Compressing neural networks

Sjoerd Roelof de Jong
Fitzwilliam College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Machine Learning, Speech, and Language Technology

University of Cambridge
Engineering Department
Trumpington Steet
Cambridge CB2 1PZ
United Kingdom

Email: sd782@cam.ac.uk

August 17, 2018

Declaration

I Sjoerd Roelof de Jong of Fitzwilliam College, being a candidate for the M.Phil in Machine Learning, Speech, and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 13177

Signed:

Date:

Acknowledgements

I wish to thank my supervisors, Andreas Damianou and Pablo Moreno, both for their guidance, technical support and their very valuable comments and feedback.

Abstract

Modern neural networks achieve state of the art performance on a wide range of machine learning tasks, but this performance comes at the cost of large memory, power and computational requirements. Compression of neural networks tries to offer a solution to this conundrum. In this thesis we explore how weight-pruning methods can achieve compression of neural networks. We specifically focus on variational inference (VI) methods, which are theoretically justified for use as a compression method through their relation with the minimum description length (MDL) [27] principle. We compare several well known VI methods with sparse priors and present a novel technique in VI; *Bayesian Group Lasso*, which achieves competitive compression rates. Furthermore we propose the use of Laplace posterior and Laplace prior for VI compression, and provide an analytical approximation to the Kullback-Leibler (KL) divergence between a Gaussian variational distribution and a Laplace posterior.

Contents

Chapter 1

Introduction

1.1 Motivation

In the last 5-10 years, deep neural networks have achieved state of the art performance on a wide range of machine learning tasks [38], sometimes surpassing human-perforamnce [25]. A major factor in this increased performance is the larger size and complexity of neural networks, which is partly facilitated by the use of faster hardware such as Graphical processing units (GPU). However this development has also greatly increased the required resources for neural networks in training and production. Modern Neural networks require a signifiant amount of working memory, energy and computation power. The storage size of the VGG convolution neural network (CNN) is over 500MB, which restricts its use on mobile and edge device applications. Deploying neural networks to resource-constrained devices such as mobile, cars, robots or smart home devices requires strong improvements in storage, memory, and energy usage.

One response to this need to decrease the size of the networks is to simply train smaller networks from scratch. However it appears that large networks are easier to optimize than smaller networks although their large size is not needed for a high performance. Multiple methods have been proposed which decrease the size of a large network significantly with no negative impact on the performance of the networks. [23, 26, 5, 16]. This suggests a possible role for compression in the training of any network, wherein first a larger network is trained, after which it is pruned to the desired size. Extra motivation for this approach comes from the rise of cloud computing, which could train very large neural networks and compress them, to later be deployed on more resource-constrained applications.

There are a larger number of approaches towards compression, of which a popular one is to prune away weights from the neural network [12]. This pruning is often preceded by training the network with a regularizer which induces sparsity in the weights of the network. Among the regularization techniques, one with particular attractive theoretical justifications is training Bayesian neural networks (BNN) using variational inference (VI) with sparsity-inducing priors. Variational inference is related to the Minimum Description Length (MDL) principle [53, 27, 28], an information-theoretic formalization of Occam's razor. In this thesis the focus is on compression of networks through pruning, with a particular focus on using variational inference with sparsity inducing priors.

1.2 Contributions

An extensive exploration and comparison of different pruning based compression methods of neural networks is provided, exploring both structured as well as unstructured sparsity methods, and exploring different sparsity-inducing priors with VI. We introduce a new technique for using VI for compression; *Bayesian Group Lasso*, which is shown to lead in a principled Bayesian manner to competitive compression results. For doing VI with Laplace priors we derive an analytical approximation of the Kullback-Leibler (KL) divergence between a Gaussian variational distribution and the prior and show its importance for reducing the variance of gradients in the network. Finally we show promising results of the use of compression for transfer learning, in which we both speed up convergence as well as improve final performance by using compressed models.

1.3 Thesis outline

Chapter 2 gives background theory and preliminaries about neural networks, Bayesian inference using VI, and the sparsity inducing regularization techniques used later on. Chapter 3 discusses related work in compression of neural networks. Chapter 4 elaborates on the methods and implementations details of this thesis, presenting simultaneously some novel methods and formulations. Chapter 5 presents and discusses the results. Finally Chapter 6 summarizes the results, draws some conclusions from the results and proposes a multitude of interesting future research directions.

Chapter 2

Background

2.1 Neural networks

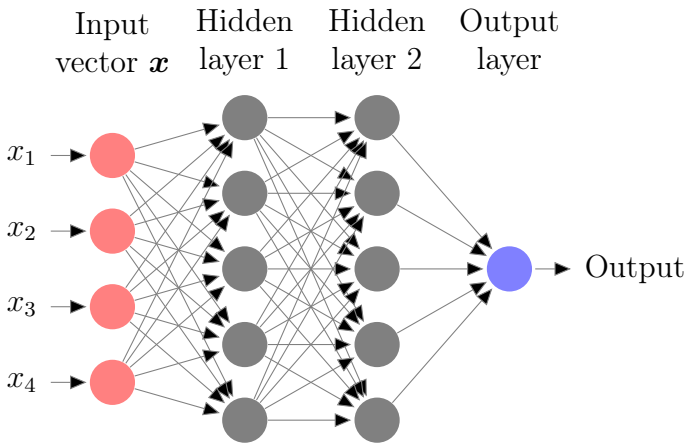
In its most basic form a neural networks consists of a set of nodes, commonly called neurons, with each neurons having ingoing and outgoing edges. The nodes take in input data represented by incoming edges, perform a transformation on this input and output the result through its outgoing edges. Deep neural networks are networks in which the neurons are arranged in consecutive layers, with each layer taking the output of the previous layer as input. When a node is connected to each of the nodes in the previous layer, the neural network layer is fully connected.

A fully connected network implements a non-linear function mapping $f : \mathbb{R}^n \to \mathbb{R}^m$, which is implemented by taking an input $\boldsymbol{x} \in R^n$, and applying iteratively multiple linear transformations, with each linear transformation followed by an non-linear activation function. The final network outputs a vector $\boldsymbol{y} \in R^m$. For a deep neural network with 2 hidden layers, the mapping f can be represented by:

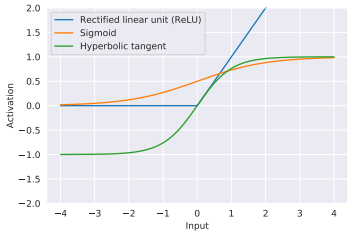
$$f(x) = \mathcal{F}_2\left(W_2\mathcal{F}_1\left(W_1\boldsymbol{x} + \boldsymbol{b_1}\right) + \boldsymbol{b_2}\right) \tag{2.1}$$

where a single layer performs first a linear transformation of the form $\boldsymbol{z} = W\boldsymbol{x}+\boldsymbol{b}$, and then applies a non-linear activation function elementwise to the vector $\boldsymbol{z}$. A diagram is shown in Figure 2.1a. Figure 5.7 Some examples of commonly used activation functions are:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent: $f(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Rectified linear united (ReLu): $f(x) = \max\{0, x\}$



(a) Simplified diagram of fully connected NN



(b) Activation functions

Figure 2.1

The activation function could in principle be a linear function, however two layers consecutive neural network layers with linear activation functions can always be written as one layer. Another widely used class of neural networks are Convolutional Neural Networks (CNN's). CNN's usually consist of a combination of convolutional layers as well as fully connected layers. The convolutional layers decrease the number of parameters by sharing weights in such a way as to encode translational invariance into the architecture. Because of this property CNN's are predominantly used in visual machine learning tasks. The neural networks used in this thesis are all discriminative models which are used for classification tasks; given an input $\boldsymbol{x}$ they output a categorical distribution over possible classes to which $\boldsymbol{x}$ could belong. The loss is then computed as the cross entropy between the categorical distribution outputted by the network represented by $\boldsymbol{y}$, and the true distribution of the classes $\boldsymbol{y}_{\text{ground-truth}}$. Neural networks are usually trained by stochastic gradient descent (SGD), in which the gradient is computed by the backpropragation algorithm [54]. Several improvements upon SGDwhich have been proposed are also widely in use today, such as the Adam optimizer, which adapts learning rates based on estimates of the first and second-order moments of the gradients [32].

2.2 Bayesian inference

Neural networks usually implement a deterministic mapping as described in Section 2.1. However when making predictions using these networks, no measure of uncertainty about the prediction is available. Furthermore, deterministic neural networks are prone to overfitting. Bayesian inference attempts to solve both these problems.

In the this section we describe shortly Bayesian inference, and in Section 2.3 we will describe a practical method to perform Bayesian inference in neural networks. Given a dataset D with observations $x_i, ...$ and a set of parameters $\boldsymbol{w}$, Bayesian inference tries to find the posterior distribution $p(\boldsymbol{w}|D)$, given by Bayes rule:

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)} = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{\int p(D|\boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}} \tag{2.2}$$

where $p(\boldsymbol{w})$ denotes the prior distribution over the parameters, $p(D|\boldsymbol{w})$ is the likelihood function and p(D) is called the evidence, sometimes called the marginal likelihood. If the posterior is known, it can be used to compute the predictive posterior distribution:

$$p(\tilde{x}|D) = \int p(\tilde{x}|\boldsymbol{w}, D)p(\boldsymbol{w}|D) \tag{2.3}$$

where $p(\tilde{x}|D)$ is the predictive posterior distribution which gives the distribution over future possible observations $\tilde{x}$, conditioned on the data already seen in dataset D. Bayesian inference is in contrast with maximum likelihood estimation in which the goal is to try to find the parameters $\boldsymbol{w}$ that maximize the likelihood function $p(D|\boldsymbol{w})$. The main difficulty of performing full Bayesian inference is the computation of the evidence: $p(D) = \int p(D|\boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}$. Evaluating the integral analytically is intractable even for small neural networks and the posterior distribution is thus usually approximated. For this reason, the problem is often reformulated to the tractable Maximum a Posteriori (MAP) estimation in which the goal is to find the mode of the posterior distribution:

$$\boldsymbol{w}_{\text{MAP}} \approx \underset{\boldsymbol{w}}{\operatorname{argmax}}\, p(D|\boldsymbol{w})p(\boldsymbol{w}). \tag{2.4}$$

However there are approximations to full Bayesian inference in use. Approximation methods for the posterior distributions roughly fall into two classes [6]. Markov chain Monte Carlo (MCMC) methods provide a way of approximating the posterior distribution through sampling. The other class of approximation methods is variational inference, which is further described in Section 2.3.

2.3 Variational inference

In variational inference, the posterior distribution $p(\boldsymbol{w}|D)$ is approximated by a parameterized variational distribution $q_{\boldsymbol{\phi}}(\boldsymbol{w})$, with variational parameters $\boldsymbol{\phi}$. The goal is to vary the variational parameters $\boldsymbol{\phi}$ in such a way as to make the variational distribution as similar to the posterior as possible. There are different measures in use for the similarity of the two distributions. The most commonly used is the Kullback-Leibler

divergence [36] (KL-divergence) given by:

$$KL(q_{\boldsymbol{\phi}}(\boldsymbol{w})||p(\boldsymbol{w})) = \mathbb{E}_q\left[\log\left(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{w})}{p(\boldsymbol{w})}\right)\right]. \tag{2.5}$$

The KL divergence is non-negative [1] and non-symmetric under swapping of its inputs arguments [7]. The KL-divergence can also be replaced by its generalization of the Rényi divergences [41]:

$$D_\alpha[q_{\boldsymbol{\phi}}(\boldsymbol{w})||p(\boldsymbol{w})] = \frac{1}{\alpha-1}\log\int q_{\boldsymbol{\phi}}(\boldsymbol{w})^\alpha p(\boldsymbol{w})^{1-\alpha}d\boldsymbol{w} \tag{2.6}$$

The KL-divergence between the posterior and its variational approximation can be minimized by maximizing the evidence-lower-bound (ELBO) which can be derived by rewriting Equation 2.2 as:

$$\log p(D) = \log p(D|\boldsymbol{w}) + \log p(\boldsymbol{w}) - \log p(\boldsymbol{w}|D) \tag{2.7}$$

$$= \log p(D|\boldsymbol{w}) + \log p(\boldsymbol{w}) - \log q_{\boldsymbol{\phi}}(w) + \log q_{\boldsymbol{\phi}}(w) - \log p(\boldsymbol{w}|D) \tag{2.8}$$

and by taking the expectation over $q_{\boldsymbol{\phi}}(w)$;

$$\mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p(D))] = \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p(D|\boldsymbol{w}))] + \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p(\boldsymbol{w})) - \log(q_{\boldsymbol{\phi}}(w))] + \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(q_{\boldsymbol{\phi}}(w)) - \log(p(\boldsymbol{w}|D))] \tag{2.9}$$

$$\log(p(D)) = \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p(D|\boldsymbol{w}))] + \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log\left(\frac{p(\boldsymbol{w})}{q_{\boldsymbol{\phi}}(w)}\right)] + KL[q_{\boldsymbol{\phi}}(w)||p(w|D)]. \tag{2.10}$$

Since the model evidence $p(D)$ is a constant with respect to the variational distribution $q_{\boldsymbol{\phi}}(\boldsymbol{w})$, and the KL-divergence is nonnegative, the KL-divergence between $q_{\boldsymbol{\phi}}(w)$ and the true posterior distribution can thus be minimized by maximizing with respect to the variational parameters:

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p(D|w)] + \mathbb{E}_{q_{\boldsymbol{\phi}}}\left[\log\frac{p(w)}{q_{\boldsymbol{\phi}}(w)}\right] \tag{2.11}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p(D|w)] - KL[q_{\boldsymbol{\phi}}(w)||p(w)] \tag{2.12}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p(D|w)] + H_{\text{cross}}(q_{\boldsymbol{\phi}}, p(w)) - H(q_{\boldsymbol{\phi}}(w)) \tag{2.13}$$

where $H(q_{\boldsymbol{\phi}}(w))$ denotes the entropy of the variational distribution and $H_{\text{cross}}(q_{\boldsymbol{\phi}}, p(w))$ the cross-entropy between the variational distribution and the prior. The ELBO $\mathcal{L}(\boldsymbol{\phi})$ is occasionally called the negative variational free energy. In practice instead of maxi-

[1] That the KL-divergence is non-negative can quickly be seen by using Jensen's inequality for concave functions: $-KL(q(x)||p(x)) = \int \log(\frac{p(x)}{q(x)}) \leq \log\int q(x)(\frac{p(x)}{q(x)}) = \log(1) = 0$.

mizing the ELBO, often the negative ELBO is minimized by gradient descent methods. There is a close relationship between Bayesian variational inference and the minimum description length principle [45]. Subsection 2.3.1 will elaborate further on this connection and discuss bits-back coding a technique which allows re-interpreting the ELBO as a code length of the data [28]. Although the MDL principle and bits-back coding provides another theoretical justification and interesting connection between variational inference and compression, it is not necessary to understand the compression techniques used later on, and readers may wish to skip to Section 2.4 in which the method of parameterization of the variational posterior is discussed.

2.3.1 Minimum description length principle

Occam's razor is a principle that gives a preference for simpler theories/models [45]. One formalization of Occam's razor is the minimum description length principle [53], which states that the model which has the minimum description length for some dataset D is the best model. The description length consists of the number of bits needed to describe the model, as well as the number of bits needed to encode the data using the model, which is simply the number of bits needed to describe the deviations from the prediction made by the model M. The model here is chosen from a set of allowable models, (e.g. all 2 dimensional Gaussians distributions). More formally, given a set of possible models $\mathcal{M}$ and a dataset D, the best model is the one which minimizes the description length L(M):

$$M = \underset{M \in \mathcal{M}}{\operatorname{argmin}}(L(M) + L(D|M)) \tag{2.14}$$

where $L(M)$ denotes the description length of the model and $L(D|M))$ the description length of the data given the model. In the context of neural networks, the set of possible models $\mathcal{M}$ can be defined in many ways. One way is to let $\mathcal{M}$ be a certain neural network architecture with all possible weight configurations, but $\mathcal{M}$ can also contain different neural network architectures.

Bits-back coding; a connection between the MDL principle and variational inference

To evaluate the description length of a neural network, it makes sense to consider a setup with a sender and receiver, in which the sender tries to communicate a dataset of observations, by encoding and sending both the weights as well as the data misfits achieved using those weights. One approach is to simply encode the data misfits and

the weights, where the weights are quantized floatingpoint numbers. However, by using a variational distribution, the description length can be made smaller due to *bits-back* at the end of coding scheme [27].

For a neural network with parameters $\boldsymbol{w} = \{w_j ... w_K\}$, and a dataset $D = \{x_i ... x_N\}$ MAP estimation with a factorized prior distribution $p(\boldsymbol{w}|M) = \prod_i p(w_i|M))$ is given by:

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \big[-(p(D|\boldsymbol{w}, M)p(\boldsymbol{w}|M)) \big] \tag{2.15}$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} \big[-\log p(D|\theta, M) - \log p(\theta|M) \big] \tag{2.16}$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} \big[-\sum_i \log(p(x_i|w_i, M) - \sum_j \log p(w_i|M) \big]. \tag{2.17}$$

In the case of weights which are encoded to finite precision, as is always the case in neural networks, the continuous distribution $p(x_i|\boldsymbol{w}, M)$ and $p(w_j|M)$ can be approximated by discrete distributions, where the distribution is split up into intervals with a certain width ϵ. The weight have been defined up to a tolerance ϵ_w and the observations up to a tolerance of ϵ_x:

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \big[-\sum_i \log \left(p \left(x_i|w_i, M \right) \epsilon_x \right) - \sum_j \log \left(p \left(w_j|M \right) \epsilon_w \right) \big]. \tag{2.18}$$

if we now assume that for a given x_i, the value of x_i is discretized (a floating point value) and can take values from the countably finite set $Z = \{z_1, ...\}$, then $\log(p(x_i|\boldsymbol{w}, M)\epsilon)$ with $x_i \in Z$ can be interpreted as the description length of the data misfit of x_i up to precision ϵ_x. In the same manner $\log(p(w_j|M)\epsilon)$ can be interpreted as the description length of the parameter w_j given a prior distribution $p(\boldsymbol{w}|M)$ and using precision ϵ_w. This is made more clear by Theorem 2.3.1.

Theorem 2.3.1. *(Extended Kraft inequality [13]) For any countably infinite set of codewords $S = \{s_i, ...\}$ over an alphabet of size A, the length $l_i(s_i)$ of the codewords satisfies the inequality:*

$$\sum_i^{\infty} A^{-l_i} \leq 1 \tag{2.19}$$

conversely, given a set of codeword lengths $L = \{l_i, ...\}$ satisfying the inequality in Equation 2.19, a prefix code with these codeword lengths: $L = \{l_i, ...\}$ can be constructed.

Note that thus for any normalized discrete probability distribution $p(x)$ over a set $X = \{x_i, ...\}$, a prefix code C for X exists, such that each x_i has a length of $\log(p(x))$. Thus minimizing Equation 2.18 is equivalent to minimizing the description length of

the data D using a model M and parameters $\boldsymbol{w}$. If the ϵ's in Equation 2.18 are ignored, minimizing the description length becomes equal to doing maximum-a-posteriori estimation. While still ignoring the ϵ's, a variational distribution $q(\boldsymbol{w})$ can be introduced. The variational distribution is also assumed to be a discrete distribution over values of $\boldsymbol{w}$ which given up to precision of ϵ_w. Instead of describing a weight w_j by a single codeword, now there are multiple codewords (the number of values which w_j is allowed to take), where the probability of choosing a certain codeword is given by $q(\boldsymbol{w})$. The description of the data under the model L then thus becomes:

$$\mathbb{E}_q[L(D)] = -\mathbb{E}_q[L(\boldsymbol{w})] - \mathbb{E}_q[L(D|\boldsymbol{w})] \tag{2.20}$$

$$= -\mathbb{E}_q[\log p(\boldsymbol{w}|M)] - \mathbb{E}_q[\log p(D|\boldsymbol{w}, M)] \tag{2.21}$$

However, when coding using the variational distribution $q(\theta)$ the receiver can reconstruct $q(\theta)$ and get *bits-back* according to the following scheme [28] [27]:

- The sender and receiver have agreed on a prior distribution and an inference algorithm, which given a set of observations, yields the variational distribution $q(\boldsymbol{w})$.
- The sender runs the inference algorithm and computes variational distribution $q(\boldsymbol{w})$.
- The sender collapses variational distribution using a source of auxiliary bits to weight values w_j.
- The sender encodes weights $\boldsymbol{w}$ using the prior distribution $p(\boldsymbol{w}|M)$, and the data misfits using $p(x_i|\boldsymbol{w}, M)$
- Receiver decodes the weights also using $p(\boldsymbol{w}|M)$.
- The receiver can then decode the observations x_i from the data misfits and $p(x_i|\theta, M)$.
- Using the same inference algorithm as the sender, the receiver can now decode the variational distribution $q(\boldsymbol{w})$.
- Using this distribution $q(\boldsymbol{w})$, and the weights $\boldsymbol{w}$ the receiver can now infer the auxiliary bits which where used by the sender to collapse the variational distribution. Thus, the auxiliary bits have now also been communicated, resulting in *bits-back*

The amount of information which can be communicated through the auxiliary bits is given by the entropy of $q(\boldsymbol{w})$. Thus the entropy of the $q(\boldsymbol{w})$ need to be subtracted

from the description length given in Equation 2.21, which gives:

$$\mathbb{E}_q[L(D)]_{\text{bits-back}} = -\mathbb{E}_q[\log p(\boldsymbol{w}|M)] - \mathbb{E}_q[\log p(D|\boldsymbol{w}, M)] - (-\mathbb{E}_q[\log q(\boldsymbol{w})]) \quad (2.22)$$

$$= -\mathbb{E}_q[\log p(D|\boldsymbol{w}, M)] + \mathbb{E}_q[\log \left(\frac{p(\boldsymbol{w}|M)}{q(\boldsymbol{w})}\right)] \quad (2.23)$$

$$= -\mathbb{E}_q[\log p(D|\boldsymbol{w}, M)] + KL[q(\boldsymbol{w})||p(\boldsymbol{w}|M)] \quad (2.24)$$

$$= -\mathbb{E}_q[\log p(D|\boldsymbol{w}, M)] + H(q(\boldsymbol{w})||p(\boldsymbol{w}|M)) - H(q(\boldsymbol{w})) \quad (2.25)$$

where Equation 2.24 is the negative ELBO, and minimizing the description length is thus equivalent to maximizing the ELBO, as in variational inference. Whether the auxiliary bits are of any practical value does not change the fact that the information has been transmitted, resulting in the *bits-back*. Moreover, practical coding schemes using bits-back coding have been developed (see [17]).

2.4 Reparameterization tricks

To actually optimize variational inference objective as given by Equation 2.12, the gradient of the ELBO needs to be computed. The objective in Equation 2.12 is an expectation over the variational distribution $q_{\boldsymbol{\phi}}(\boldsymbol{w})$, and thus the gradient we want to compute is:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\mathcal{L}_{ELBO}] = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}\, q(\boldsymbol{w})\mathcal{L}_{ELBO} \quad (2.26)$$

$$= \nabla_{\phi} \int q(\boldsymbol{w})\mathcal{L}_{ELBO}(\boldsymbol{w})d\boldsymbol{w}. \quad (2.27)$$

However this integral intractable in neural networks, and it is approximated by Monte Carlo samples, which makes the gradient usually infeasible to compute [30]. To still estimate the gradient there are various reparameterization tricks in use.

2.4.1 Global reparameterization trick

One method to compute the gradient using the backpropagation algorithm is to use the global reparameterization trick [8]. Let $\boldsymbol{w} = t(\boldsymbol{\phi}, \boldsymbol{\epsilon})$, where t is a deterministic function, $\boldsymbol{\phi}$ are the variational parameters and $\boldsymbol{\epsilon}$ a random sample such that $q(\boldsymbol{w})d\boldsymbol{w} = p(\boldsymbol{\epsilon})d\boldsymbol{\epsilon}$.

Denoting the ELBO objective by $\mathcal{L}_{ELBO}(\boldsymbol{w}) = \mathcal{L}_{ELBO}(t(\boldsymbol{\phi}, \boldsymbol{\epsilon}))$. Then:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\mathcal{L}_{ELBO}] = \nabla_{\boldsymbol{\phi}} \int q(\boldsymbol{w}) \mathcal{L}_{ELBO}(\boldsymbol{w}) d\boldsymbol{w} \tag{2.28}$$

$$= \nabla_{\boldsymbol{\phi}} \int p(\boldsymbol{\epsilon}) \mathcal{L}_{ELBO}(t(\boldsymbol{\phi}, \boldsymbol{\epsilon})) d\boldsymbol{\epsilon} \tag{2.29}$$

$$= \int \nabla_{\boldsymbol{\phi}} p(\boldsymbol{\epsilon}) \mathcal{L}_{ELBO}(t(\boldsymbol{\phi}, \boldsymbol{\epsilon})) d\boldsymbol{\epsilon} \tag{2.30}$$

$$= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} p(\boldsymbol{\epsilon}) \mathcal{L}_{ELBO}(t(\boldsymbol{\phi}, \boldsymbol{\epsilon}))] \tag{2.31}$$

$$\tag{2.32}$$

The swapping of the gradient and the expectation is known as the *pathwise derivative* estimator and requires t to be a continous function for all $\boldsymbol{w}$. It is sufficient for t to be almost-everywhere differentiable [55] [19] [8]. These conditions are met for practically all commonly used neural network activations and output functions. The general procedure is given in Algorithm 1. In the case of a Gaussian factorized posterior, the variational distribution of a particular weight can be reparameterized as $w = \mu + \sigma \cdot \epsilon = \mu + \log(1 + exp(\rho)) \cdot \epsilon$ [8] where μ and ρ are the variational parameters and ϵ is parameter free and sampled from a standard normal $\mathcal{N}(0, 1)$. ρ is used instead of the standard deviation σ to enable unconstrained optimization while still enforcing the non-negativity of the variance through the softplus. This is equivalent to sampling from a Gaussian $\mathcal{N}(\mu, \log(1+\exp(\rho))) = \mathcal{N}(\mu, \sigma^2)$ The softplus function $\log(1+\exp(\rho)$ is used to enforce positivity of the standard deviation, while allowing unconstrained stochastic gradient descent optimization of the parameter ρ.

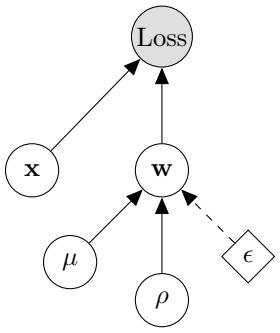


Figure 2.2: Reparameterized neural network, allowing backpropagation through variational parameters.

The posterior however does not have to be a Gaussian, and any distribution which can be reparameterized can be used. Any proper probability distribution is also a valid choice for the prior. By choosing an appropriate prior and variational distributions, the KL-divergence between the prior and variational distribution term can be computed

analytically instead of using Monte Carlo samples. This can result in lower variance in the gradients, which might help optimization and enable faster convergence.

Algorithm 1 Stochastic gradient variational Bayesian (SGVB) inference [8] [34]

Initialise variational parameters ϕ
repeat
  1. Sample minibatch D_i, with $\frac{N}{M}$ datapoints
  2. Sampling weights
  $\boldsymbol{\epsilon} \leftarrow p(\boldsymbol{\epsilon})$
  $\boldsymbol{w} \leftarrow t(\boldsymbol{\phi}, \boldsymbol{\epsilon})$
  3. Compute loss
  $\text{ELBO}_{minibatch} \leftarrow \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p(D_i|\boldsymbol{w}))] - \frac{1}{M} KL[q_{\boldsymbol{\phi}}(\boldsymbol{w})||p(\boldsymbol{w})]$
  4.Compute gradients
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\phi}} \Big[\text{ELBO4}_{minibatch}\Big]$
  5. Update variational parameters $\boldsymbol{\phi}$ using gradients $\boldsymbol{g}$
until convergence

2.4.2 Local reparameterization trick

In the global reparameterization trick, the Monte Carlo samples of the weights are the same for all datapoints in the minibatch. Writing the loss of a minibatch with K datapoints with the total dataset containing N datapoints as; $L_D^{SGVB}(\boldsymbol{\phi}) = \frac{N}{K} \sum_{i=1}^{K} L_i$, where L_i is the dataloss term $-\log(p(D|\theta, m))$ the variance of this estimator is given by [33]:

$$Var[L_D^{SGVB}(\boldsymbol{\phi})] = N^2(\frac{1}{K} Var[L_i] + \frac{M-1}{M} Cov[L_i, L_j]). \tag{2.33}$$

The second term does not decrease even when the minibatch size is increased. To decrease the variance of the estimator, it is preferred to have the covariance between the loss of two different datapoints in a minibatch be zero: $Cov[L_i, L_j] = 0$. One way to achieve this is to sample new weights for each datapoints in the minibatch. This is however computationally expensive. The local reparameterization trick circumvents this problem by sampling the activations of a neural network layer instead of the weights. For a minibatch with K datapoints, and a neural network layer with B neurons and an input dimension of A, this requires KB samples when using the local reparameterization trick, whereas if new weights would be sampled for each datapoint in the minibatch, the number of required samples would be KBA. The reason that the activations can be sampled instead of the weights depends on the fact that the sum of N Gaussian independently distributed random variables with means $\mu_1, \mu_2, ..., \mu_N$ and variances $\sigma_1^2, \sigma_2^2, ..., \sigma_3^2$ then the sum $S_N = \sum_i^N X_i$ is again Gaussian distributed as distributed $S_N \sim \mathcal{N}(\sum_i^N \mu_i, \sum_i^N \sigma_i^2)$ This property of Gaussian distribution makes

them particularly suitable for use in conjunction with the local reparameterization trick. The sum of N independent Laplace random variables for example has not a simple distribution from which can be sampled [35].

2.5 Compression

The central problem of compression of neural networks is to take a neural network of a certain size, and make it smaller according to some pre-defined metric. Although compression traditionally primarily deals with decreasing the number of bits required to store information, in the context of neural networks there are more considerations than only the number of bits. Next to the storage size, the memory usage, computational speed and energy usage are also important. Different metrics and compression methods are often optimal only for one of the pre-named factors. Hence it is necessary to tailer the implementation of the compression algorithm to the desired application. Compression methods in use for neural networks can be roughly divided into three groups: weight pruning, weight sharing, low-rank factorization and knowledge distillation, sometimes called student-teacher training.

In weight pruning either weights or groups of weights are pruned by setting them to zero. This is usually preceded by first training the network in a way which enforces sparsity. The selection of which weights to prune can then be done on the basis of the absolute values of the weights or other heuristics. Weights can also be quantized to a lower precision, thus requiring less bits to represent them. By letting different weights share the same value, the required bits to represent the complete set of weights can be decreased. A drawback of this method is that in general the memory usage and speed of the network is not decreased, since the full weight matrix needs to be restored.

Low-rank approximation methods attempt to exploit redundancy in the weight matrix and decompose it to speed up computation, which is further discussed in Section 3.2. In knowledge distillation or student-teacher training, a smaller network attempts to mimick the function implemented by a larger network. This can be done by optimizing on the ℓ_2 loss between the logits (the inputs to a softmax function) of the smaller and larger network, or by taking the cross-entropy between the smaller network and larger network's softened softmax output [26].

2.6 Regularization of neural networks

Since regularization plays a large role in the weight pruning methods of compression, we briefly review ℓ_2-regularization and ℓ_1-regularization, which albeit being basic techniques, are still widely used and have often close relations to other regularization techniques.

In ℓ_2 regularization the penalty term which is added is:

$$\ell_{2,\text{penalty}} = \lambda_{\ell_2} |\boldsymbol{w}|_2^2 = \lambda \sum_i w_i^2 \tag{2.34}$$

where the sum is over all the weights, which is actually the square of the ℓ_2-norm and λ_{ℓ_2} is a hyperparemeter controlling the strength of regularization. ℓ_2 regularization can be shown to be equivalent to doing MAP-inference with a zero mean Gaussian prior.

In ℓ_1 regularization, a penalty term is added to the loss term, which consists of the sum of the absolute values of all the weights:

$$\ell_{1,\text{penalty}} = \lambda_{\ell_1} |\boldsymbol{w}|_1 = \lambda_{\ell_1} \sum_i |w_i| \tag{2.35}$$

where λ_{ℓ_1} is a hyperparameter used to control the strength of the ℓ_1 regularization. Since neural networks normally optimize the negative log-likelihood ℓ_1 regularization is equivalent to doing MAP-inference with a zero mean Laplace prior Laplace$(0, b)$:

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\text{argmin}} \Big[-\log p(D|\boldsymbol{w}) - \log(p(\boldsymbol{w}) \Big] \tag{2.36}$$

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\text{argmin}} \Big[-\log p(D|\boldsymbol{w}) + \sum_i \log \big(\frac{1}{2b} e^{-|w_i|} \big) \Big] \tag{2.37}$$

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\text{argmin}} \Big[-\log p(D|\boldsymbol{w}) + \sum_i |w_i| \Big] \tag{2.38}$$

$$\tag{2.39}$$

Compared to ℓ_1 regularization, ℓ_2 regularization will in general lead to less sparse solutions. This can be seen since the gradient $\frac{\partial |\boldsymbol{w}|_2^2}{w} = 2w$ and thus gets smaller as the weight w gets closer to zero, whereas for ℓ_1 regularization it is $\frac{\partial |\boldsymbol{w}|_1}{w} = 1$ which is independent of the weight value.

2.6.1 Group lasso

If there is structure in the sparsity pattern of the weights this has the advantage over random sparsity patterns that it might allow directly decreasing the size neural network architecture. When for example all the weights corresponding to one neuron have been pruned, the size of the layer and the weight matrix can be directly decreased. Random sparsity in the weights is harder to directly translate in speedups [63]. Structured sparsity learning uses group Lasso regularization to bring about sparsity in a structured way. Structured here means that certain groups of parameters have all been pruned away. Group lasso was introduced by [66] and [2], in order to select groups of variables in regression problems. The group lasso regularization penalty is given by [63]:

$$l_{grouplasso} = \lambda_g \sum_{g=1}^{G} \sqrt{\sum_{i=1}^{|w_i^{(g)}|} w_i^{(g)}} \tag{2.40}$$

where G is the number of parameter groups. Group Lasso regularziation achieves this by having penalties grouped, and thus forcing unimportant structures to zero. The properties of group lasso can be illustrated with the following example. Consider a group of two weights w_1, w_2 represented by the vector $\boldsymbol{w}$, with two penalties, a group lasso $|\boldsymbol{w}|_2 = \sqrt{w_1^2 + w_2^2}$ and regular ℓ_1 regularization $|\boldsymbol{w}|_1 = |w_1| + |w_2|$. The gradients of the penalties with respect to w_1 are;

$$\frac{\partial |\boldsymbol{w}|_2}{\partial w_1} = \frac{2w_1}{2\sqrt{w_1^2 + w_2^2}} = \frac{w_1}{\sqrt{w_1^2}\sqrt{1 + \frac{w_2^2}{w_1^2}}} = \frac{1}{\sqrt{1 + (\frac{w_2}{w_1})^2}} \tag{2.41}$$

$$\frac{\partial |\boldsymbol{w}|_1}{\partial w_1} = 1 \tag{2.42}$$

gradient for ℓ_1 regularization is independent of the weights, whereas for the group lasso penalty, the gradient with respect to w_1 is dependent on w_2;

$$\frac{\partial |\boldsymbol{w}|_2}{\partial w_1} \approx \begin{cases} 1 & \text{if } w_1 \gg w_2 \\ \frac{1}{\sqrt{2}} & \text{if } w_1 = w_2 \\ \frac{|w_1|}{|w_2|} \approx 0 & \text{if } w_1 \ll w_2 \end{cases} \tag{2.43}$$

This coupled behavior results in weights going to zero together or together staying non-zero.

Chapter 3

Related Work

Compression of models has been investigated since long before the advent of deep neural networks. Hence many compression methods which have been applied to deep neural networks have their roots in older methods, although neural networks often pose unique challenges. Here a number of general compression methods are discussed. Thereafter related work in pruning based compression methods is explored.

3.1 Quantization

One of the simplest methods to compress a trained neural network is quantization of the weights to a smaller number of bits. Most deep learning frameworks have as default datatype for the weights the 32-bits floating point format, [11, 1]. Vanhoucke et al [61] quantized weights to 8-bits integers, which resulted in little accuracy loss and highly increased accuracy. Another approach is to quantize weights to a number of K values. One way to achieve this is by doing k-means clustering of the weights, where for a fixed number of n weights, the weights are clustered at k values c_j according to:

$$\min \sum_{i}^{n} \sum_{j}^{k} |w_i - c_j|^2 \tag{3.1}$$

Thus, a clusterindex is assigned to each weight w_i, which is only $\log_2(k)$ bits, and a codebook of size k is constructed, where the values c_j can be encoded to arbitrary precision.

Gong et al, [20] used k-means clustering to compress neural networks, where they achieved a 16-24 times compression of the network, while having a 1 % accuracy loss on the 1000 category Imagenet task. While this method is attractive for its simplicity

and a relative good compression rate, at prediction time the original weight matrix needs to be restored from the codebook. If the full weight matrix is restored, the memory usage is the same as in the uncompressed case and the prediction speed is not faster.

3.2 Matrix factorization

Another simple way to approximate a weight matrix $W \in \mathbb{R}^{m \times n}$ with a smaller number of parameters is by utilizing the singular value decomposition:

$$W = USV^T, \text{with } \; U \in \mathcal{R}^{m \times m}, S \in \mathcal{R}^{m \times n}, T \in \mathcal{R}^{n \times n} \tag{3.2}$$

then taking the first r singular values (diagonal elements), of S and finally truncating U and T accordingly;

$$\tilde{W} = \tilde{U}\tilde{S}\tilde{V}^T, \text{with } \; \tilde{U} \in \mathcal{R}^{m \times r}, \tilde{S} \in \mathcal{R}^{r \times r}, \tilde{T} \in \mathcal{R}^{r \times n} \tag{3.3}$$

which results in a compression rate of $(mn)/(mr + r + rn)$ [20] and results in an computational speed up of $\mathcal{O}((mn)/(mr + r^2 + nr))$ [14].

3.3 Student Teacher training (Knowledge distillation)

Student Teacher (ST) training tries to transfer knowledge from a larger teacher network to a smaller student network. Hinton et al [26] used softened softmax outputs from a teacher as an extra target for a smaller student network to train. The student tries to mimick the function mapping of the larger teacher network.

Balan et al [4] distilled a Monte Carlo approximation to the posterior predictive distribution of a Bayesian neural network into a smaller neural network. The information hidden in the teacher network is called *Bayesian Dark Knowledge*. The smaller network is a deterministic non-Bayesian neural network which is regularized by the Bayesian teacher network. The smaller network can then be used for fast approximations to the Bayesian teacher network at test time.

3.4 Pruning

A network can be made smaller in size by pruning connections, which corresponds to setting certain weights to zero. This can be done in either a random (non-structured) or structured manner. The decision of which weights should be set to zero can be made in variety of ways. One heuristic is to simply consider the magnitude of weights, and delete the weights with the smallest magnitudes. Since the perturbation needed to set these weights to zero is smaller than for large magnitude weights, the expectation is that they will result in a smaller accuracy loss or increase in the loss function. This heuristic works especially well if there is sparsity in the weights, where certain parameters in the network are orders of magnitude smaller than others.

More sophisticated methods look at second order gradients, for example in [40], Le Cun et al used a method called Optimal Brain Damage. In Optimal Brain Damage, the loss function is assumed to be at a minimum and is approximated by a Taylor series, where higher than order 2 terms are ignored, and the Hessian is assumed to be diagonal. The change of the loss function when setting weight w_i to zero is then given by:

$$\delta L = \frac{1}{2} H_{ii} (w_i)^2 \tag{3.4}$$

with H_{ii} being the i-th diagonal element of the Hessian. After computing the Hessian, for each paremeter, the saliency is computed, which is the change in the loss when that particular parameter is set to zero. The diagonal Hessian approximation is needed to enable the method to scale to practical neural network sizes.

Hassibi et al [24] formulated the problem of deciding which weight to prune as an constrained optimization problem, and where they used the inverse of the full Hessian matrix. A recursion relation was used to compute the inverse Hessian in a computationally tractable way. However since modern neural networks such as VGG-16 may contain more than 100 million parameters, [56] the computation of a full Hessian matrix might still be computationally prohibitive.

3.4.1 Pruning sparse networks

As noted above, if there are sets of weights that are approximately zero in a network, these weights can normally be pruned with minimal loss of accuracy of the network. The network can either be trained in a manner that induces sparsity in the weights, or a pre-trained network can be retrained in a way that induces sparsity in the weights. Wen et al [63] used Group Lasso as described in Chapter 2 to induce a structured

sparsity in the neural network. Ullrich et al [60] used soft weight sharing to both quantize parameters as well as forcing some parameters to zero. A mixture of k Gaussians prior was defined over the parameter, with one Gaussian component fixed at zero, and the mean and variance of the other $k-1$ Gaussian components where learned during training.

$$p(w) = \prod_{i}^{I} \sum_{j}^{J} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2) \tag{3.5}$$

This forces the weights into k clusters which, which enables each weight to be indexed to a cluster in $\log_2(k)$ bits. This results in high storage savings, however at test time, the full weight matrix needs to be restored again, resulting in no memory savings. In the case of soft weight sharing, one component of the prior induces sparsity in the network. This sparsity has a random pattern, as can be seen from Equation 3.5. As discussed in Chapter 2, there are advantages in having a structure in the sparsity.

Various variational inference methods have also been used to induce sparsity in neural networks. Blundell et al [8], showed how after training a network with variational inference, using a Gaussian variational distribution and a Gaussian prior, a large proportion of the weights could be pruned away by considering their signal-to-noise ratio: μ/σ. Variational Gaussian dropout was used in [49] to sparsifies networks. In [43], Louizos et al continued along this direction by reparameterizing variational Gaussian dropout for group sparsity using a Normal-Jeffreys prior. These methods will be further discussed in Section 4.3.

Chapter 4

Methods

All the compression methods where implemented using MXNet , and specifically its imperative Gluon API [11]. For group lasso compression, a new loss function was written. For all the variational inference, pre-defined MXNet-layers where rewritten as Bayesian layers to enable modular use in a Bayesian neural network. This abstraction allows easier deployment of the neural networks.

One way to test the performance of methods which induce sparsity in a network is to start off from random initialization with training. However the focus lies here on compression of already trained networks and thus all the compression methods where compared by compressing a pre-trained benchmark neural network.

First, metrics to measure the performance of compression will very shortly be presented. Then implementation of achieving structured sparsity via using group lasso regularization is discussed. We then discuss methods for achieving structured sparsity in neural networks, in which we present a novel method; Bayesian Group Lasso. Finally various methods to induce random sparsity using variational inference are presented.

4.1 Metrics of compression

The most standard metrics used to evaluate the performance of compression methods for neural networks are the compression rate and the speedup rate [12]. The compression rate is often simply defined as:

$$\alpha(M, M^*) = \frac{a}{a*} \tag{4.1}$$

where M^* is the compressed model, and a^* is the number of parameters of the compressed model, and a, M refer to the original model. This metric is very crude, since it does not take into account the size of one parameter, and it mainly gives information about the decrease in storage size, while the influence on the speed of the network remains undetermined. The speedup is also usally measured as a simple ratio [12]:

$$\beta(M, M^*) = \frac{t}{t*} \tag{4.2}$$

where t^* denotes the computation time of the compressed model. Computation time can be measured per training epoch or per test inference.

4.2 Structured sparsity via group lasso

The main decision to be made when compressing using group lasso is how the groups are defined. Groups can be chosen to be:

- The weights associated with one neuron
- The weights associated with one input dimension of a neural network layer.
- Filters of a convolutional neural network (CNN)
- Channels of a convolutional neural network (CNN)
- All the weights associated with one layer.

These different groups can also be combined, in that a weight can belong to multiple groups. To be able to compare structured sparsity by group lasso with other structured sparsity methods which often work with only type of group, only one type of group is used.

4.3 Structured sparsity via variational inference

4.3.1 Variational inference with normal-Jeffreys prior and Gaussian variational Gaussian dropout

Bayesian variational inference with a normal-Jeffreys prior was mentioned in Subsection 3.4.1. The implementation here follows [43] except for the fact that here the groups are defined over the neurons, instead over incoming features of a layer as in

[43]. The method uses a scale mixture of normals, with the prior given by;

$$w \sim \mathcal{N}(w; 0, z^2); \qquad z \sim p(z) \tag{4.3}$$

By choosing z in an appropriate manner, the distribution of w can be more fat-tailed and have more mass at zero. By letting groups of parameters share the same scale variable, the induced sparsity becomes structured with respect to the groups (e.g. groups of weights go to zero together). The choice for z in [43] is the improper log-uniform prior $p(z) = 1/|z|$, which gives for the factorized prior w after marginalizing over z:

$$p(w) = \int \frac{1}{|z|}\mathcal{N}(w|0, z^2)dz = \int \frac{1}{|z|}\frac{1}{\sqrt{2\pi z^2}}e^{-\frac{w^2}{2z^2}}dz \tag{4.4}$$

$$p(w) = 2\int_0^\infty \frac{1}{|w|}\frac{1}{\sqrt{2\pi}}e^{-\frac{u^2}{2}}du = \frac{1}{|w|} \tag{4.5}$$

where the substitution $u = \frac{w}{z}$ has been used. This parameterization of the log-uniform prior; called the Normal-Jeffreys prior was introduced by Figueiredo et al [15]. The z scale variable can be shared among any group of weights and the groups can thus be defined in any configururartion. Here we choose the groups to be all weights associated with one neuron. The joint prior for a particular fully connected layer is then given by:

$$p(\boldsymbol{W}, \boldsymbol{z}) = \prod_i^B \frac{1}{|z_i|}\prod_{ij}^{B,A}\mathcal{N}(w_{ij}|0, z_i^2) \tag{4.6}$$

where A the number of inputs to the layer, and B the number of neurons. The variational distribution is now reparameterized in such a way as to allow variational dropout to induce group sparsity. The joint variational distribution approximation $q_{\boldsymbol{\phi}}(\boldsymbol{W}, \boldsymbol{z})$ is given by:

$$q_{\boldsymbol{\phi}}(\boldsymbol{W}, \boldsymbol{z}) = \prod_{i=1}^B \mathcal{N}(z_i|\mu_{z_i}, \mu_{z_i}^2\alpha_i)\prod_{i,j}^{B,A}\mathcal{N}(w_{ij}|z_i\mu_{ij}, z_i^2\sigma_{ij}^2) \tag{4.7}$$

α_i here represents the dropout rate of group i. α_i is now reparameterized by letting $\sigma_{z_i}^2 = \mu_{z_i}^2\alpha_i$, and optimizing with respect to $\sigma_{z_i}^2$. The local reparameterization trick is then applied to this variational distribution approximation, by sampling the activations rather than the weights. The KL-divergence between the variational distribution and the prior for a given layer can then be split in to two terms, the first being:

$$KL(q_{\boldsymbol{\phi}}(\boldsymbol{W}|\boldsymbol{z})||p(\boldsymbol{W}|\boldsymbol{z})) = \frac{1}{2}\sum_{i,j}^{B,A}(-\log(\sigma_{ij}^2) - 1) \tag{4.8}$$

and the second term being:

$$KL(q_{\boldsymbol{\phi}}(\boldsymbol{z})||p(\boldsymbol{z})) \approx \sum_{i}^{B} k_1 \sigma_{\text{sigmoid}}(k_2 + k_3 \log(\alpha_i)) - 0.5 m_{softplus}(-\log(\alpha_i) - k_1) \quad (4.9)$$

where $k_1 = 0.6357, \quad k_2 = 1.8732, \quad k_3 = 1.4869$, σ_{sigmoid} denoting the sigmoid function and $m_{textsoftplus}$ the softplus function. The second KL-term is not an analytical expression, but is an curve fitted approximation to the sampled second KL-divergence term, introduced by [49]. Improper priors may sometimes lead to proper posterior, but that is not the case with the log-uniform prior in variational Gaussian dropout. It was proved in [29], that if there exists a weight $w \in \boldsymbol{W}$ for which the likelihood $p(D|\boldsymbol{W})$ is zero and continuous, the posterior is improper. The authors furthermore showed that there are not only pathologies around $w = 0$, but that the joint likelihood also has infinite mass in the tails of the distribution, giving rise to again an improper posterior.

4.3.2 Bayesian Group Lasso

In this section we transform the group lasso penalty into a prior to be used for variational inference and show the prior is normalizable by deriving a finite normalization constant. The prior for a certain group g takes the following form:

$$\frac{1}{Z} e^{-\frac{1}{b}\sqrt{\sum_i^{|g|}(w_i^g)^2}} \quad (4.10)$$

with $|g|$ being the number of elements in group g, b a scale parameter, and Z being a normalization constant, given by:

$$Z = \int_{\mathbb{R}^{|g|}} e^{-\frac{1}{b}\sqrt{\sum_i^{|g|}(w_i^g)^2}} dw_1 dw_2 \cdots dw_{|g|} \quad (4.11)$$

which can be transformed into an integral over a $|g|$-dimensional hypersphere using hyperspherical coordinates; $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, ..., \boldsymbol{\phi}_{|g|-1}, r$, with all the angular coordinates ranging over $[0, \pi]$, except for $\boldsymbol{\phi}_{|g|-1}$ which ranges over $[0, 2\pi]$. The radial component is given by: $r^2 = \sum_i^{|g|}(w_i^g)^2$. Since $e^{-\frac{1}{b}\sqrt{\sum_i^{|g|}(w_i^g)^2}} = e^{-\frac{r}{b}}$ only depends on r, all the angular coordinates and the Jacobian can directly be integrated out and replaced by the

surface element of the hypersphere;

$$Z = \int_0^\infty S_{|g|-1}(r) e^{-\frac{r}{b}} dr \tag{4.12}$$

$$= \frac{2\pi^{|g|/2}}{\Gamma(\frac{|g|}{2})} \int_0^\infty r^{|g|-1} e^{-\frac{r}{b}} dr \tag{4.13}$$

where $S_{|g|-1}(r)$ is the surface of an $|g|$-dimensional hypersphere [42], and Γ the gamma function. By defining $I_{g-1} = \int_0^\infty r^{|g|-1} e^{-\frac{r}{b}} dr$ and using recursively integration by parts we get:

$$I_{g-1} = \int_0^\infty r^{|g|-1} e^{-\frac{r}{b}} dr = -b r^{|g|-1} e^{-\frac{r}{b}} \Big|_0^\infty + \int_0^\infty b(|g|-1) r^{|g|-2} e^{-\frac{r}{b}} \tag{4.14}$$

$$= b(|g|-1) I_{g-2} \tag{4.15}$$

$$= (b(|g|-1))(b(|g|-2)) \cdots I_0 \tag{4.16}$$

$$= (b(|g|-1))(b(|g|-2)) \cdots \int_0^\infty e^{-\frac{r}{b}} \tag{4.17}$$

$$= (b(|g|-1))(b(|g|-2)) \cdots b \tag{4.18}$$

$$= b^{|g|}(|g|-1)! \tag{4.19}$$

$$= b^{|g|}\Gamma(|g|) \tag{4.20}$$

which thus gives:

$$Z = \frac{2\pi^{\frac{|g|}{2}}\ b^{|g|}\ \Gamma(|g|)}{\Gamma(\frac{|g|}{2})} \tag{4.21}$$

For a neural network with G defined weight groups, the full prior then becomes:

$$p(\mathbf{w}) = \prod_{j=1}^{G} \frac{\Gamma(\frac{|g_j|}{2})}{2\pi^{\frac{|g_j|}{2}}\ b^{|g_j|}\ \Gamma(|g_j|)} e^{-\frac{1}{b}\sqrt{\sum_i^{|g_j|}(w_i^{g_j})^2}} \tag{4.22}$$

This prior is combined with a factorized Gaussian as a variational distribution. This also allows using the local reparameterization trick. For the fully factorized Gaussian variational distribution, there is no analytical expression for the KL-divergence, which is thus estimated by means of Monte Carlo sampling. Note that the KL-divergence can be split as usual in an entropy term and cross-entropy term, where the entropy of the factorized Gaussian posterior is known analytically, whereas only the cross-entropy term needs to be sampled. Bayesian Group Lasso has been used in Bayesian statistics in the context of regression [65], and the Bayesian Group Lasso prior has been shown to be equivalent to a scale mixture of Normal distributions with Gamma hyperpriors [37]. For a prior over a group g of size $|g|$ with weights $\boldsymbol{w^{(g)}} = \{w_1, ...w_{|g|}\}$

as in Equation 4.10, the prior can be expressed as:

$$p(\boldsymbol{w^g}|\tau_g) = \prod_i^{|g|} \mathcal{N}(w_i; 0, b\tau_g^2), \quad p(\tau_g^2) = \text{Gamma}\Big(\frac{|g|+1}{2}, \frac{1}{2}\Big) \tag{4.23}$$

In the Appendix we show how there exists an analytical approximation for the KL-divergence between a Gaussian variational distribution where the weights in one group share the variance parameter:

$$q_{\boldsymbol{\phi}}(\boldsymbol{w}) = \prod_g^G \prod_i^{|g|} \mathcal{N}(\mu_{g,i}, \sigma_g^2) \tag{4.24}$$

However here only a fully factorized Gaussian variational distribution is used, and the exploration of using a shared variance is left for future work.

4.4 Random sparsity via variational inference

In this section we introduce the a number of prior and variational-posterior combinations for variational inference to induce random, unstructured sparsity in a network. We present a novel formulation in which we combine a Laplace variational distribution and a Laplace prior to induce sparsity. Furthermore we derive an analytical approximation to the KL-divergence between a variational Gaussian distribution and a Laplace prior. If a Gaussian variational distribution is used, the local reparameterization trick is used to sample the activations rather than the weights.

4.4.1 Gaussian variational distribution with a Gaussian prior

The KL-divergence between two one-dimensional Gaussians is given by [18]:

$$KL(\mathcal{N}_q(;\mu_q,\sigma_q^2), \mathcal{N}(;\mu_p,\sigma_p^2)) = \frac{1}{2\sigma_p^2}[(\mu_q-\mu_p)^2 + \sigma_q^2 - \sigma_p^2] + \log\frac{\sigma_p}{\sigma_q} \tag{4.25}$$

For a zero-mean prior with variance σ_p^2, the total KL-divergence cost for the whole network thus becomes:

$$KL(q_{\boldsymbol{\phi}}(\boldsymbol{w})||p(\boldsymbol{w})) = \sum_i \Big(\frac{1}{2\sigma_p^2}[(\mu_{q,i})^2 + \sigma_{q,i}^2 - \sigma_p^2] + \log\frac{\sigma_p}{\sigma_{q,i}}\Big) \tag{4.26}$$

where the index i ranges over all the weights in the network.

4.4.2 Laplace variational distribution, laplace prior

In contrast with the other methods for random sparsity, the local reparameterization can not be used here. Since a uni-variate Laplace distribution has the following cumulative density function (CDF) [3]:

$$F(x) = \begin{cases} \frac{1}{2}e^{-x}, & \text{if } x \leq 0 \\ 1 - \frac{1}{2}e^{-x}, & \text{if} x > 0 \end{cases} \tag{4.27}$$

and by computing the inverse CDF and using inverse transform sampling, the global reparameterization trick for a factorized Laplace posterior can be written as:

$$\text{Laplace}(w) = \mu - b \ \text{sign}(\epsilon) \log(1 - 2|\epsilon|) \tag{4.28}$$

with:

$$\epsilon \sim \text{Uniform}\big[(-0.5, 0.5)\big] \tag{4.29}$$

There is some small chance of numerical instabilities in MXNet when ϵ is very close to zero, and thus the expression in 4.28 must be replaced by:

$$\text{Laplace}(\boldsymbol{w_i}) = \mu - b \ \text{sign} \log(1 - 2|\epsilon| + \alpha) \tag{4.30}$$

with α being a very small postivive constant. An analytical expression for the KL-divergence between two Laplace distributions $\text{Laplace}(\mu_1, b_1)$, $\text{Laplace}(\mu_2, b_2)$ is given by [18]:

$$KL(\text{Laplace}(\mu_1, b_1)||\text{Laplace}(\mu_2, b_2)) = \log\left(\frac{b_2}{b_1}\right) + \frac{|\mu_1 - \mu_2|}{b_2} + \frac{b_1}{b_2}e^{-|\mu_1 - \mu_2|/b_1} - 1 \tag{4.31}$$

which for a zero-mean Laplace prior $p(\boldsymbol{w}) \sim \text{Laplace}(0, b_p)$ and $q_{\boldsymbol{\phi}}(\boldsymbol{w}) \sim \text{Laplace}(\mu_q, b_q)$gives:

$$KL(q_{\boldsymbol{\phi}}||p(\boldsymbol{w})) = \log\left(\frac{b_p}{b_q}\right) + \frac{|\mu_q|}{b_p} + \frac{b_q}{b_p}e^{-|\mu_q|/b_q} - 1 \tag{4.32}$$

4.4.3 Variational inference with Laplace prior and Gaussian variational distribution

Similar to Subsection 4.4.1 the local reparameterization trick is used. The KL divergence between a factorized posterior Normal distribution $q(w_i) \sim \mathcal{N}(w_i; \mu_{G,i}, \sigma_i^2)$ and

a prior Laplace distribtion $p(w_i) \sim \text{Laplace}(w_i; \mu_{L,i}, b_i)$ for one weight is given by:

$$KL(q(w)||p(w)) = \int_R \mathcal{N}(w; \mu_G, \sigma^2) log \frac{\mathcal{N}(w; \mu_G, \sigma^2)}{\text{Laplace}(w; \mu_L, b)} dw \tag{4.33}$$

$$= \int_R \mathcal{N}(w; \mu_G, \sigma^2) log(\mathcal{N}(w; \mu_G, \sigma^2)) dw - \tag{4.34}$$

$$\int_R \mathcal{N}(w; \mu_G, \sigma) log(\mathcal{L}(\mu_L, s))) dw \tag{4.35}$$

$$= -\mathcal{H}(q(w)) - \int_R \mathcal{N}(w; \mu_G, \sigma^2) log(\frac{1}{2b}) dw + \int_R \mathcal{N}(w; \mu_G, \sigma^2) \frac{|w - \mu_L|}{b} dw \tag{4.36}$$

$$= -\mathcal{H}(q(w)) - log(\frac{1}{2b}) + \int_R \mathcal{N}(w; \mu_G, \sigma^2) \frac{|w - \mu_L|}{b} dw \tag{4.37}$$

where $\mathcal{H}(q(w))$ is the entropy of the posterior distribution for weight w, which for the variational Gaussian posterior has a closed form solution. The third term in Equation 4.37 can be rewritten as;

$$\int_R \mathcal{N}(w; \mu_G, \sigma^2) \frac{|w - \mu_L|}{b} dw = \int_{-\infty}^{\mu_L} \mathcal{N}(w; \mu_G, \sigma^2) \frac{-(w - \mu_L)}{b} dw + \int_{\mu_L}^{\infty} \mathcal{N}(w; \mu_G, \sigma^2) \frac{w - \mu_L}{b} dw \tag{4.38}$$

$$= 2 \int_{\mu_L}^{\infty} \mathcal{N}(w; \mu_G, \sigma^2) \frac{w - \mu_L}{b} dw - \int_{-\infty}^{\infty} \mathcal{N}(w; \mu_G, \sigma^2) \frac{w - \mu_L}{b} dw \tag{4.39}$$

$$= 2 \int_{\mu_L}^{\infty} \mathcal{N}(w; \mu_G, \sigma^2) \frac{w - \mu_L}{b} dw - \frac{\mu_G - \mu_L}{b} \tag{4.40}$$

where Equation 4.39 follows from Equation 4.38 since:

$$\int_{-\infty}^{\mu_L} -x + \int_{\mu_L}^{-\infty} x = \int_{-\infty}^{\mu_L} -x + \int_{\mu_L}^{\infty} x + \left(\int_{-\infty}^{\infty} x - \int_{-\infty}^{\infty} x \right) \tag{4.41}$$

$$= \int_{\infty}^{\mu_L} -x + \int_{\mu_L}^{-\infty} x + \left(\int_{-\infty}^{\mu_L} x + \int_{\mu_L}^{\infty} x - \int_{-\infty}^{\infty} x \right) \tag{4.42}$$

$$= 2 \int_{\mu_L}^{-\infty} x - \int_{-\infty}^{\infty} x \tag{4.43}$$

Furthermore, $2\int_{\mu_L}^{\infty}\mathcal{N}(w;\mu_G,\sigma^2)\frac{w-\mu_L}{b}dw$ can be rewritten as;

$$\frac{1}{b}\left[2\int_{\mu_L}^{\infty} w\ \mathcal{N}(w;\mu_G,\sigma^2)dw - 2\int_{\mu_L}^{\infty}\mu_L\ \mathcal{N}(w;\mu_G,\sigma^2)dw\right] = \tag{4.44}$$

$$\frac{1}{b}\left[2\int_{\mu_L}^{\infty} w\ \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(w-\mu_G)^2}{2\sigma^2}}dw - 2\mu_L(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L))\right] = \tag{4.45}$$

$$\frac{1}{b}\left[2\int_{\mu_L}^{\infty}\ \frac{(w-\mu_G)}{\sqrt{2\pi\sigma^2}}e^{-\frac{(w-\mu_G)^2}{2\sigma^2}}dw + 2\int_{\mu_L}^{\infty}\ \frac{\mu_G}{\sqrt{2\pi\sigma^2}}e^{-\frac{(w-\mu_G)^2}{2\sigma^2}}dw - 2\mu_L(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L))\right] = \tag{4.46}$$

$$\frac{1}{b}\left[2\int_{\frac{(\mu_L-\mu_G)^2}{2\sigma^2}}^{\infty}\ \frac{\sigma}{\sqrt{2\pi}}e^{-y}dy + 2\mu_G(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L)) - 2\mu_L(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L))\right] \tag{4.47}$$

$$\frac{1}{b}\left[2\frac{\sigma}{\sqrt{2\pi}}(-0-(-e^{-\frac{(\mu_L-\mu_G)^2}{2\sigma^2}})) + 2(\mu_G-\mu_L)(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L))\right] \tag{4.48}$$

$$\frac{1}{b}\left[\frac{2\sigma}{\sqrt{2\pi}}e^{-\frac{(\mu_L-\mu_G)^2}{2\sigma^2}} + 2(\mu_G-\mu_L)(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L))\right] \tag{4.49}$$

where CDF denotes the cumulative density function, which can also be expressed in terms of the error function $\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(w) = \frac{1}{2} + \frac{1}{2}\text{erf}(\frac{w-\mu_G}{\sigma\sqrt{(2)}})$. Plugging the expression of Equation 4.49 back into Equation 4.40 and subsequently Equation 4.40 into Equation 4.37 yields the following expression for the KL-divergence:

$$KL(q(w)||p(w)) = -\mathcal{H}(q(w)) - \log(\frac{1}{2b}) - \tag{4.50}$$

$$\left(\frac{1}{b}\left[\frac{2\sigma}{\sqrt{2\pi}}e^{-\frac{(\mu_L-\mu_G)^2}{2\sigma^2}} + 2(\mu_G-\mu_L)(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L))\right] - \frac{\mu_G-\mu_L}{b}\right) \tag{4.51}$$

$$= -\mathcal{H}(q(w)) + \log(2b) + \tag{4.52}$$

$$\frac{1}{b}\left(\frac{2\sigma}{\sqrt{2\pi}}e^{-\frac{(\mu_L-\mu_G)^2}{2\sigma^2}} + 2(\mu_G-\mu_L)(1-\text{CDF}_{\mathcal{N}(w,\mu_G,\sigma^2)}(\mu_L)) - (\mu_G-\mu_L)\right) \tag{4.53}$$

MXnet does not have the CDF function or the error function implemented. The error function can be approximated by [64]:

$$erf(x) = \text{sgn}(x)\sqrt{1-e^{-x^2\frac{\frac{4}{\pi}+ax^2}{1+ax^2}}} \tag{4.54}$$

with $a \approx 0.147$ being a constant. In practice during experiments it was found that the addition of a numerical stability constant was necessary to prevent overflow of

gradients:

$$erf(x) = \text{sgn}(x)\sqrt{1 + \alpha - e^{-x^2 \frac{\frac{4}{\pi} + \alpha x^2}{1 + \alpha x^2}}} \tag{4.55}$$

with α a very small positive constant.

4.5 Implementation details

4.5.1 KL-warmup

With all the variational method used for compression, a KL-warmup was applied. The idea of a KL warmup was first introduced by [57], in which the KL divergence in the ELBO is scaled by a factor α:

$$\mathcal{L} = -\mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p(D|\boldsymbol{w}))] + \alpha KL[q_{\boldsymbol{\phi}}(\boldsymbol{w})||p(\boldsymbol{w})] \tag{4.56}$$

where *alpha* is increased linearly from 0 to 1 during an initial N epochs. Throughout the experiments in this work, the warmup was done over $N = 100$ epochs. The warmup slowly interpolates between an objective which has a delta-distribution as a solution, and the ELBO as an objective. The warmup can be seen as an optimization heuristic, since the objective still corresponds to performing variational inference when *alpha* is 1 following the warmup period. While the KL-warmup technique was proposed for ladder variational auto-encoders, the need for it might be more pressing when compressing pre-trained models since the original weights where trained with a significantly different objective than the ELBO. In addition to this we found we found that the gradients due to KL-divergence to dominate the total gradient during early training stages. A KL-warmup thus might provide more stable training.

4.5.2 Variance constraints and initialization

In [44] it was found that constraining the variance of the variational distribution improved training. In contrast with the KL-warmup, even during the final stages, the ELBO is adapted by this, and the constraints result in a bias and looser bound. The variance constraints where used for some of the variational inference compression methods, but their impact on the stability of training was not significant. To stabilize training at the early stages, the variances or scales of the variational distribution where initialized to be roughly e^{-6} at the start of training. The small variance of the variational distribution, results in the network first being mostly deterministic, and

slowly moving into a Bayesian regime.

4.6 Pruning

After having induced sparsity in a network through training, a suitable pruning criterion needs to be selected. A readily available heuristic is to simply compute absolute value of the weights, sort the weights and then select a pruning threshold according to pre-specified required amount percentage of pruning. Several other pruning heuristics where employed:

- In the case of Group Lasso, the group penalty of a group can be used as the pruning criterion. These penalties might be reweighted by dividing them by the number of weights in a group.
- In case of variational dropout with a Normal-Jeffreys prior, the implicit dropout rate for a group $\alpha = \frac{\sigma^2}{\mu^2}$ can be used to select a pruning threshold.
- In [8], it was suggest to use the *signal-to-noise* ratio $\frac{|\mu|}{\sigma}$ as a pruning threshold after variational inference with a Gaussian variational distribution and Gaussian prior.
- For Bayesian networks the absolute value of a weight divided by the entropy of the posterior distribution for that weight: e.g. $\frac{|\mu|}{\log(2be)}$ for a Laplace(μ, s) distribution and $\frac{2|\mu|}{\log(2\pi\sigma^2)}$ for Gaussian posterior.
- One attractive pruning criterion might be the KL-divergence for a certain weights. It can be motivated by the fact that the gradients for each weight or group are determined by gradients due to a data-loss term, and a term due to the KL-divergence. If a particular weight/group is unimportant for the data-loss, the gradient due to the data-loss will be small, and the KL-divergence will force that weight's variational distribution to be as close to the prior as possible resulting in a low KL-divergence for the weight. The same reasoning can be applied to structured sparsity pruning.

When pruning away complete neurons, the next layer should be adapted as well since one input dimension to the next layer is now permanently zero. Thus all the weights in the next layer connected to the pruned neuron are also pruned away.

Chapter 5

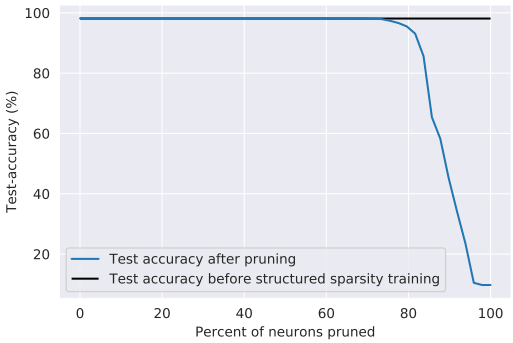
Experiments

In this chapter we describe the results of the experiments carried out. First the results of the compression methods for inducing structured sparsity in a network are described; Group Lasso, variational inference with a Normal-Jeffrey's prior, Bayesian Group Lasso and some of the properties of the compression methods are further explored.

Next we describe the experimental results for the unstructured sparsity variational inference compression methods; Gaussian priors combined with a Gaussian variational distribution, Laplace priors combined with a Gaussian variational distribution as well as a Laplace variational distribution. Then some exploratory experiments on using variational inference compression for transfer-learning are presented. Finally we show the results of training Bayesian networks on datasets with randomized labels.

To allow easy comparison, and due to computational limitations, the compression methods are mostly evaluated on the MNIST dataset [39], consisting of 70000 grey-scale images of handwritten digits with a size of 28 by 28 pixels. Furthermore, the main model used is LeNet-300-100, a 2 hidden layer neural network, with 300 neurons in the first layer, 100 in the second layer, and which outputs a 10 dimension vector to the last softmax-layer. The softmax layer is also regularized, although there is very little pruning in the final softmax layer compared to the earlier layers for unstructured sparsity methods, and no pruning for the structured sparsity methods.

Figure 5.1: Test accuracy vs percentage of neurons pruned.



5.1 Structured sparsity through group lasso regularization

Because for LeNet-300-100 there are no pre-trained parameters available the model is first trained on MNIST to convergence, where it had an test-error of 1.88 %, which is approximately the same as the test-error reported by [43] (1.89%). For a fair comparison between compressed models and the benchmark model, it is important that the benchmark model has been regularized. This is implemented by training the bench-mark with ℓ_2 regularization. This model is then used as a benchmark throughout further experiments.

The model was subsequently trained with Group Lasso penalties for 150 epochs. The λ parameter giving the strength of the group lasso penalties in $\lambda \sum_g |\boldsymbol{w}_g|_2^2$ and was tuned extensively over a range of [0.001,500], in parallel with tuning of the learning rate and found to be optimal for $lambda = 0.05$. The optimal learning rate was roughly the default value of 0.001. After the network had been trained for 150 epochs, the group-lasso penalties for each neuron are computed and sorted. A threshold was then determined according to the required percentage of neurons to prune.Figure 5.1 shows the test-accuracy after pruning for a range of percentages of neurons pruned. The figure presents a model which has been trained to convergence, after which it is copied 50 times, and the figure shows the accuracy after pruning for different pruning thresholds.

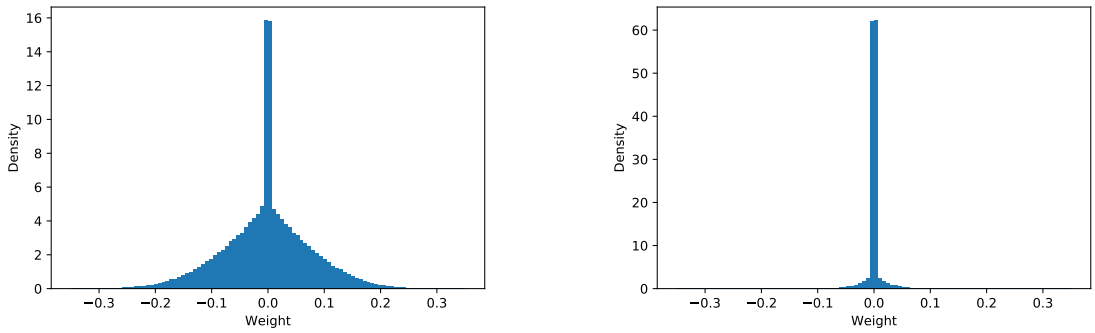
When 68 % of the neurons is pruned, the accuracy of 98.18 (error of 1.82%) % is still better than the benchmark model. As a sanity check the model size after pruning 68% is extracted, which gives 82 neurons for the first layer, 40 for the second, and 10 for the final softmax layer and a model of that size is trained from a random initialization. This model achieved a 97.39 % accuracy. The result that it is seems easier to train

a larger network and then compress it to a smaller network than training the smaller network from a random initalization has earlier been found by [23] [26] [5]. A possible explantation of this phenomenon called the *lottery-ticket hypothesis* was offered in [16] where it was argued that any large network that trains succesfully contains a subnetwork that is initalized such that if it where trained in isolation, it would reach the same accuracy as the larger network in the same number of epochs. This results in the conjecture that it is easier to train large networks since they have more possible combinations of such subnetworks.

Figure 5.2 shows the weight distributions before and after training with group lasso penalties, when no explicit pruning is yet applied. Before training with a group lasso penalty, there is already a peak at zero in the weight distribution. Because the digits of MNIST are centered in the image, the border pixels are almost completely irrelevant, and the weights corresponding to those pixels go to zero even with only normal l-2 regularization. The same behavior occurs with Group Lasso, which can be seen in Figure 5.5. The figure shows the heatmaps of the weight matrixes of layer 1 and 2 after group lasso training. The elimination of a significant amount neurons is clearly visible as rows with weights all being zero. On closer inspection, there is also a periodic pattern with period 28 in the columns of the first layer.

Since the input data vector is a flattened version of the 28 by 28 MNIST image, the periodic pattern corresponds to the boundaries of the images where pixels always have the value 0. It is thus the same effect as described above and shown in Figure 5.2. This effect can be used to, in addition to pruning of the neurons, also prune weights associated to the boundary of the MNIST images. In general, the features of which the associated the weights have become zero due to sparsity induced regularization can be discarded resulting in automatic relevance determination. An illustration of that process is given in Figure 5.3 and subplots.Figure 5.3a shows the effect of pruning away weights associated with particular input features after group lasso regularized training. With no accuracy drop, 44 % of the input features can be discarded. That Group Lasso also manages to prune individual weights next to larger structures is a desirable property of the method.

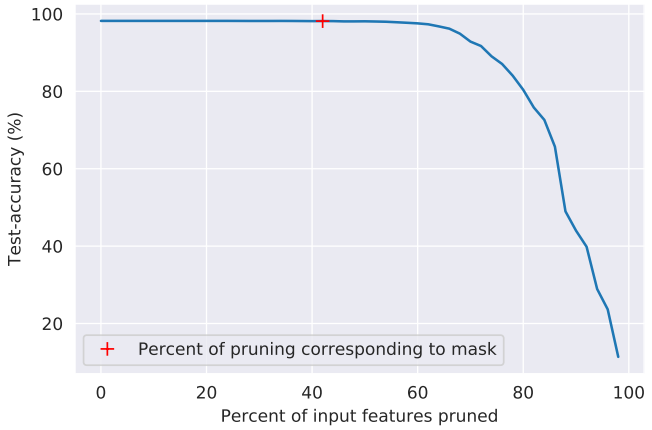
Figure 5.3b shows orginal MNIST images, and the same images with a mask consisting of the input features that have been pruned away superimposed on the images. Compared to a factorized Laplace prior on the weights, having solely a group lasso prior on the neurons will prune away less input features. For a weight w_i belonging to neuron with N input features, the gradient with respect to the group lasso penalty is $w_i/(\sqrt{\sum_j^N w_j^2})$, and even if the individual weight w_i is irrelevant, if the rest of the neuron weights are of significant magnitude, w_i will go slowly to zero. A possible solu-



(a) Before group lasso regularized training

(b) After group lasso regularized training

Figure 5.2: Distributions of weights before and after structured sparsity.



(a) Pruning input features

(b) Original and masked MNIST images

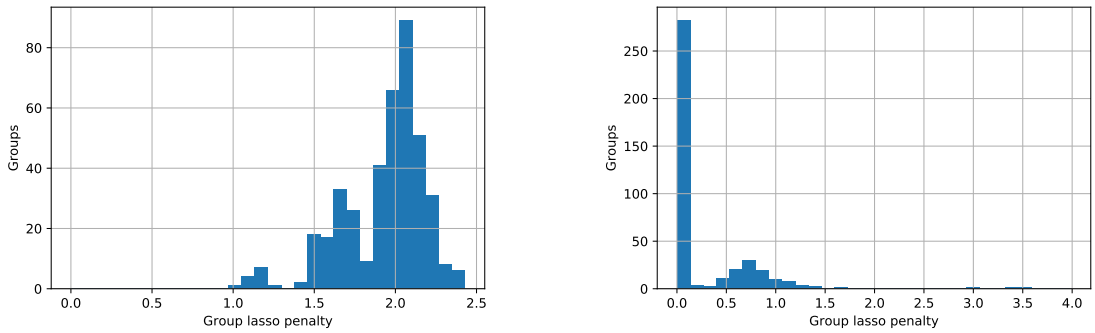
Figure 5.3

tion to this could be to introduce additional ℓ_1 penalties on individual weights, or have group-lasso penalties be defined over both the neurons as well as the input features to a given layer. Particularly the first layer of a fully connected neural network might be able to perform extensive automatic relevance determination of the features.

Figure 5.4 shows a frequency plot of the group lasso penalties, which shows how group lasso training has resulted in two clearly seperated groups after group lasso training.

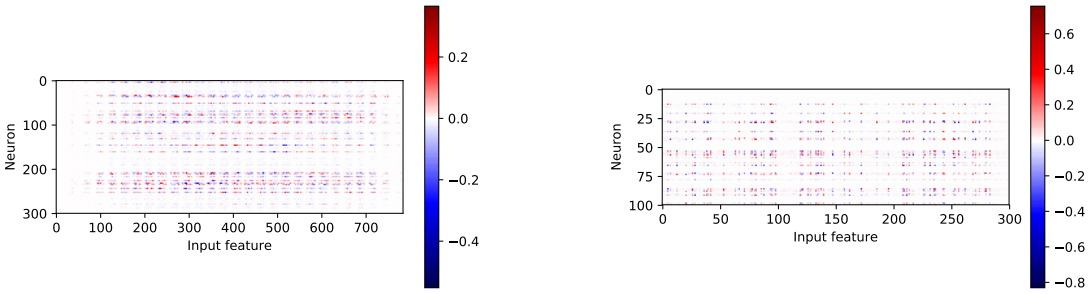
Structured sparsity for Convolutional Neural Networks

Figure 5.6 shows how group lasso training can also be used to prune away complete filters from CNN's. The convolutional neural network used here has 2 convolutional layers with 20 filters in the first layer, 50 filters in the second layer, with each layer being followed by a maxpool-layer and finally a fully connected layer with 512 neurons.



(a) Before group lasso regularized training (b) After group lasso regularized training

Figure 5.4: Frequency plot of group lasso penalties for structured sparsity



(a) Layer 1 (b) Layer 2

Figure 5.5: Heat maps of weights of first two layers after Group Lasso training

Figure 5.6: Group lasso training with CNN's, exploratory experiment

(a) First 6 filters of first layer before group lasso training. Test accuracy: 0.986

(b) First 6 filters of first layer after group lasso training5.6a. Test accuracy: 0.985

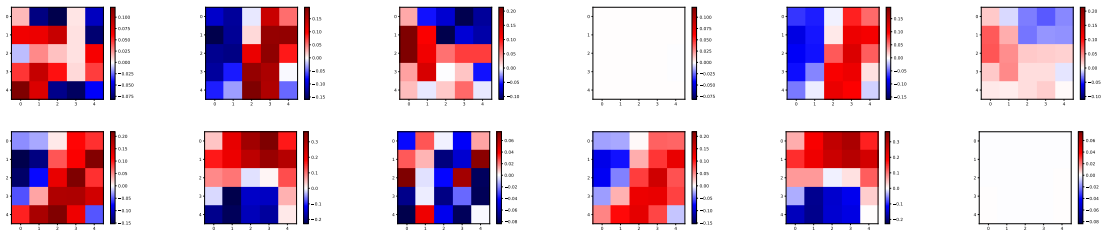


Figure 5.6 presents the first 6 filters of the first layer before and after the group lasso training. It is clear how the weights 2 of the 6 filters have become effectively zero and can be pruned away with little consequences. Due to computational limitations the choice was made to carry further experiments out only on fully connected neural networks.

5.2 Variational inference with Normal-Jeffrey's prior

Again the pre-trained Le-Net-300-100 with an test-error of 1.88% was used as a benchmark model. The network was initialized with the weights of the benchmark model and trained for 150 epochs, with 1 Monte Carlo sample per batch and using the expression for approximate KL-divergence as discussed in Subsection 4.3.1. A KL-warmup was done over 100 epochs, as described in Subsection 4.5.1. The KL-warmup was found to be important for stable training of the network. Furthermore the variances of the variational distributions where initialized to very low values on the order of 10^{-4}. Initializing the variances to higher values resulted in highly unstable training and worse performance. It was observed that during the first epochs, the variance parameters of the variational distribution changed much faster than the mean parameters. The very first training batch was often unstable, and some tuning of the learning-rate was required. A learning-rate scheduler might be able to ameliorate some of these problems. However a learning-rate scheduler was not yet compatible with MXNet's Adam optimizer.

A good heuristic to achieve a better compression turned out to be early stopping; not training to convergence. When training the network convergence, the test-accuracy dropped below the benchmark, while there was little extra sparsity in the network. The resulting compression is shown in Figure 5.7, which plots the test-accuracy a function of the percentage of neurons pruned. While testing the model, the means of the weights

Figure 5.7: Bayesian compression Normal-Jeffreys prior.

where used for a deterministic forward pass trough the network, which usually resulted in slightly higher performance than taking just one Monte Carlo sample. It can be seen that the method outperforms group lasso in terms of compression while keeping the accuracy the same.

5.3 Bayesian Group Lasso

The KL-divergence is sampled using Monte Carlo samples, with the number of samples being varied along experiments, with slight gains for a higher number of samples. For the data loss only one Monte Carlo sample was used. The local reparameterization trick was used, and thus to compute the log-likelihood the activations where sampled rather than the weights. To be able to compute the KL-divergence, the weights then needed to be sampled separately. Figure 5.8a shows the performance of Bayesian Group Lasso and the distribution of the the pruning criterion for each group: $\left|\frac{\boldsymbol{\mu}^{(g)}}{\boldsymbol{\sigma}^{(g)}}\right|_2$. The performance of Bayesian Group Lasso can seen to be on par with the results of variational inference with a Normal-Jeffreys prior.

The hyperparameters where tuned extensively, and the scale parameter b as defined in Equation 4.11 was found to be optimal for $b = 0.025$, with comparatively good performance for $b \in [0.005, 1]$. During testing of high values for the scale parameter, $b \in [5, 100]$, a recurring problem was a very high variance of weights which where close to zero. Without constraining the variance, the variance would go to very high values. Since these weights are approximately zero, they do not contribute much to the prediction put out by the network, and likewise contribute little to the dataloss. Therefore the variance is fairly unconstrained. This does however resulted in a much higher variance of the losses, with training very suddenly becoming unstable once enough weights took on mean zero and a high variance. Following the approach de-

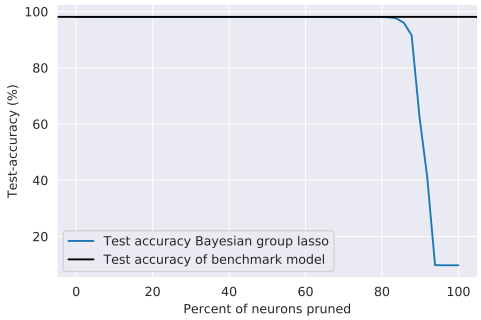
scribed in Subsection 4.5.2, the variance of the variational distribution was clipped. However when using priors with a lower scale this effect disappeared, and there was no difference between performance with and without variance clipping. The absence of the need to clip the variance can be seen as an advantage of Bayesian Group Lasso.

Figure 5.9a presents the mean values of the weight matrixes W of the first and second layer of a network which has been trained to convergence with Bayesian Group Lasso as heatmaps. Each row corresponds to the weights of 1 neurons, whereas the columns corresponds to the weights connect to 1 input feature. It is clearly visible that a significant number of rows are virtually zeroed out. It also shows how if a neuron i in the first layer is effectively zeroed out, the weights connected to this neuron i are also set to zero.

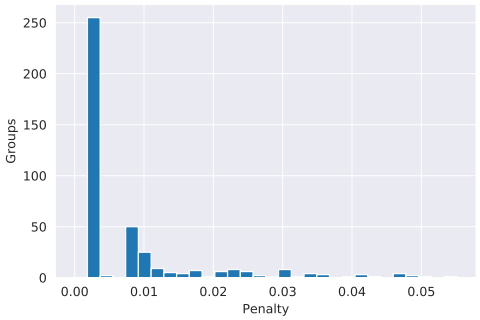
Figure 5.9b shows heatmaps of the corresponding variances. Some rows (neurons) with weights of significant magnitudes, have quite some difference between the variances of their individual weights. After pruning this network, these differences in variances could be used for quantization of weights by making decision of quantization levels based on the variance of that particular weight. Quantization using posterior uncertainties has been done in [43], where gave significant increases in compression rates. However, although it is possible to train with 16-bits floats, or have certain layers be of a lower precision than others, it is currently not possible to have mixed precision within one layer within the major deep learning framworks.

During experiments we observed that weights which went during Bayesian Group Lasso training to zero, mostly stayed zero and would be pruned at the end of training. This suggests a n adaption of the training method in which pruning is done iteratively. After every 5-th epoch, the network was pruned as much as possible, with an allowed drop in accuracy of 0.1 %. After pruning the neurons, the parameters were copied over to a smaller architecture, and training continued. Iterative pruning was not found to give any significant improvement in compression. However it did slightly decrease training time per epoch, because the architecture in later stages was significantly smaller than the uncompressed model. However, more exploration is needed to further establish the influence of iterative pruning on the training speed.

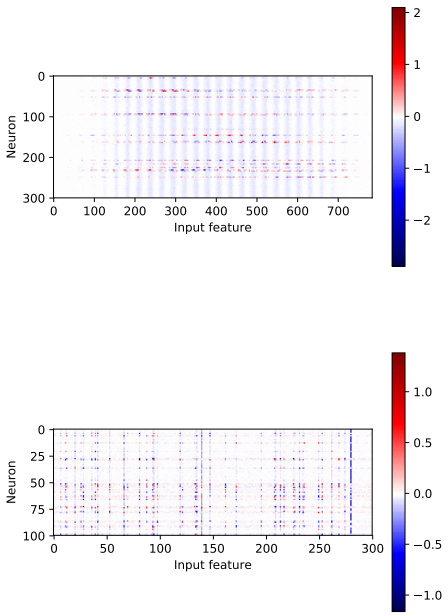
Figure 5.8: Bayesian Group Lasso with learning-rate:0.001, priorscale:0.025.



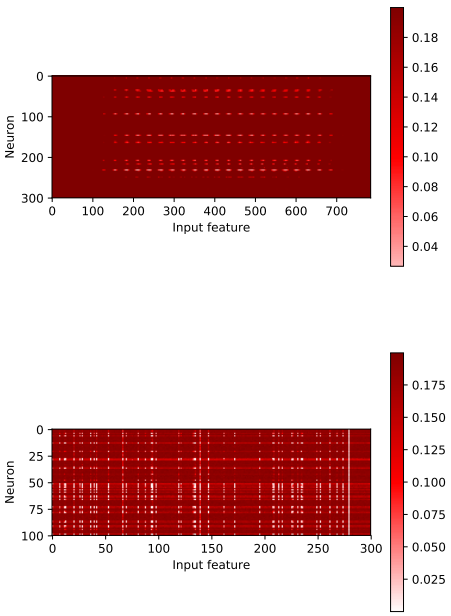
(a) Accuracy vs pruning



(b) Distribution of pruning penalties



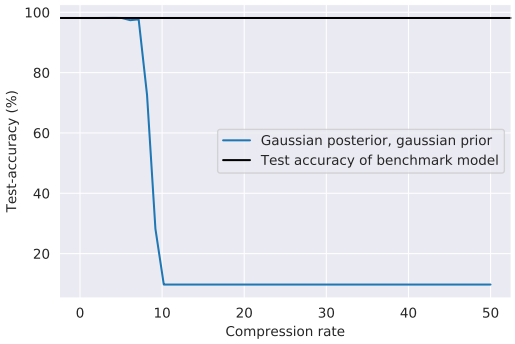
(a) Heatmaps of the means μ of the variational distribution $\mathcal{N}(\mu, \sigma^2)$ for the first and second layer.



(b) Heatmaps of the variances σ^2 of the variational distribution $\mathcal{N}(\mu, \sigma^2)$ for the first and second layer.

Figure 5.9

Figure 5.10: Variational inference with Gaussian variational distribution and Gaussian prior.



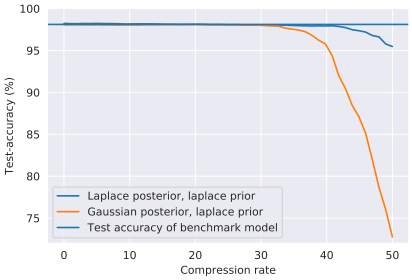
5.4 Random sparsity for compression using variational inference

5.4.1 Gaussian variational distribution, Gaussian prior

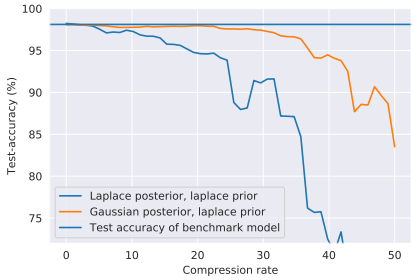
Figure 5.10 shows the result of compressing the Le-Net-300-100 network using a Gaussian variational distribution and Gaussian prior. Compression rate is here measured as $\alpha(M, M^*) = \frac{a}{a*}$, as defined in Section 4.1. As a pruning criterion the signal-to-noise $\frac{|\mu|}{\sigma}$ ratio as proposed by [22] [8] was used. The variance hyperparameter of the prior was tuned and found to be optimal for $\sigma^2 = 10$ for the LeNet-300-100 network. The KL-divergence between the posterior and prior was computed analytically for each minibatch while 1 Monte-Carlo sample was used to sample the data-loss term. [8] tested a 2 layer fully connected bayesian neural network with 1200 neurons per layer which was trained on MNIST. There a compression rate of 4 without any accuracy drop compared to their uncompressed model was reported, and a compression rate of 20 with 0.05% accuracy drop. The compression performance we report here for the Gaussian variational distribution and Gaussian prior is likely worse because of the smaller size of the LeNet-300-100 network. The network used in [8] has roughly 10 times more parameters, and a 20 compression rate corresponds to still half as much as the amount of parameters in the original LeNet-300-100 network.

5.4.2 Laplace priors

A Laplace prior was combined with a factorized Gaussian variational distribution and a Laplace posterior. For the Laplace posterior with Laplace prior an analytical expression for the KL-divergence is available (see Subsection 4.4.2). The KL-divergence



(a) Pruning criterion: squared weights divided by scale of posterior.



(b) Pruning criterion: absolute value of weights .

Figure 5.11: Compression rate vs test accuracy. Scale of prior is 0.2. Prune criterion using squared weights divided by scale or variance of posterior. Benchmark accuracy is the horizontal line.

between the Gaussian posterior and Laplace prior can either be computed using Monte Carlo samples or be analytically approximated as shown in Subsection 4.4.2. Figure 5.11 shows a comparison of performing variational inference with a Laplace posterior and a Gaussian posterior, where in both cases the prior is a factorized Laplace distribution over all the weights. The plot shows the accuracy for different pruning criteria. Both methods where found to be optimal for a prior scale of $b = 0.2$. The compression performance can differ greatly depending on the choice of pruning criterion. The pruning criterion which worked best was the square of each weights divided by the scale for the Laplace variational distribution and the variance for the Gaussian variational distribution. As shown in Figure 5.11a, the Laplace posterior with a Laplace prior outperforms the Gaussian posterior. One advantage of using the Gaussian posterior is that the local reparameterization trick can be used. Here no consideration has been given to the speed of the convergence of the two methods, and both were trained for 150 epochs. Further exploration of the degree to which the local reparameterization trick could speed up the training would be needed to examine if there is a trade-off between a better compression and a higher training speed.

If one Monte Carlo sample of a parameter in the neural network with mean, μ a weight w is sampled, the Monte Carlo sample estimate of the cross entropy between that weight's variational distribution and the zero-mean Laplace prior is: $-\log(\frac{1}{2b}e^{-\frac{|w|}{b}})$. This gives as the gradient of the KL-term for the mean of the posterior of that particular parameter:

$$\frac{\partial H(q||p)}{\partial \mu} = \frac{\partial}{\partial \mu}\Big(\log(2b) + \frac{|w|}{b}\Big) = \frac{\text{sgn}(w)}{b} \tag{5.1}$$

which is independent of the magnitude of the mean of that particular parameter. This is particiculary problematic when the goal is to bring weights to zero, since if a

weight is close to zero with a variance which is significantly greater than the mean, the gradients for that weight will be approximately distributed as being $\frac{w}{b}$ with chance 0.5 and $\frac{-w}{b}$ with chance 0.5. Note that this is a particular pathology of using a Laplace prior with Monte Carlo sampling. With a Gaussian prior the gradient is dependent on the magnitude of the weight sample, and thus the gradient decreases in magnitude as a weight gets close to zero. Figure 5.12 presents an exploration of the behavior of the gradients in the LeNet-300-100 network over the course of 200 compression training epochs, where the KL-divergence between the Laplace prior and variational Gaussian posterior has been sampled. The gradients due to the KL-divergence are several orders of magnitude larger than those due to the data-loss term. Furthermore the variance of gradients due to the KL-divergence are an order of magnitude larger than their average magnitude, while for gradients due to the data loss, the variance is much smaller than the magnitude.

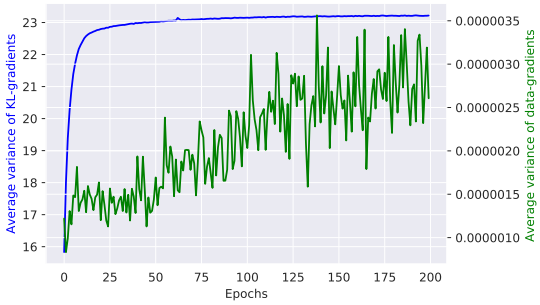
For a compression training run of 200 epochs, each epoch the KL-divergence was sampled with 500 samples, and analytically approximated. The result is shown in Figure 5.13a from which it can be seen that the analytical approximation very closely follows the sampled KL-divergence. Figure 5.13bshows the difference between the sampled KL-divergence and the analytical approximation. It appears that there is a small bias during the early stages of training. This might be attributed to the low variance initialization, which makes the error function part of the analytical KL-divergence;

$$\mu_G \operatorname{erf}\Big(\frac{\mu_G}{\sigma\sqrt{(2)}}\Big) \tag{5.2}$$

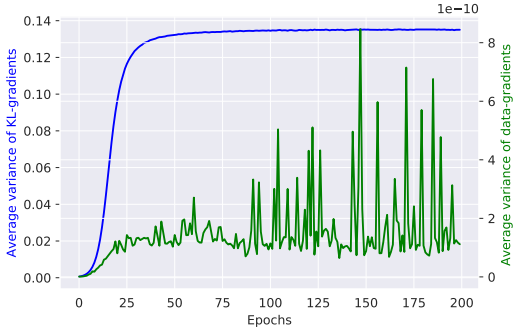
operate in the regime where the erf has problems with rounding errors since the term in Equation 5.2 becomes either very negative or very positive depending on the sign of μ_G. However the bias is still relatively small and quickly decreases, which will make its impact limited.

5.5 Student-teacher training combined with structured sparsity

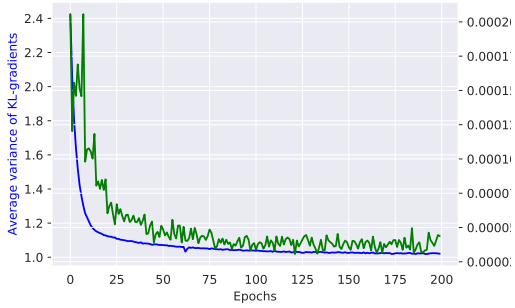
Student-teacher training [26] and compression trough group lasso regularization can be combined to potentially improve upon the performance of both individual methods. In the set-up used here, two copies of a large network are made, of which one becomes the teacher, and one the student. The teacher network outputs a categorical distribution over the 10 classes of MNIST. The student network is trained with a group lasso



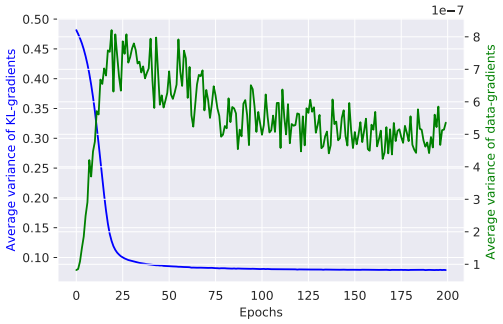
(a) Average variances of gradients for the μ parameters of the factorized posterior: $\mathcal{N}(\mu, \sigma^2)$ as a function of the number of epochs.



(b) Average variances of gradients for the $\log(\sigma^2)$ parameters of the factorized posterior: $\mathcal{N}(\mu, \sigma^2)$ as a function of the number of epochs.

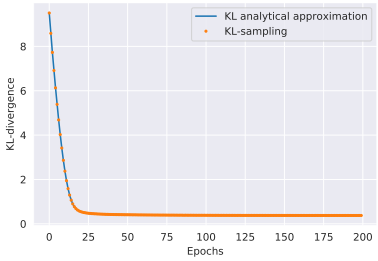


(c) Means of the magnitude of the gradients for the μ parameters of the factorized variational distribution: $\mathcal{N}(\mu, \sigma^2)$ as a function of the number of epochs.

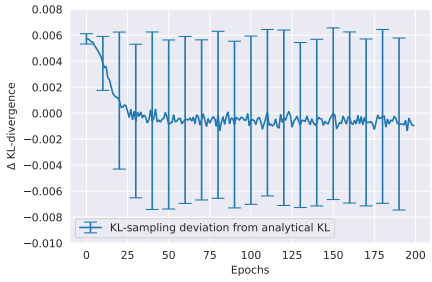


(d) Means of the magnitude of the gradients for the σ^2 parameters of the factorized variational distribution: $\mathcal{N}(\mu, \sigma^2)$ as a function of the number of epochs.

Figure 5.12: Exploration of variances of the gradients for variational inference with Gaussian variational distribution and a Laplace$(0, 0.2)$ prior



(a) Analytical approximation for the KL-divergence from Subsection 4.4.3 and Monte Carlo sampling of the KL-divergence. The lines are nearly identical.



(b) Difference between the sample mean $\bar{x}$ of the KL-divergence and the analytical approximation. Erorrbars are 1 standard deviation of the sampled KL-divergence

Figure 5.13

penalty. In addition to the group lasso penalty and the data-loss, a cross-entropy loss between the output of a softened softmax teacher and a softened softmax of the student network is added to the loss [26]. A softened softmax is one in which the temperature parameter is set higher:

$$\sigma(\boldsymbol{x})_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}} \tag{5.3}$$

In [26] it was argued that for a given teacher model, there is information about the function which is implemented by the network in the ratios of small probabilities of the categorical distribution which have been outputted by the softmax. Adding a loss term to a student model which depends on the categorical distribution of the teacher would help the student to mimick the function implemented by the teacher network.

Here the intuition is that when doing compression on a pretrained model, the goal is to decrease size while keeping the function implemented by the network the same as the uncompressed model. Adding a loss-term which forces the compressed model to mimick the teacher-networks categorical distribution might help the compressed model retain this function. The loss of the student network is given by:

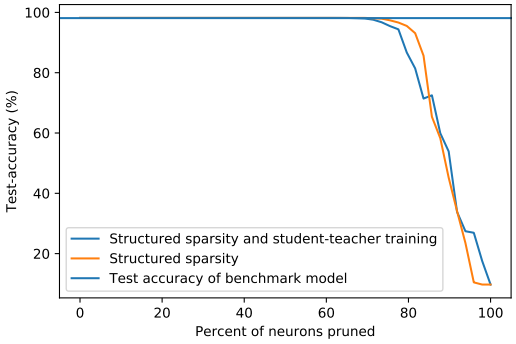
$$L = \lambda L_{teacher} + (1 - \lambda)(L_{student} + L_{grouplasso}) \tag{5.4}$$

the influence of the teacher network causes the optimal group lasso strength to be different, and thus needs new tuning. The group-lasso-strength factor should be multiplied by $\frac{1}{1-\lambda}$ to ensure the group lasso penalty stays roughly the same strength.

Figure 5.14 shows a comparison of compression using structured sparsity against using structured sparsity and combining it with student-teacher training. The addition of student teacher training results in less good compression than doing group lasso alone. The plots shows the test accuracy as a function of the number of the neurons pruned away, after taking from both methods the model after the final epoch of training. The training of the combination of group lasso with student-teacher training was found to be very sensitive to the temperature and strength of the group lasso parameter. It might be expected that the student-teacher training does not help the smaller network significantly until the accuracy of the smaller network drops below that of the larger network. When the accuracy of the smaller network is roughly the same as the teacher network, the function implemented by both networks is not likely to be significantly different, and thus the teacher-term will not have much effect. However further experiments are needed to examine this.

As discussed in Section 3.3, student teacher training can also be done with a Bayesian

Figure 5.14: Student teacher combined with Structured sparsity compression vs structured sparsity on its own



teacher network. In [4], a teacher network trained by Stochastic Gradient Langevin dynamics [62] provided the Monte Carlo samples of the predictive posterior distribution which where used to train a student network. This strategy is however not restricted to MCMC methods. A possible other strategy would be to take samples from the predictive posterior for the categorical distribution of a network trained by variational inference.

5.6 Compression for Transfer learning

Transfer learning attempts to increase the performance on a new *target* task through transfering knowledge from a previously learned *source* task [59]. A Le-Net-300-100 model was trained on MNIST to a test error of 1.88%, and after convergence trained on a variationMNIST dataset. VariationMNIST is a set of datasets in which different data transformations have been applied to the original MNIST dataset [39]. The dataset used here is MNIST images which superimposed on a background consisting of uniform grey-scale noise. Some examples are given in Figure 5.15. 5000 images where used as a training-set, with the test constituting 60000 images. Two different methods of transfer learning were compared, in the first, the LeNet model was directly trained on the transfer dataset, whereas in the second method, the model was first compressed by performing variational inference with a Laplace posterior and Laplace prior. After the variational inference training, the weights where fixed, and weights where pruned over a range of compression rates. A model with a compression rate of 40 (97.5% of the weights pruned), is then used for transfer learning.

The compressed model is the same as the one shown for compression rate 40 in Figure 5.11a. The reasoning for using a the model with compression rate 40 is that

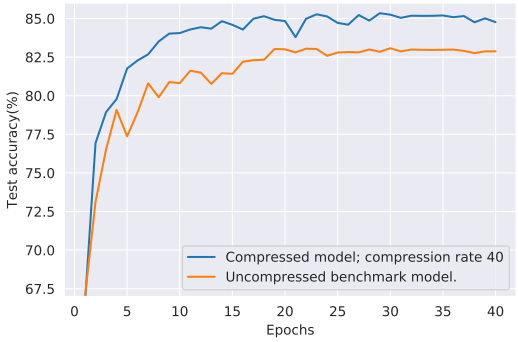
Figure 5.15: Examples from dataset with random noise backgrounds.

this is the point up to which the model could be compressed before the test-accuracy dropped. All the weights from the Bayesian neural network are then used to initalize a deterministic network the same size as the benchmark model. The weights which where pruned and thus set to zero are now initialized with zero-mean Gaussian noise which has a small variance. As shown in Figure 5.16 the compressed model achieves a significantly higher test-accuracy than the benchmark model. To make the plot more robust to variance in the test-accuracy, the test-accuracies where estimated by computing the test accuracy after 9 consecutive batches near the end of an epoch, and then taking the average of those test-accuracies.

To further explore the impact of compressing networks before transfer learning, a CNN was trained to 1% test-error on MNIST, which was then used as the benchmark CNN model. A copy of this model was then trained with strong $\ell - 1$ regularization and 6 models where extracted after different number of epochs (1,6,11,16,21 and 26 epochs). The uncompressed CNN-benchmark model and the 6 models where then all trained on a variationMNSIT dataset, in which the MNIST images have been rotated over an angle sampled uniformly from Uniform$(0, 2\pi)$ radians. In addition to this, the rotated digits have been superimposed on random patches of an arbitrary black-white image. Figure 5.21 shows some example images from the dataset. The transfer-learning was done for 30 epochs.

Figure 5.17 shows the resulting performance of the CNN's for different number of epochs trained with strong $\ell - 1$ regularization. transfer learning experiment using a CNN. All models eventually converge to the same accuracy. However there is a slight increase in test-accuracy in test-performance in the early stages of training for the compressed models. To further show the early stages of the transfer learning Figure 5.18 shows the test-accuracy on the full dataset after each batch. Both models were trained for 5 epochs, which was repeated 100 times for both the benchmark as the compressed models, and the average test-accuracy and standard deviation are shown. Although the compressed model consistently has a higher test-accuracy during the first 5 epochs, the difference in speed of convergence during the first epoch is especially striking.

Figure 5.16: **Transfer learning on variationMNIST images with noisy background.** Comparison of transfer learning performance of the uncompressed benchmark model and a compressed.



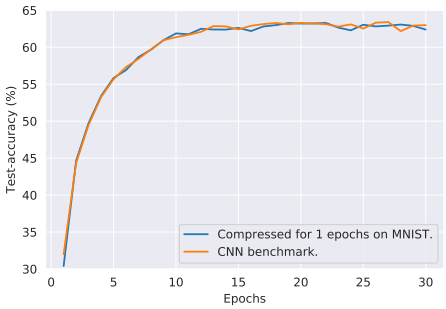
5.7 Random labels

In [67] it was shown that neural networks can easily achieve 0% training error on a dataset where the labels have been randomized. This suggests that the effective capacity of (non-Bayesian) neural networks is sufficient for memorizing the entire dataset. Since the labels are random, the test-accuracy is very low. In [49] it was experimentally found that variational Gaussian dropout was not able to fit randomized labels well, whereas in contrast non-variational networks could fit the random labels. Even when the non-variational network did have Bernoulli dropout (as proposed in [58] with p = 0.5), the non-variational networks where still able to fit the random labels. When a fully-connected network was trained on MNIST or a CNN on the CIFAR-10 dataset, if trained from random initialization, both of the models became completely sparse and gave an equal probability to any of the categories, resulting in a training error approximately equal to the test-error and hence a low generalization error. However when initializing the variation Gaussian dropout networks which had been previously trained in a deterministic neural network of the same size on random labels to a training error of 0%, variational Gaussian dropout was also able to fit the random labels. The authors argued that these results might imply that variational Gaussian dropout penalizes memorization of the data set and improves the generelization performance.

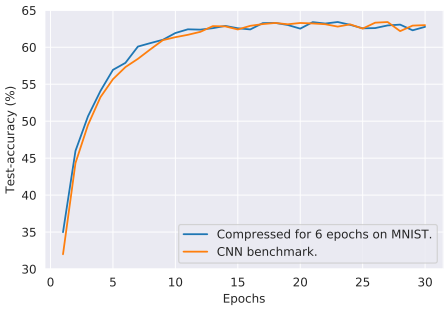
As shown in [29], and discussed in Subsection 4.3.1 the improper log-uniform prior as used in [49] however leads to an improper posterior in general. Here we perform variational inference with a Laplace variational distribution and (proper) Laplace prior on a dataset from the variationMNIST, in which MNIST images have been rotated and superimposed on random background images, as shown in figure 5.21.

The labels where randomized by sampling an integer from the discrete uniform distri-

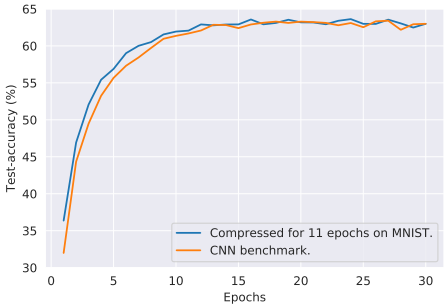
Figure 5.17: Transfer learning for CNN's on variationMNSIT, rotated images with a random background image



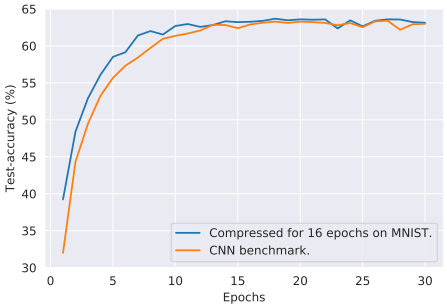
(a) Test-accuracy of CNN on variationMNIST with rotated digits with a random background. Trained with ℓ_1-regularization for 1 epoch on MNIST, after which the test-accuracy was 98.98%



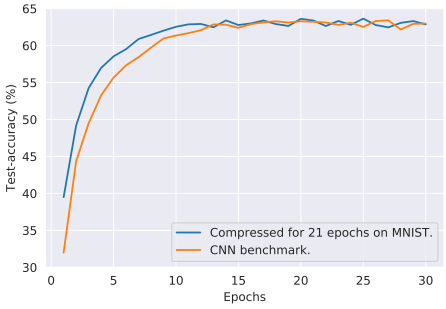
(b) Test-accuracy of CNN on variationMNIST with rotated digits with a random background. Trained with ℓ_1-regularization for 6 epoch on MNIST, after which the test-accuracy was 98.97%



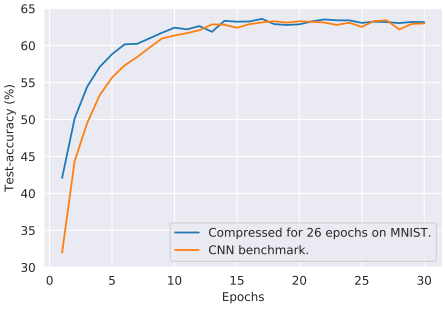
(c) Test-accuracy of CNN on variationMNIST with rotated digits with a random background. Trained with ℓ_1-regularization for 11 epoch on MNIST, after which the test-accuracy was 98.93%



(d) Test-accuracy of CNN on variationMNIST with rotated digits with a random background. Trained with ℓ_1-regularization for 16 epoch on MNIST, after which the test-accuracy was 98.92%

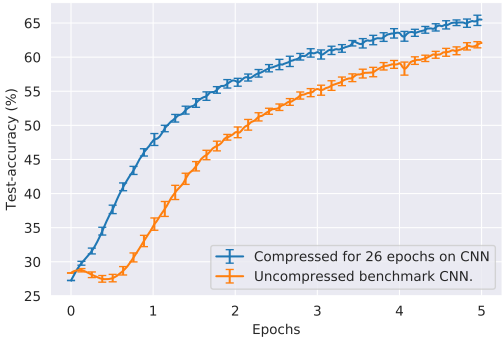


(e) Test-accuracy of CNN on variationMNIST with rotated digits with a random background. Trained with ℓ_1-regularization for 21 epoch on MNIST, after which the test-accuracy was 98.89%



(f) Test-accuracy on variationMNIST with rotated digits with a random background. Trained with ℓ_1-regularization for 26 epoch on MNIST, after which the test-accuracy was 98.84%

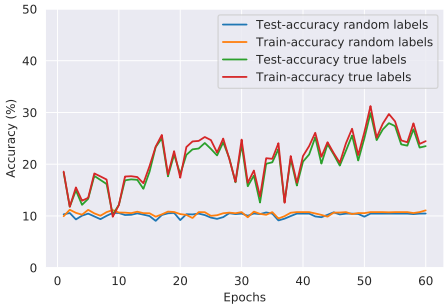
Figure 5.18: CNN first 5 epochs experiment with the CNN regularized for 26 epochs on MNIST, final accuracy being 98.84



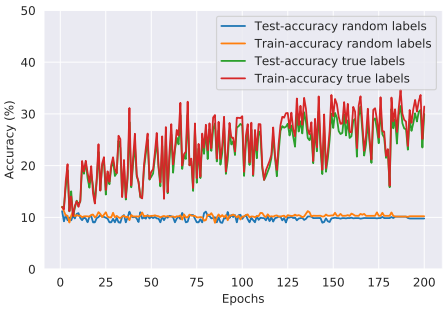
bution with possible values $\in [0, 9]$. Figure 5.19 shows that it does not seem possible to let the Bayesian networks with a Laplace variational distribution and Laplace prior fit random labels from a random initialization. This is in contrast with Figure 5.20, where a non-Bayesian network which was trained with $\ell - 2$ regularization still fits the training data eventually perfectly. To further explore whether the Bayesian networks are able to fit the random labels, 8 different Bayesian networks with different scales for the prior where trained on random labels. The results in Figure 5.22, show that it still does not seem possible for a wider range of prior scales to fit the random labels.

Figure 5.23 shows the training accuracy of a BNN, trained with VI using a Laplace prior and Laplace variational distribution, where the weights have been initalized with weights which were previously trained to 100% train-accuracy in a non-Bayesian network on random labels. To make fitting the random labels easier, a KL-warmup and a low learning rate were used. Within 5 epochs, all of the models have a training accuracy of 10% and completely fail to fit the random labels. This is thus in contrast with variational Gaussian dropout, for which weight initialization made it possible to fit random labels. One possible explanation could be that variational Gaussian dropout gets stuck in a local minima and therefore still fits the random labels. However it might also imply that proper variational inference with a Laplace prior and Laplace posterior penalizes memorization of the dataset, and favours generalization more than variational Gaussian dropout does.

Figure 5.19: Trained from a random initialization either on a variationMNIST dataset with true labels, or randomized labels.



(a) Bayesian Le-Net 300.



(b) Bayesian 3 layer-MLP, 1500 neurons per layer.

Figure 5.20: Deterministic/vanilla 3 layer-MLP, 1500 neurons per layer. Training error reaches 100% training-accuracy on a dataset with random labels.

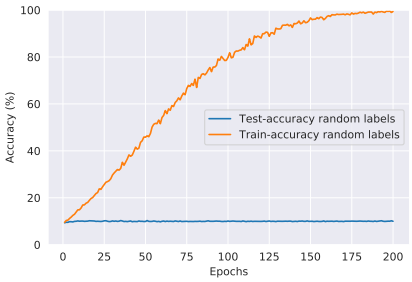
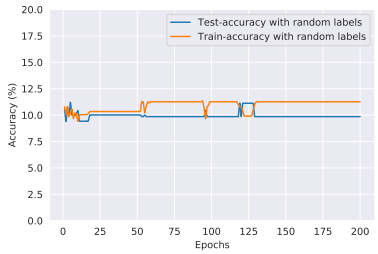
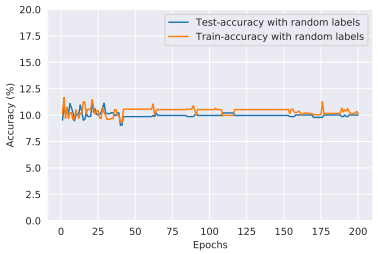


Figure 5.21: Examples from dataset with rotated MNIST images and random backgrounds.

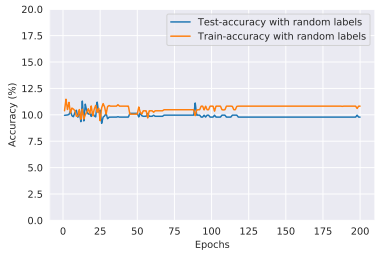
Figure 5.22: Random label experiment: Test-accuracy on variationMNIST with rotated digits with a random background.



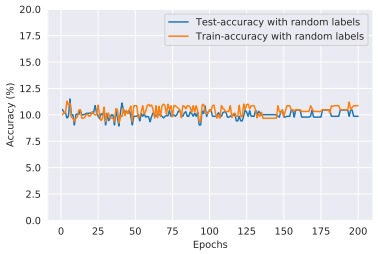
(a) Prior scale 0.001



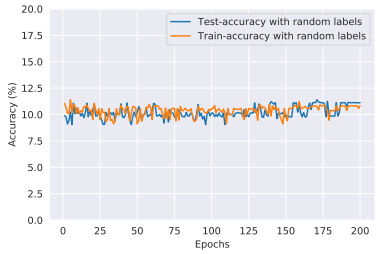
(b) Prior scale 0.05



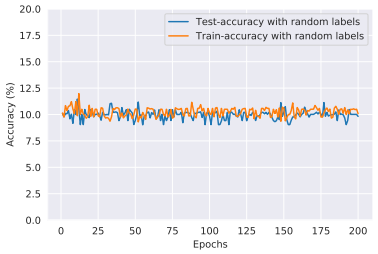
(c) Prior scale 0.1



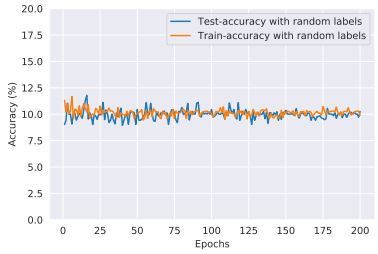
(d) Prior scale 0.5



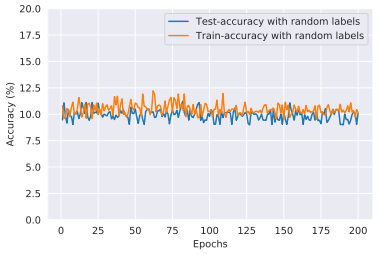
(e) Prior scale 1.0



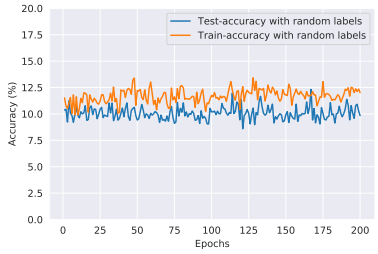
(f) Prior scale 2.0



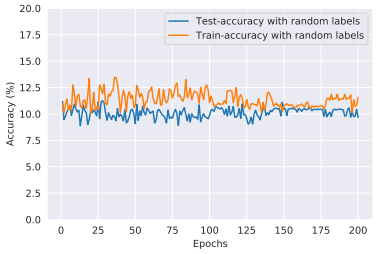
(g) Prior scale 5.0



(h) Prior scale 10.0

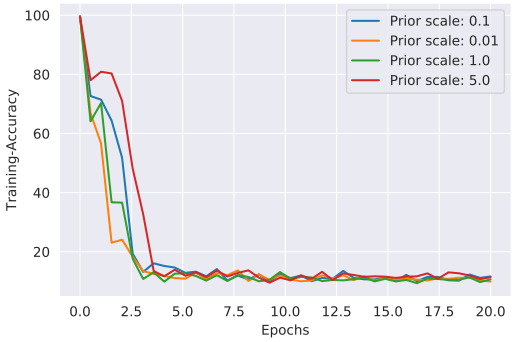


(i) Prior scale 40.0



(j) Prior scale 100.0

Figure 5.23: Fitting of random layers with 3-layer Bayesian MLP, initalized with weights wich where pretrained to an training accuracy of 100%.



Chapter 6

Summary and Conclusions

6.1 Conclusion

In this thesis we have explored compression of neural networks, with a particular focus on using variational inference to induce sparsity, after which the network is pruned. We explored the performance of various training methods, Bayesian and non-Bayesian, for inducing structured or non-structured sparsity in neural networks.

We showed that Bayesian group lasso has a performance on par with variational inference using a Normal-Jeffreys prior, and that both methods outperformed compression through (non-Bayesian) Group Lasso. Bayesian group lasso does have in addition also more theoretical grounding as a fully Bayesian method.

Furthermore we explored using variational inference for creating random sparsity in the weights of a network, by Gaussian posteriors with a Gaussian and Laplace prior, and a Laplace posterior with a Laplace prior. An analytical approximation to the KL-divergence between a Gaussian variational distribution and a Laplace prior was derived. We showed that while compressing a pre-trained network, the total gradients where dominated by the gradients due to the KL-loss term. This suggests a possible improved training performance when using the analytical expression instead of Monte Carlo sampling.

The influence of using compression for transfer learning was also explored, with improvements in both convergence speed as well as accuracy on some tasks, whereas on other tasks an improvement in the speed of transfer learning was found.

Finally we explore the ability of Bayesian neural networks trained with variational inference to fit data with random labels, and show that they are not able to fit the

random labels when non-Bayesian networks do so easily. Furthermore, in contrast to variational Gaussian dropout, it is not possible to fit the random labels even when the network is initialized with weights from a non-Bayesian network which was trained to fit the random labels perfectly.

6.2 Future work

For future work there are multiple interesting directions to explore;

- One possible modification to Bayesian group lasso is having a shared variance parameter for each group in Bayesian group lasso. This allows the KL-divergence to have an analytical form. See for a proof of the analytical KL-expression the Appendix. From experiments it was found that the variance of groups in Bayesian group lasso have roughly the same variance, and thus the expressivity of the variational distribution might not be hurt too much. Furthermore the analytical KL-expression might be able to help speed up training by eliminating the need for Monte-Carlo sampling of the KL-divergence.
- The combination of student-teacher training with group lasso was found to have little improvement on the performance of the compression algorithms. However as discussed in Section 5.5, the combination of both methods might perform better when the compressed model is allowed to have a significantly lower accuracy than the uncompressed model.
- The posterior uncertainties of the Bayesian neural networks could be used to determine quantization levels of weights [43], by decreasing the precision of weights with high variance drastically.
- In all the experiments done in these thesis, the scale or variances of priors were kept constant among the weights. By using priors with different scales, it might be possible to encode beliefs about which parts of the network should be compressed more. One example might be to have more compression in the final layers of a CNN than in the first layers.
- While student-teacher training has been explored here in a deterministic context, the variational posterior distribution could also be used for *Bayesian Dark Knowledge*. While *Bayesian Dark Knowledge* was originally performed using sampling methods in Bayesian neural networks, it could easily be extended to use variational posteriors.
- Stable training with variational inference is not automatic, and it seems that

there are a lot of optimization and initialization heuristics which improve training. Some heuristics which could also be used are to have different learning rates for variances and for means. The good compression result using variational inference suggest that pre-initializing Bayesian neural networks with pre-trained weights might speed up convergence.

- A further exploration of iterative methods might yield interesting results. Iterative pruning might be able to speed up the time to compress a network with variational inference significantly.

Bibliography

[1] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Sergey Bakin et al. "Adaptive regression and model selection in data mining problems". In: (1999).

[3] N. Balakrishnan. *A primer on statistical distributions N. Balakrishnan and V.B. Nevzorov.* eng. Hoboken, N.J.: Wiley, 2003. ISBN: 0471427985.

[4] Anoop Korattikara Balan et al. "Bayesian dark knowledge". In: *Advances in Neural Information Processing Systems*. 2015, pp. 3438–3446.

[5] Yoshua Bengio et al. "Convex neural networks". In: *Advances in neural information processing systems*. 2006, pp. 123–130.

[6] Chris Bishop, Christopher M Bishop, et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] Charles Blundell et al. "Weight uncertainty in neural networks". In: *arXiv preprint arXiv:1505.05424* (2015).

[9] Aydin Buluç et al. "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks". In: *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. ACM. 2009, pp. 233–244.

[10] Roland W Butler, Andrew TA Wood, et al. "Laplace approximations for hypergeometric functions with matrix argument". In: *The Annals of Statistics* 30.4 (2002), pp. 1155–1177.

[11] Tianqi Chen et al. "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems". In: *arXiv preprint arXiv:1512.01274* (2015).

[12] Yu Cheng et al. "A survey of model compression and acceleration for deep neural networks". In: *arXiv preprint arXiv:1710.09282* (2017).

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

[14] Emily L Denton et al. "Exploiting linear structure within convolutional networks for efficient evaluation". In: *Advances in neural information processing systems*. 2014, pp. 1269–1277.

[15] Mário Figueiredo. "Adaptive sparseness using Jeffreys prior". In: *Advances in neural information processing systems*. 2002, pp. 697–704.

[16] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Training pruned neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[17] Brendan J. Frey and Geoffrey E. Hinton. "Efficient stochastic source coding and an application to a Bayesian network source model". In: *The Computer Journal* 40.2_and_3 (1997), pp. 157–165.

[18] Manuel Gil. "On Rényi divergence measures for continuous alphabet sources". PhD thesis. 2011.

[19] Paul Glasserman. *Monte Carlo methods in financial engineering*. Vol. 53. Springer Science & Business Media, 2013.

[20] Yunchao Gong et al. "Compressing deep convolutional networks using vector quantization". In: *arXiv preprint arXiv:1412.6115* (2014).

[21] Ronald L Graham et al. "Concrete mathematics: a foundation for computer science". In: *Computers in Physics* 3.5 (1989), pp. 106–107.

[22] Alex Graves. "Practical variational inference for neural networks". In: *Advances in neural information processing systems*. 2011, pp. 2348–2356.

[23] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[24] Babak Hassibi and David G Stork. "Second order derivatives for network pruning: Optimal brain surgeon". In: *Advances in neural information processing systems*. 1993, pp. 164–171.

[25] Kaiming He et al. "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.

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

[27] 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.

[28] Antti Honkela and Harri Valpola. "Variational learning and bits-back coding: an information-theoretic view to Bayesian learning". In: *IEEE Transactions on Neural Networks* 15.4 (2004), pp. 800–810.

[29] Jiri Hron, Alexander G de G Matthews, and Zoubin Ghahramani. "Variational Gaussian Dropout is not Bayesian". In: *arXiv preprint arXiv:1711.02989* (2017).

[30] Martin Jankowiak and Fritz Obermeyer. "Pathwise Derivatives Beyond the Reparameterization Trick". In: *arXiv preprint arXiv:1806.01851* (2018).

[31] Eric Jones, Travis Oliphant, and Pearu Peterson. "{SciPy}: open source scientific tools for {Python}". In: (2014).

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

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

[34] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[35] Samuel Kotz, Tomasz Kozubowski, and Krzystof Podgorski. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Springer Science & Business Media, 2012.

[36] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[37] Minjung Kyung et al. "Penalized regression, standard errors, and Bayesian lassos". In: *Bayesian Analysis* 5.2 (2010), pp. 369–411.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[39] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[40] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[41] Yingzhen Li and Richard E Turner. "Rényi divergence variational inference". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1073–1081.

[42] Pavel Loskot and Norman C Beaulieu. "On monotonicity of the hypersphere volume and area". In: *Journal of Geometry* 87.1-2 (2007), pp. 96–98.

[43] Christos Louizos, Karen Ullrich, and Max Welling. "Bayesian compression for deep learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3290–3300.

[44] Christos Louizos and Max Welling. "Multiplicative normalizing flows for variational bayesian neural networks". In: *arXiv preprint arXiv:1703.01961* (2017).

[45] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[46] N.W McLachlan. *Bessel functions for engineers*. eng. 2nd ed. Oxford engineering science series. Oxford: Clarendon, 1961.

[47] KS Miller. "Distributions involving norms of correlated Gaussian vectors". In: *Quarterly of Applied Mathematics* 22.3 (1964), pp. 235–243.

[48] KS Miller, RI Bernstein, and LE Blumenson. "Generalized rayleigh processes". In: *Quarterly of Applied Mathematics* 16.2 (1958), pp. 137–145.

[49] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. "Variational dropout sparsifies deep neural networks". In: *arXiv preprint arXiv:1701.05369* (2017).

[50] Keith E Muller. "Computing the confluent hypergeometric function, M (a, b, x)". In: *Numerische Mathematik* 90.1 (2001), pp. 179–196.

[51] JH Park Jr. "Moments of the generalized Rayleigh distribution". In: *Quarterly of Applied Mathematics* 19.1 (1961), pp. 45–49.

[52] John W Pearson, Sheehan Olver, and Mason A Porter. "Numerical methods for the computation of the confluent and Gauss hypergeometric functions". In: *Numerical Algorithms* 74.3 (2017), pp. 821–866.

[53] Jorma Rissanen. "Modeling by shortest data description". In: *Automatica* 14.5 (1978), pp. 465–471.

[54] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), p. 533.

[55] John Schulman et al. "Gradient estimation using stochastic computation graphs". In: *Advances in Neural Information Processing Systems*. 2015, pp. 3528–3536.

[56] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[57] Casper Kaae S*ø*nderby et al. "Ladder variational autoencoders". In: *Advances in neural information processing systems*. 2016, pp. 3738–3746.

[58] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[59] Lisa Torrey and Jude Shavlik. "Transfer learning". In: *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*. IGI Global, 2010, pp. 242–264.

[60] Karen Ullrich, Edward Meeds, and Max Welling. "Soft weight-sharing for neural network compression". In: *arXiv preprint arXiv:1702.04008* (2017).

[61] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. "Improving the speed of neural networks on CPUs". In: *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*. Vol. 1. Citeseer. 2011, p. 4.

[62] 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.

[63] Wei Wen et al. "Learning structured sparsity in deep neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082.

[64] Sergei Winitzki. "A handy approximation for the error function and its inverse". In: *A lecture note obtained through private communication* (2008).

[65] Xiaofan Xu, Malay Ghosh, et al. "Bayesian variable selection and estimation for group lasso". In: *Bayesian Analysis* 10.4 (2015), pp. 909–936.

[66] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[67] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).

Appendix

Compressed sparse row (CSR) format and Compressed sparse row format

A weight matrix with high sparsity can be stored in the compressed sparse-row format. Compressed sparse row matrix stores an $m \times n$ matrix W, using three vectors A, IR, IC. All the non-zero elements of W are stored consecutively in a vector A of floats, where the order of storing is in column-major order, which means that the column index changes fastest. The vector IR has length $m + 1$ and is defined recursively, where $IR[0] = 0$ and $IR[i] = IR[i-1] +$ #of non-zero element in row (i-1) of W. IC is a vector which stores the column index [9]. The compression rate α achieved by using the CSR is:

$$\alpha = \frac{mn}{(n-1) + 2\sum_i \sum_j A_{i,j} I(A_{i,j} \neq 0)} \tag{1}$$

However note that only the vector A has to store floats while IR and IC can store integers, improving the practical compression rate.

Bayesian group lasso with shared variance parameter

Let the variational distribution be a factorized Gaussian, where the weights have been divided into G groups, and each group has a shared variance parameter: $p(\boldsymbol{w}) = \prod_g^G \prod_i \mathcal{N}(w_{i,g}; \mu_{i,g}, \sigma_g^2)$. The prior is a Bayesian group lasso prior, defined over the same groups as the posterior:

$$p(\mathbf{w}) = \prod_{i=1}^{G} \frac{\Gamma(\frac{|g_i|}{2})}{2\pi^{\frac{|g_i|}{2}}\ b^{|g_i|}\ \Gamma(|g_i|)} e^{-\frac{1}{b}\sqrt{\sum_i^{|g_i|}(w_i^{g_i})^2}} \tag{2}$$

The total KL-divergence can then be split up into a sum over the KL-divergence of each group. The KL-divergence for a group of size $|g|$ can be given as, while switching

notation and $\boldsymbol{w}$ now denotes all weight in this group.:

$$KL(q(\boldsymbol{w})||p(\boldsymbol{w})) = -H(q(\boldsymbol{w}) - H(q(\boldsymbol{w}), p(\boldsymbol{w})) \tag{3}$$

$$KL(q(\boldsymbol{w})||p(\boldsymbol{w})) = -H(q(\boldsymbol{w})) - \mathbb{E}_{q(\boldsymbol{w})}\left[\log(p(\boldsymbol{w}))\right] \tag{4}$$

$$KL(q(\boldsymbol{w})||p(\boldsymbol{w})) = -H(q(\boldsymbol{w})) - \int_{\mathbb{R}^{|g|}} \prod_{i}^{|g|} \mathcal{N}(w_i; \mu_i, \sigma^2) \log\Big(.\frac{\Gamma(\frac{|g|}{2})}{2\pi^{\frac{|g|}{2}}\ b^{|g|}\ \Gamma(|g|)}\Big) + \tag{5}$$

$$\int_{\mathbb{R}^{|g|}} \prod_{i}^{|g|} \mathcal{N}(w_i; \mu_i, \sigma^2) \frac{1}{b} \sqrt{\sum_{i}^{|g|} (w_i)^2} = \tag{6}$$

$$KL(q(\boldsymbol{w})||p(\boldsymbol{w})) = -H(q(\boldsymbol{w})) - \log\Big(\frac{\Gamma(\frac{|g|}{2})}{2\pi^{\frac{|g|}{2}} b^{|g|} \Gamma(|g|)}\Big) + \int_{\mathbb{R}^{|g|}} \mathcal{N}\big(\boldsymbol{w}; \mu, \sigma^2 I\big) \frac{1}{b} \sqrt{\sum_{i}^{|g|} (w_i)^2} \tag{7}$$

The entropy can be easily computed for a factorized Gaussian distribution, and the second term is a constant. The third term can be recognized as the expected ℓ_2 norm: $\mathbb{E}[|\boldsymbol{w}|_2]$ of the Gaussian multivariate distribution with mean-vector μ and diagonal covariance matrix $\sigma^2 I$; $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{w}, \sigma^2 I)$. Using Theorem .0.1, the KL-divergence can be expressed as:

$$KL(q(\boldsymbol{w})||p(\boldsymbol{w})) = -H(q(\boldsymbol{w})) - \log\left(\frac{\Gamma(\frac{|g|}{2})}{2\pi^{\frac{|g|}{2}} b^{|g|} \Gamma(|g|)}\right) + \tag{8}$$

$$\sigma\sqrt{2} e^{\frac{-\lambda^2}{2}} \frac{\Gamma((|g|+1)/2)}{\Gamma(|g|/2)} M\left(\frac{|g|+1}{2}, \frac{|g|}{2}, \frac{\lambda^2}{2}\right). \tag{9}$$

with $M\left(\frac{|g|+1}{2}, \frac{|g|}{2}, \frac{\lambda^2}{2}\right)$ being the confluent hypergeometric function and $\lambda = \sqrt{\sum_i (\frac{\mu_i}{\sigma})^2}$. Although the confluent hypergeometric function is not yet implemented in any widely used deep learning framework, it is implemented in the python package scipy [31]. Furthermore, the number of times the confluent hypergeometric function needs to be evaluated is equal to the number of groups since $M()$ is only dependent on $|g|$ and λ. Approximations of the confluent hypergeometric are well known [50], but the computation is usally done by summing an equivalent powerseries or using an assymptotic expansion [31]. However what is important when using variational inference in neural network and using the back-propagation algorithm is that the derivative of the approximation is computatianally feasible. For a hyperconfluent geometric function M(a,b,x) there is a Laplace approximation [10] available. For a further recent overview of approximation methods of the confluent hypergeometric function, see [52]. A very attractive property of the problem here is that the linearity of the expecation operator can be used to scale λ to a value where the confluent hypergeometric function is more easy to compute, or good approximations of limits are known, which would provide safeguarding against over and underflow errors.

Theorem .0.1. *If* $Z = \sqrt{\sum_i W_i^2}$ *with* $W_i \sim \mathcal{N}(w; \mu_i, \sigma^2)$*:*

$$\mathbb{E}[Z] = \sigma e^{\frac{-\lambda^2}{2}} \sqrt{2} \frac{\Gamma(\frac{|g|+1}{2})}{\Gamma(\frac{|g|}{2})} M\Big(\frac{|g|+1}{2}, \frac{|g|}{2}, \frac{\lambda^2}{2}\Big) \tag{10}$$

Proof. A sketch of the proof for case $\sigma = 1$ is given in [51]. Note that the result for $\sigma = 1$ can easily be generalized to arbitrary sigmas by using the linearity of the expectation operator. The full derivation proof is given below. Given a variable $Y = \sqrt{\sum_i X_i^2}$ with $X_i \sim \mathcal{N}(x; \frac{\mu_i}{\sigma}, 1)$, Y is distributed as a noncentral chi-distribution sometimes called the generalized rayleigh distribution with the pdf being [48] [47]:

$$p(y) = \lambda\big(\frac{y}{\lambda}\big)^{|g|/2} e^{\frac{-(\lambda^2+y^2)}{2}} I_{(|g|-2)/2}(y\lambda) \tag{11}$$

with λ being the ℓ_2 norm of the means of W_i. $\lambda = \sqrt{\sum_i (\frac{\mu_i}{\sigma})^2}$. And $I_\alpha()$ is the modified Bessel function of the first kind. By now making a change of variables, $Z = \sigma Y = \sqrt{\sum_i (\sigma X_i)^2}$, with $\sigma > 0$, the pdf of Z is:

$$p(z) = \lambda\big(\frac{z/\sigma}{\lambda}\big)^{|g|/2} e^{\frac{-(\lambda^2+(\frac{z}{\sigma})^2)}{2}} I_{(|g|-2)/2}\Big(\frac{z\lambda}{\sigma}\Big) \Big|\frac{dy}{dz}\Big| \tag{12}$$

$$= \frac{1}{\sigma}\lambda\big(\frac{z/\sigma}{\lambda}\big)^{|g|/2} e^{\frac{-(\lambda^2+(\frac{z}{\sigma})^2)}{2}} I_{(|g|-2)/2}\Big(\frac{z\lambda}{\sigma}\Big) \tag{13}$$

Note that σX_i is distributed as $W_i \sim \mathcal{N}(w, \mu_i, \sigma^2)$. The expectation of Z is:

$$\mathbb{E}[Z] = \int_0^\infty z \frac{1}{\sigma}\lambda\big(\frac{z/\sigma}{\lambda}\big)^{|g|/2} e^{\frac{-(\lambda^2+(\frac{z}{\sigma})^2)}{2}} I_{(|g|-2)/2}\Big(\frac{z\lambda}{\sigma}\Big) dz \tag{14}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \int_0^\infty \frac{z}{\sigma}\big(\frac{z}{\sigma}\big)^{|g|/2} e^{\frac{-(\frac{z}{\sigma})^2}{2}} I_{(|g|-2)/2}\Big(\frac{z\lambda}{\sigma}\Big) dz \tag{15}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \int_0^\infty \big(\frac{z}{\sigma}\big)^{1+\frac{|g|}{2}} e^{-\frac{(\frac{z}{\sigma})^2}{2}} I_{(|g|-2)/2}\Big(\frac{z\lambda}{\sigma}\Big) dz \tag{16}$$

$$\tag{17}$$

and now using the expression of the modified Bessel function of the first kind as a

power series, as given in [46]; $I_\alpha(x) = \sum_{r=0}^{\infty} \frac{(\frac{x}{2})^{\alpha+2r}}{r!\Gamma(\alpha+r+1)}$.

$$\mathbb{E}[Z] = \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \int_0^\infty \left(\frac{z}{\sigma}\right)^{1+\frac{|g|}{2}} e^{-\frac{(\frac{z}{\sigma})^2}{2}} \sum_{r=0}^{\infty} \frac{\left(\frac{z\lambda}{2\sigma}\right)^{\frac{|g|-2}{2}+2r}}{r!\Gamma(\frac{(|g|-2)}{2}+r+1)} dz \tag{18}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \int_0^\infty \left(\frac{z}{\sigma}\right)^{1+\frac{|g|}{2}} \sum_{r=0}^{\infty} \frac{e^{-\frac{(\frac{z}{\sigma})^2}{2}} \left(\frac{\lambda}{2}\right)^{\frac{|g|-2}{2}+2r} \left(\frac{z}{\sigma}\right)^{\frac{|g|-2}{2}+2r}}{r!\Gamma((|g|+2r)/2)} dz \tag{19}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \int_0^\infty \left(\frac{z}{\sigma}\right)^{1+\frac{|g|}{2}} \sum_{r=0}^{\infty} \frac{e^{-\frac{(\frac{z}{\sigma})^2}{2}} \left(\frac{z}{\sigma}\right)^{\frac{|g|-2}{2}+2r}}{r!\Gamma((|g|+2r)/2)} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} dz \tag{20}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \int_0^\infty \sum_{r=0}^{\infty} \frac{e^{-\frac{(\frac{z}{\sigma})^2}{2}} \left(\frac{z}{\sigma}\right)^{|g|+2r}}{r!\Gamma((|g|+2r)/2)} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} dz \tag{21}$$

$$\tag{22}$$

and by interchanging the summation and integration:

$$\mathbb{E}[Z] = \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} \frac{1}{r!\Gamma((|g|+2r)/2)} \int_0^\infty \left(\frac{z}{\sigma}\right)^{|g|+2r} e^{-\frac{(\frac{z}{\sigma})^2}{2}} dz \tag{23}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} \frac{2^{\frac{(|g|+2r)}{2}}}{r!\Gamma((|g|+2r)/2)} \int_0^\infty \left(\frac{(\frac{z}{\sigma})^2}{2}\right)^{\frac{(|g|+2r)}{2}} e^{-\frac{(\frac{z}{\sigma})^2}{2}} dz \tag{24}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} \frac{2^{\frac{(|g|+2r)}{2}}}{r!\Gamma((|g|+2r)/2)} \int_0^\infty \left(\frac{(\frac{z}{\sigma})^2}{2}\right)^{\frac{(|g|+2r-1+1)}{2}} e^{-\frac{(\frac{z}{\sigma})^2}{2}} dz \tag{25}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} \frac{2^{\frac{(|g|+2r)}{2}}}{r!\Gamma((|g|+2r)/2)} \int_0^\infty \left(\frac{(\frac{z}{\sigma})^2}{2}\right)^{\frac{(|g|+2r-1)}{2}} \frac{\left(\frac{z}{\sigma}\right)}{\sqrt{2}} e^{-\frac{(\frac{z}{\sigma})^2}{2}} dz \tag{26}$$

$$\tag{27}$$

and subsequently rewriting the integral as a gamma function $\Gamma(k) = \int_0^\infty x^{k-1} e^{-x} dx$ by

making the substitution $x = \frac{(\frac{z}{\sigma})^2}{2}$, $\frac{dx}{dz} = \frac{z}{\sigma^2}$:

$$\mathbb{E}[Z] = \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} \frac{2^{\frac{(|g|+2r)}{2}}}{r!\Gamma((|g|+2r)/2)} \frac{1}{\sqrt{2}} \int_0^{\infty} (x)^{\frac{(|g|+2r-1)}{2}} \sigma e^{-x} dz \tag{28}$$

$$= \lambda^{1-\frac{|g|}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{\frac{|g|-2}{2}+2r} \frac{2^{\frac{(|g|+2r)}{2}}}{r!\Gamma((|g|+2r)/2)} \frac{1}{\sqrt{2}} \int_0^{\infty} (x)^{\left(\frac{|g|+2r+1}{2}\right)-1} \sigma e^{-x} dz \tag{29}$$

$$= \lambda^{1-\frac{|g|}{2}} (\frac{\lambda}{2})^{\frac{|g|-2}{2}} e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{2r} \frac{2^{\frac{(|g|+2r)}{2}}}{r!\Gamma((|g|+2r)/2)} \frac{\sigma}{\sqrt{2}} \Gamma(r + \frac{|g|+1}{2}) \tag{30}$$

$$= \lambda^0 e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda}{2})^{2r} \frac{2^{\frac{(|g|+2r-|g|+2)}{2}}}{r!\Gamma((|g|+2r)/2)} \frac{\sigma}{\sqrt{2}} \Gamma(r + \frac{|g|+1}{2}) \tag{31}$$

$$\mathbb{E}[Z] = e^{\frac{-\lambda^2}{2}} \sum_{r=0}^{\infty} (\frac{\lambda^2}{2})^r \frac{\sqrt{2}\sigma}{r!\Gamma((|g|+2r)/2)} \Gamma(r + \frac{|g|+1}{2}) \tag{32}$$

$$= e^{\frac{-\lambda^2}{2}} \sqrt{2}\sigma \sum_{r=0}^{\infty} \frac{\Gamma(r + \frac{|g|+1}{2})}{r!\Gamma((|g|+2r)/2)} (\frac{\lambda^2}{2})^r \tag{33}$$

The gamma function $\Gamma(x) = (x-1)!$, and thus $u\Gamma(u) = \Gamma(u+1)$, therefore;:

$$\frac{|g|+1+2r}{|g|+2r} \frac{\Gamma(r + \frac{|g|+1}{2})}{\Gamma((|g|+2r)/2)} = \frac{\frac{|g|+1+2r}{2}}{\frac{|g|+2r}{2}} \frac{\Gamma(r + \frac{|g|+1}{2})}{\Gamma((|g|+2r)/2)} = \frac{\Gamma(1 + r + \frac{|g|}{2})}{\Gamma(1 + (|g|+2r)/2)} \tag{34}$$

and this can be generalized to:

$$\frac{\Gamma(r + \frac{|g|+1}{2})}{r!\Gamma((|g|+2r)/2)} = \prod_{j=0}^{r-1} \frac{|g|+1+2j}{|g|+2j} \frac{\Gamma(\frac{|g|+1}{2})}{\Gamma(\frac{|g|}{2})} \tag{35}$$

and thus Equation 33 can be written as:

$$\mathbb{E}[Z] = e^{\frac{-\lambda^2}{2}} \sqrt{2}\sigma \frac{\Gamma(\frac{|g|+1}{2})}{\Gamma(\frac{|g|}{2})} \left[1 + \sum_{r=1}^{\infty} \frac{1}{r!} \Big(\prod_{j=0}^{r-1} \frac{|g|+1+2j}{|g|+2j} \Big) (\frac{\lambda^2}{2})^r \right] \tag{36}$$

$\prod_{j=0}^{r-1} \frac{|g|+1+2j}{|g|+2j}$ is a ratio of two rising factorials [21]:

$$\prod_{j=0}^{r-1} \frac{|g|+1+2j}{|g|+2j} = \prod_{j=0}^{r-1} \frac{2(\frac{|g|+1}{2}+j)}{2(\frac{|g|}{2}+j)} = \frac{2^r}{2^r} \prod_{j=0}^{r-1} \frac{\frac{|g|+1}{2}+j}{\frac{|g|}{2}+j} \tag{37}$$

and thus the term in brackets in Equation 36 is the power series for a confluent hypergeometric function [21] M(a,b,z) with $a = \frac{|g|+1}{2}$, $b = \frac{|g|}{2}$ and $z = \frac{\lambda^2}{2}$. Finally we can write:

$$\mathbb{E}[Z] = \sigma e^{\frac{-\lambda^2}{2}} \sqrt{2} \frac{\Gamma(\frac{|g|+1}{2})}{\Gamma(\frac{|g|}{2})} M\Big(\frac{|g|+1}{2}, \frac{|g|}{2}, \frac{\lambda^2}{2} \Big) \tag{38}$$

□

Dear examiner,

I have made a code-repository on the MLSALT-machines under;
/homes/sd782/Thesis_Sjoerd_de_Jong_Code.

Please let me know if anything is unclear or more information is needed.

Sincerely,

Sjoerd de Jong