken van grote schaal, terwijl er slechts gebruik wordt gemaakt van 4 bits per weight en activatie.

We sluiten dit proefschrift af door antwoord te geven op de hoofd onderzoeksvragen, waarbij ook de valkuilen en nadelen van de voorgestelde methodes worden besproken, alsmede veelbelovende onderzoeksrichtingen.

ACKNOWLEDGEMENTS

I would first like to extend my gratitude to my supervisor, Max Welling; if it was not for you, I would not have pursued a Ph.D. You gave me the opportunity, guided me through it, and encouraged me to explore things that were interesting to me. Our discussions about novel ideas on the whiteboard were always a pleasure and I've learned quite a lot from you. I would also like to thank Klamer Schutte, for giving me the opportunity to perform this research and for pushing me to simplify things, Diederik Kingma, for great discussions that changed my way of thinking and Richard Zemel, for giving me the opportunity to research (very) interesting topics and visit the wonderful Toronto. I would also like to graciously thank all the members of my defence committee, Yee Whye Teh, Dmitry Vetrov, Clarisa Sánchez Gutiérrez, Patrick Forré and Efstratios Gavves.

I would also like to thank my great collaborators; Matthias Reisser, for the multiple insightful discussions and always pushing me for something more, Tijmen Blankevoort, for introducing me to more practical ways of thinking and Karen Ullrich, Xiahan Shi, for multiple discussions which improved my understanding on several topics. I would also like to thank Uri Shalit, David Sontag and Joris Mooij for sharing their expertise with me.

Furthermore, I would like to thank Rianne van den Berg, for all the fun times and for translating the abstract of this thesis, Patrick Forré, for stimulating discussions about novel ideas, and Jakub Tomczak, Thomas Kipf, along with all the other people from my time at the university of Amsterdam; you made the 4 years of this PhD extremely enjoyable!

Last but not least, I would also like to thank Dimitra Zafeiropoulou and my family; your support and encouragement was a big catalyst that lead to this outcome.