Bayesian Compression for Deep Learning

Christos Louizos
University of Amsterdam
TNO Intelligent Imaging
c.louizos@uva.nl

Karen Ullrich
University of Amsterdam
k.ullrich@uva.nl

Max Welling
University of Amsterdam
CIFAR*
m.welling@uva.nl

Abstract

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

1 Introduction

While deep neural networks have become extremely successful in in a wide range of applications, often exceeding human performance, they remain difficult to apply in many real world scenarios. For instance, making billions of predictions per day comes with substantial energy costs given the energy consumption of common Graphical Processing Units (GPUs). Also, real-time predictions are often about a factor 100 away in terms of speed from what deep NNs can deliver, and sending NNs with millions of parameters through band limited channels is still impractical. As a result, running them on hardware limited devices such as smart phones, robots or cars requires substantial improvements on all of these issues. For all those reasons, compression and efficiency have become a topic of interest in the deep learning community.

While all of these issues are certainly related, compression and performance optimizing procedures might not always be aligned. As an illustration, consider the convolutional layers of Alexnet, which account for only 4% of the parameters but 91% of the computation [Sze et al., 2017]. Compressing these layers will not contribute much to the overall memory footprint.

There is a variety of approaches to address these problem settings. However, most methods have the common strategy of reducing both the neural network structure and the effective fixed point precision for each weight. A justification for the former is the finding that NNs suffer from significant parameter redundancy [Denil et al., 2013]. Methods in this line of thought are network pruning, where unnecessary connections are being removed [LeCun et al., 1989, Han et al., 2015, Guo et al., 2016], or student-teacher learning where a large network is used to train a significantly smaller network [Ba and Caruana, 2014, Hinton et al., 2015].

From a Bayesian perspective network pruning and reducing bit precision for the weights is aligned with achieving high accuracy, because Bayesian methods search for the optimal model structure (which leads to pruning with sparsity inducing priors), and reward uncertain posteriors over parameters through the bits back argument [Hinton and Van Camp, 1993] (which leads to removing insignificant

*Canadian Institute For Advanced Research.

bits). This relation is made explicit in the MDL principle [Grünwald, 2007] which is known to be related to Bayesian inference.

In this paper we will use the variational Bayesian approximation for Bayesian inference which has also been explicitly interpreted in terms of model compression [Hinton and Van Camp, 1993]. By employing sparsity inducing priors for hidden units (and not individual weights) we can prune neurons including all their ingoing and outgoing weights. This avoids more complicated and inefficient coding schemes needed for pruning or vector quantizing individual weights. As a additional Bayesian bonus we can use the posterior uncertainties to assess which bits are significant and remove the ones which fluctuate too much under posterior sampling. From this we derive the optimal fixed point precision per layer, which is still practical on chip.

2 Variational Bayes and Minimum Description Length

A fundamental theorem in information theory is the minimum description length (MDL) principle [Grünwald, 2007]. It relates to compression directly in that it defines the best hypothesis to be the one that communicates the sum of the model (complexity cost $\mathcal{L}^C$) and the data misfit (error cost $\mathcal{L}^E$) with the minimum number of bits [Rissanen, 1978, 1986]. It is well understood that variational inference can be reinterpreted from an MDL point of view [Peterson, 1987, Wallace, 1990, Hinton and Van Camp, 1993, Honkela and Valpola, 2004, Graves, 2011]. More specifically, assume that we are presented with a dataset $\mathcal{D}$ that consists from N input-output pairs $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. Let $p(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^N p(y_i|\mathbf{x}_i, \mathbf{w})$ be a parametric model, e.g. a deep neural network, that maps inputs $\mathbf{x}$ to their corresponding outputs y using parameters $\mathbf{w}$ governed by a prior distribution $p(\mathbf{w})$. In this scenario, we wish to approximate the intractable posterior distribution $p(\mathbf{w}|\mathcal{D}) = p(\mathcal{D}|\mathbf{w})p(\mathbf{w})/p(\mathcal{D})$ with a fixed form approximate posterior $q_\phi(\mathbf{w})$ by optimizing the variational parameters ϕ according to:

$$\mathcal{L}(\phi) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathcal{D}|\mathbf{w})]}_{\mathcal{L}^E} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathbf{w})] + \mathcal{H}(q_\phi(\mathbf{w}))}_{\mathcal{L}^C}, \tag{1}$$

where $\mathcal{H}(\cdot)$ denotes the entropy and $\mathcal{L}(\phi)$ is known as the evidence-lower-bound (ELBO) or negative variational free energy. As indicated in eq. 1, $\mathcal{L}(\phi)$ naturally decomposes into a minimum cost for communicating the targets $\{y_n\}_{n=1}^N$ under the assumption that the sender and receiver agreed on a prior $p(\mathbf{w})$ and that the receiver knows the inputs $\{\mathbf{x}_n\}_{n=1}^N$ and form of the parametric model.

By using sparsity inducing priors for groups of weights that feed into a neuron the Bayesian mechanism will start pruning hidden units that are not strictly necessary for prediction and thus achieving compression. But there is also a second mechanism by which Bayes can help us compress. By explicitly entertaining noisy weight encodings through $q_\phi(\mathbf{w})$ we can benefit from the bits-back argument [Hinton and Van Camp, 1993, Honkela and Valpola, 2004] due to the entropy term; this is in contrast to infinitely precise weights that lead to $\mathcal{H}(\delta(\mathbf{w})) = -\infty$[2]. Nevertheless in practice, the data misfit term $\mathcal{L}^E$ is intractable for neural network models under a noisy weight encoding, so as a solution Monte Carlo integration is usually employed. Continuous $q_\phi(\mathbf{w})$ allow for the reparametrization trick [Kingma and Welling, 2014, Rezende et al., 2014]. Here, we replace sampling from $q_\phi(\mathbf{w})$ by sampling from a deterministic function of the variational parameters ϕ and noise variables ϵ:

$$\mathcal{L}(\phi) = \mathbb{E}_{p(\epsilon)}[\log p(\mathcal{D}|f(\phi, \epsilon))] + \mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathbf{w})] + \mathcal{H}(q_\phi(\mathbf{w})), \tag{2}$$

where $\mathbf{w} = f(\phi, \epsilon)$. By applying this trick, we obtain unbiased stochastic gradients of the ELBO with respect to the variational parameters ϕ, thus resulting in a standard optimization problem that is fit for stochastic gradient ascent. The efficiency of the gradient estimator resulting from eq. 2 can be further improved for neural networks by utilizing local reparametrizations Kingma et al. [2015] (which we will use in our experiments); they provide variance reduction in an efficient way by locally marginalizing the weights at each layer and instead sampling the distribution of the pre-activations.

3 Related Work

One of the earliest ideas and most direct approaches to tackle efficiency is pruning. Originally introduced by LeCun et al. [1989], pruning has recently been demonstrated to be applicable to modern

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

architectures [Han et al., 2016, Guo et al., 2016]. It had been demonstrated that an overwhelming amount of up to 99,5% of parameters can be pruned in common architectures. There have been quite a few encouraging results obtained by (empirical) Bayesian approaches that employ weight pruning [Graves, 2011, Blundell et al., 2015, Nalisnick et al., 2015, Ullrich et al., 2017, Molchanov et al., 2017]. Nevertheless, weight pruning is in general inefficient for compression since the matrix format of the weights is not taken into consideration, therefore the Compressed Sparse Column (CSC) format has to be employed. Moreover, note that in conventional CNNs most flops are used by the convolution operation. Inspired by this observation, several authors proposed pruning schemes that take these considerations into account [Wen et al., 2016, Yang et al., 2017] or even go as far as efficiency aware architectures to begin with [Iandola et al., 2017, Dong et al., 2017, Howard et al., 2017]. From the Bayesian viewpoint, similar pruning schemes have been explored at MacKay [1995], Neal [1995], Lawrence [2002], Karaletsos and Rätsch [2015].

Given optimal architecture, NNs can further be compressed by quantization. More precisely, there are 2 common techniques. First, the set of accessible weights can be reduced drastically. As an extreme example, Courbariaux et al. [2015], Mellempudi et al. [2017], Rastegari et al. [2016], Zhu et al. [2017] and Courbariaux and Bengio [2016] trained NN to use only binary or tertiary weights with floating point gradients. This approach however is in need of significantly more parameters than their ordinary counterparts. Work by Gong et al. [2015] explores various techniques beyond binary quantization: k-means quantization, product quantization and residual quantization. Later studies extent this set to optimal fixed point [Lin et al., 2015] and hashing quantization [Chen et al., 2015]. Han et al. [2016] apply k-means clustering and consequent center training. From a practical point of view, however, all these are fairly unpractical during test time. For the computation of each feature map in a net, the original weight matrix must be reconstructed from the indexes in the matrix and a codebook that contains all the original weights. This is an expensive operation and this is why some studies propose a different approach than set quantization. Precision quantization simply reduces the bit size per weight. This has a great advantage over set quantization at inference time since feature maps can simply be computed with less precision weights. Several studies show that this has little to no effect on network accuracy when using 16bit weights [Merolla et al., 2016, Gupta et al., 2015, Courbariaux et al., 2014, Venkatesh et al., 2016, Chai et al., 2017]. Somewhat orthogonal to the above discussion but certainly relevant are approaches that customize the implementation of CNNs for hardware limited devices[Howard et al., 2017, Azarkhish et al., 2017, Shi and Chu, 2017].

4 Bayesian compression with scale mixtures of normals

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

$$z \sim p(z); \qquad w \sim \mathcal{N}(w; 0, z^2), \tag{3}$$

with z^2 serving as the variance of the zero-mean normal distribution over w. By treating the scales of w as random variables we can recover marginal prior distributions over the parameters that have heavier tails and more mass at zero; this subsequently biases the posterior distribution over w to be sparse. This family of distributions is known as scale-mixtures of normals [Beale et al., 1959, Andrews and Mallows, 1974] and it is quite general, as a lot of well known sparsity inducing distributions are special cases.

One example of the aforementioned framework is the spike-and-slab distribution [Mitchell and Beauchamp, 1988], the golden standard for sparse Bayesian inference. Under the spike-and-slab, the mixing density of the scales is a Bernoulli distribution, thus the marginal $p(w)$ has a delta "spike" at zero and a continuous "slab" over the real line. Unfortunately, this prior leads to a computationally expensive inference since we have to explore a space of 2^M models, where M is the number of the model parameters. Dropout [Hinton et al., 2012, Srivastava et al., 2014], one of the most popular regularization techniques for neural networks, can be interpreted as positing a spike and slab distribution over the weights where the variance of the "slab" is zero [Gal and Ghahramani, 2016, Louizos, 2015]. Another example is the Laplace distribution which arises by considering $p(z^2) = \text{Exp}(\lambda)$. The mode of the posterior distribution under a Laplace prior is known as the Lasso [Tibshirani, 1996] estimator and has been previously used for sparsifying neural networks at Wen et al. [2016], Scardapane et al. [2016]. While computationally simple, the Lasso estimator is prone to "shrinking" large signals [Carvalho et al., 2010] and only provides point estimates about the parameters. As a result it does not provide uncertainty estimates, it can potentially overfit and, according to the bits-back argument, is inefficient for compression.

For these reasons, in this paper we will tackle the problem of compression and efficiency in neural networks by adopting a full Bayesian treatment and inferring a posterior distribution over the parameters under a scale mixture prior. We will consider two choices for prior over the scales $p(z)$; the hyperparameter free log-uniform prior [Figueiredo, 2002, Kingma et al., 2015] and the half-Cauchy prior, which results into a horseshoe [Carvalho et al., 2010] distribution. Both of these distributions correspond to a continuous relaxation of the spike-and-slab prior and we provide a brief discussion on their shrinkage properties at Appendix C.

4.1 Reparametrizing variational dropout for group sparsity

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

$$p(w) \propto \int \frac{1}{|z|} \mathcal{N}(w|0, z^2) dz = \frac{1}{|w|}. \tag{4}$$

This alternative parametrization of the the log uniform prior is known in the statistics literature as the normal-Jeffreys prior and has been introduced by Figueiredo [2002]. This formulation allows to "couple" the scales of weights that belong to the same group (e.g. neuron or feature map), by simply sharing the corresponding scale variable z in the joint prior:

$$p(\mathbf{W}, \mathbf{z}) \propto \prod_{i}^{A} \frac{1}{|z_i|} \prod_{ij}^{A,B} \mathcal{N}(w_{ij}|0, z_i^2), \tag{5}$$

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

$$q_\phi(\mathbf{W}, \mathbf{z}) = \prod_{i=1}^{A} \mathcal{N}(z_i|\mu_{z_i}, \mu_{z_i}^2 \alpha_i) \prod_{i,j}^{A,B} \mathcal{N}(w_{ij}|z_i \mu_{ij}, z_i^2 \sigma_{ij}^2), \tag{6}$$

where α_i is the dropout rate [Srivastava et al., 2014, Kingma et al., 2015, Molchanov et al., 2017] of the given group. As explained at Kingma et al. [2015], Molchanov et al. [2017], the multiplicative parametrization of the approximate posterior over $\mathbf{z}$ suffers from high variance gradients; therefore we will follow Molchanov et al. [2017] and re-parametrize it in terms of $\sigma_{z_i}^2 = \mu_{z_i}^2 \alpha_i$, hence optimize w.r.t. $\sigma_{z_i}^2$. The lower bound under this prior and posterior becomes:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{z}) q_\phi(\mathbf{W}|\mathbf{z})}[\log p(\mathcal{D}|\mathbf{W})] - \mathbb{E}_{q_\phi(\mathbf{z})}[KL(q_\phi(\mathbf{W}|\mathbf{z})||p(\mathbf{W}|\mathbf{z}))] - KL(q_\phi(\mathbf{z})||p(\mathbf{z})). \tag{7}$$

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

$$KL(q_\phi(\mathbf{W}|\mathbf{z})||p(\mathbf{W}|\mathbf{z})) = \frac{1}{2} \sum_{i,j}^{A,B} \left(\log \frac{\cancel{z_i^2}}{\cancel{z_i^2} \sigma_{ij}^2} + \frac{\cancel{z_i^2} \sigma_{ij}^2}{\cancel{z_i^2}} + \frac{\cancel{z_i^2} \mu_{ij}^2}{\cancel{z_i^2}} - 1 \right). \tag{8}$$

This independence can be better understood if we consider a non-centered parametrization of the prior [Papaspiliopoulos et al., 2007]. More specifically, consider reparametrizing the weights as $\tilde{w}_{ij} = \frac{w_{ij}}{z_i}$; this will then result into $p(\mathbf{W}|\mathbf{z})p(\mathbf{z}) = p(\tilde{\mathbf{W}})p(\mathbf{z})$, where $p(\tilde{\mathbf{W}}) = \prod_{i,j} \mathcal{N}(\tilde{w}_{ij}|0, 1)$ and $\mathbf{W} = \text{diag}(\mathbf{z})\tilde{\mathbf{W}}$. Now if we perform variational inference under the $p(\tilde{\mathbf{W}})p(\mathbf{z})$ prior with a posterior of the form $q_\phi(\tilde{\mathbf{W}}, \mathbf{z}) = q_\phi(\tilde{\mathbf{W}}) q_\phi(\mathbf{z})$, with $q_\phi(\tilde{\mathbf{W}}) = \prod_{i,j} \mathcal{N}(\tilde{w}_{ij}|\mu_{ij}, \sigma_{ij}^2)$, then we see that we arrive at the same expressions for the negative KL-divergence from the prior to the posterior. Finally, the negative KL-divergence from the normal-Jeffreys scale prior $p(\mathbf{z})$ to the Gaussian posterior $q_\phi(\mathbf{z})$ depends only on the "implied" dropout rate, $\alpha_i = \sigma_{z_i}^2 / \mu_{z_i}^2$, and takes the following form [Molchanov et al., 2017]:

$$-KL(q_\phi(\mathbf{z})||p(\mathbf{z})) \approx \sum_{i}^{A} \big(k_1 \sigma(k_2 + k_3 \log \alpha_i) - 0.5 m(-\log \alpha_i) - k_1 \big), \tag{9}$$

where $\sigma(\cdot)$, $m(\cdot)$ are the sigmoid and softplus functions respectively[3] and $k_1 = 0.63576$, $k_2 = 1.87320$, $k_3 = 1.48695$. We can now prune entire groups of parameters by simply specifying a threshold for the variational dropout rate of the corresponding group, e.g. $\log \alpha_i = (\log \sigma^2_{z_i} - \log \mu^2_{z_i}) \geq t$. It should be mentioned that this prior parametrization readily allows for a more flexible marginal posterior over the weights as we now have a compound distribution, $q_\phi(\mathbf{W}) = \int q_\phi(\mathbf{W}|\mathbf{z}) q_\phi(\mathbf{z}) d\mathbf{z}$; this is in contrast to the original parametrization and the Gaussian approximations employed by Kingma et al. [2015], Molchanov et al. [2017]. At test time, in order to have a single feedforward pass we replace the distribution over $\mathbf{W}$ at each layer with a single weight matrix, the masked posterior mean:

$$\hat{\mathbf{W}} = \text{diag}(\mathbf{m})\, \mathbb{E}_{q(\mathbf{z})q(\tilde{\mathbf{W}})}[\text{diag}(\mathbf{z})\tilde{\mathbf{W}}] = \text{diag}\left(\mathbf{m} \odot \boldsymbol{\mu}_z\right)\mathbf{M}_W, \tag{10}$$

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

4.2 Group horseshoe with half-Cauchy scale priors

Another choice for $p(z)$ is a proper half-Cauchy distribution: $\mathcal{C}^+(0, s) = 2(s\pi(1 + (z/s)^2))^{-1}$; it induces a horseshoe prior [Carvalho et al., 2010] distribution over the weights, which is a well known sparsity inducing prior in the statistics literature. More formally, the prior hierarchy over the weights is expressed as (in a non-centered parametrization):

$$s \sim \mathcal{C}^+(0, \tau_0); \qquad \tilde{z}_i \sim \mathcal{C}^+(0, 1); \qquad \tilde{w}_{ij} \sim \mathcal{N}(0, 1); \qquad w_{ij} = \tilde{w}_{ij}\tilde{z}_i s, \tag{11}$$

where τ_0 is the free parameter that can be tuned for specific desiderata. The idea behind the horseshoe is that of the "global-local" shrinkage; the global scale variable s pulls all of the variables towards zero whereas the heavy tailed local variables z_i can compensate and allow for some weights to escape. Instead of directly working with the half-Cauchy priors we will employ a decomposition of the half-Cauchy that relies upon (inverse) gamma distributions [Neville et al., 2014] as this will allow us to compute the KL-divergence from the scale prior $p(\mathbf{z})$ to the log-normal scale posterior $q_\phi(\mathbf{z})$ in closed form (the derivation is given in Appendix D). More specifically, we have that the half-Cauchy prior can be expressed in a non-centered parametrization as:

$$p(\tilde{\beta}) = \mathcal{IG}(0.5, 1); \qquad p(\tilde{\alpha}) = \mathcal{G}(0.5, k^2); \qquad z^2 = \tilde{\alpha}\tilde{\beta}, \tag{12}$$

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

$$s_b \sim \mathcal{IG}(0.5, 1); \quad s_a \sim \mathcal{G}(0.5, \tau_0^2); \quad \tilde{\beta}_i \sim \mathcal{IG}(0.5, 1); \quad \tilde{\alpha}_i \sim \mathcal{G}(0.5, 1); \quad \tilde{w}_{ij} \sim \mathcal{N}(0, 1);$$
$$w_{ij} = \tilde{w}_{ij}\sqrt{s_a s_b \tilde{\alpha}_i \tilde{\beta}_i}. \tag{13}$$

It should be mentioned that the improper log-uniform prior is the limiting case of the horseshoe prior when the shapes of the (inverse) Gamma hyperpriors on $\tilde{\alpha}_i, \tilde{\beta}_i$ go to zero [Carvalho et al., 2010]. In fact, several well known shrinkage priors can be expressed in this form by altering the shapes of the (inverse) Gamma hyperpriors [Armagan et al., 2011]. For the variational posterior we will employ the following mean field approximation:

$$q_\phi(s_b, s_a, \tilde{\boldsymbol{\beta}}) = \mathcal{LN}(s_b|\mu_{s_b}, \sigma^2_{s_b})\mathcal{LN}(s_a|\mu_{s_a}, \sigma^2_{s_a}) \prod_i^A \mathcal{LN}(\tilde{\beta}_i|\mu_{\tilde{\beta}_i}, \sigma^2_{\tilde{\beta}_i}) \tag{14}$$

$$q_\phi(\tilde{\boldsymbol{\alpha}}, \tilde{\mathbf{W}}) = \prod_i^A \mathcal{LN}(\tilde{\alpha}_i|\mu_{\tilde{\alpha}_i}, \sigma^2_{\tilde{\alpha}_i}) \prod_{i,j}^{A,B} \mathcal{N}(\tilde{w}_{ij}|\mu_{\tilde{w}_{ij}}, \sigma^2_{\tilde{w}_i j}), \tag{15}$$

where $\mathcal{LN}(\cdot, \cdot)$ is a log-normal distribution. It should be mentioned that a similar form of non-centered variational inference for the horseshoe has been also successfully employed for undirected models at [Ingraham and Marks, 2016]. Notice that we can also apply local reparametrizations [Kingma et al., 2015] when we are sampling $\sqrt{\tilde{\alpha}_i \tilde{\beta}_i}$ and $\sqrt{s_a s_b}$ by exploiting properties of the log-normal

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

distribution[4] and thus forming the implied:

$$\tilde{z}_i = \sqrt{\tilde{\alpha}_i \tilde{\beta}_i} \sim \mathcal{LN}(\mu_{\tilde{z}_i}, \sigma^2_{\tilde{z}_i}); \qquad s = \sqrt{s_a s_b} \sim \mathcal{LN}(\mu_s, \sigma^2_s) \tag{16}$$

$$\mu_{\tilde{z}_i} = \frac{1}{2}(\mu_{\tilde{\alpha}_i} + \mu_{\tilde{\beta}_i}); \quad \sigma^2_{\tilde{z}_i} = \frac{1}{4}(\sigma^2_{\tilde{\alpha}_i} + \sigma^2_{\tilde{\beta}_i}); \quad \mu_s = \frac{1}{2}(\mu_{s_a} + \mu_{s_b}); \quad \sigma^2_s = \frac{1}{4}(\sigma^2_{s_a} + \sigma^2_{s_b}). \tag{17}$$

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

$$p(z_i) = \mathcal{LN}(z_i|\mu_{z_i}, \sigma^2_{z_i}); \qquad \mu_{z_i} = \mu_{\tilde{z}_i} + \mu_s; \qquad \sigma^2_{z_i} = \sigma^2_{\tilde{z}_i} + \sigma^2_s, \tag{18}$$

i.e. prune when $(\sigma^2_{z_i} - \mu_{z_i}) \geq t$. This ignores dependencies among the z_i elements induced by the common scale s, but nonetheless we found that it works well in practice. Similarly with the group normal-Jeffreys prior, we will replace the distribution over $\mathbf{W}$ at each layer with the masked posterior mean during test time:

$$\hat{\mathbf{W}} = \text{diag}(\mathbf{m})\, \mathbb{E}_{q(\mathbf{z})q(\tilde{\mathbf{W}})}[\text{diag}(\mathbf{z})\tilde{\mathbf{W}}] = \text{diag}\left(\mathbf{m} \odot \exp(\boldsymbol{\mu}_z + \frac{1}{2}\boldsymbol{\sigma}^2_z)\right)\mathbf{M}_W, \tag{19}$$

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

5 Experiments

We validated the compression and speed-up capabilities of our models on the well-known architectures of LeNet-300-100 [LeCun et al., 1998a], LeNet-5-Caffe[6] on MNIST [LeCun et al., 1998b] and, similarly with [Molchanov et al., 2017], VGG [Simonyan and Zisserman, 2015][7] on CIFAR 10 Krizhevsky and Hinton [2009]. The groups of parameters were constructed by coupling the scale variables for each filter for the convolutional layers and for each input neuron for the fully connected layers. We provide the algorithms that describe the forward pass using local reparametrizations for fully connected and convolutional layers with each of the employed approximate posteriors at appendix F. For the horseshoe prior we set the scale τ_0 of the global half-Cauchy prior to a reasonably small value, e.g. $\tau_0 = 1e-5$. This further increases the prior mass at zero, which is essential for sparse estimation and compression. We also found that constraining the standard deviations as described at Louizos and Welling [2017] and "warm-up" [Sønderby et al., 2016] helps in avoiding bad local optima of the variational objective. Further details about the experimental setup can be found at Appendix A. Determining the threshold for pruning can be easily done with manual inspection as usually there are two well separated clusters (signal and noise). We provide a sample visualization at Appendix E.

5.1 Architecture learning & bit precisions

We will first demonstrate the group sparsity capabilities of our methods by illustrating the learned architectures at Table 1. We also provide the inferred bit precision per layer for the Bayesian methods, which we obtain by using the marginal posterior variances of the weights as a proxy[8]:

$$\mathbb{V}(w_{ij})_{NJ} = \sigma^2_{z_i}\left(\sigma^2_{ij} + \mu^2_{ij}\right) + \sigma^2_{ij}\mu^2_{z_i} \tag{20}$$

$$\mathbb{V}(w_{ij})_{HS} = (\exp(\sigma^2_{z_i}) - 1)\exp(2\mu_{z_i} + \sigma^2_{z_i})\left(\sigma^2_{ij} + \mu^2_{ij}\right) + \sigma^2_{ij}\exp(2\mu_{z_i} + \sigma^2_{z_i}). \tag{21}$$

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

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

[5] Empirically, it slightly better separates the scales compared to the negative log-mean $-(\mu_{z_i} + 0.5\sigma^2_{z_i})$.

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

[7] The adapted CIFAR 10 version described at `http://torch.ch/blog/2015/07/30/cifar.html`.

[8] $\mathbb{V}(w_{ij}) = \mathbb{V}(z_i\tilde{w}_{ij}) = \mathbb{V}(z_i)\left(\mathbb{E}[\tilde{w}_{ij}]^2 + \mathbb{V}(\tilde{w}_{ij})\right) + \mathbb{V}(\tilde{w}_{ij})\,\mathbb{E}[z_i]^2$.

Table 1: Learned architectures with Sparse VD [Molchanov et al., 2017], Generalized Dropout (GD) [Srinivas and Babu, 2016] and Group Lasso (GL) Wen et al. [2016]. Bayesian Compression (BC) with group normal-Jeffreys (BC-GNJ) and group horseshoe (BC-GHS) priors correspond to the proposed models. We show the amount of neurons left after pruning along with the average bit precisions for the weights at each layer.

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

5.2 Compression Rates

For the actual compression task we compare our method to current work in three different scenarios: (i) compression achieved only by pruning, here, for non-group methods we use the CSC format to store parameters; (ii) compression based on the former but with reduced bit precision per layer (only for the weights); and (iii) the maximum compression rate as proposed by Han et al. [2016]. We believe

Table 2: Compression results for our methods. "DC" corresponds to Deep Compression method introduced at Han et al. [2016], "DNS" to the method of Guo et al. [2016] and "SWS" to the Soft-Weight Sharing of Ullrich et al. [2017]. Numbers marked with * are best case guesses.

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

these to be relevant scenarios because (i) can be applied with already existing frameworks such as Tensorflow [Abadi et al., 2016], (ii) is a practical scheme given upcoming GPUs and frameworks will be designed to work with low and mixed precision arithmetics [Lin and Talathi, 2016, Gysel, 2016].

For (iii), we perform k-means clustering on the weights with k=32 and consequently store a weight index that points to a codebook of available weights. Note that the latter achieves highest compression rate but it is however fairly unpractical at test time since the original matrix needs to be restored for each layer. As we can observe at Table 2, our methods are competitive with the state-of-the art for LeNet-300-100 while offering significantly better compression rates on the LeNet-5-Caffe architecture, without any loss in accuracy. Do note that group sparsity and weight sparsity can be combined so as to further prune some weights when a particular group is not removed, thus we can potentially further boost compression performance at e.g. LeNet-300-100. For the VGG network we observe that training from a random initialization yielded consistently less accuracy (around 1%-2% less) compared to initializing the means of the approximate posterior from a pretrained network, similarly with Molchanov et al. [2017], thus we only report the latter results[9]. After initialization we trained the VGG network regularly for 200 epochs using Adam with the default hyperparameters. We observe a small drop in accuracy for the final models when using the deterministic version of the network for prediction, but nevertheless averaging across multiple samples restores the original accuracy. Note, that in general we can maintain the original accuracy on VGG without sampling by simply finetuning with a small learning rate, as done at Molchanov et al. [2017]. This will still induce (less) sparsity but unfortunately it does not lead to good compression as the bit precision remains very high due to not appropriately increasing the marginal variances of the weights.

5.3 Speed and energy consumption

We demonstrate that our method is competitive with Wen et al. [2016], denoted as GL, a method that explicitly prunes convolutional kernels to reduce compute time. We measure the time and energy consumption of one forward pass of a mini-batch with batch size 8192 through LeNet-5-Caffe. We average over 10^4 forward passes and all experiments were run with Tensorflow 1.0.1, cuda 8.0 and respective cuDNN. We apply 16 CPUs run in parallel (CPU) or a Titan X (GPU). Note that we only use the pruned architecture as lower bit precision would further increase the speed-up but is not implementable in any common framework. Further, all methods we compare to in the latter experiments would barely show an improvement at all since they do not learn to prune groups but only parameters. In figure 1 we present our results. As to be expected the largest effect on the speed up is caused by GPU usage. However, both our models and best competing models reach a speed up factor of around $8\times$. We can further save about $3\times$ energy costs by applying our architecture instead of the original one on a GPU. For larger networks the speed-up is even higher: for the VGG experiments with batch size 256 we have a speed-up factor of $51\times$.

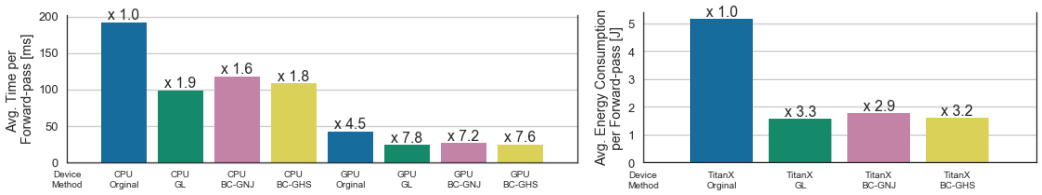


Figure 1: **Left:** Avg. Time a batch of 8192 samples takes to pass through LeNet-5-Caffe. Numbers on top of the bars represent speed-up factor relative to the CPU implementation of the original network. **Right:** Energy consumption of the GPU of the same process (when run on GPU).

6 Conclusion

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

[9]We also tried to finetune the same network with Sparse VD, but unfortunately it increased the error considerably (around 3% extra error), therefore we do not report those results.

Acknowledgments

We would like to thank Dmitry Molchanov, Dmitry Vetrov, Klamer Schutte and Dennis Koelma for valuable discussions and feedback. This research was supported by TNO, NWO and Google.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

D. F. Andrews and C. L. Mallows. Scale mixtures of normal distributions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 99–102, 1974.

A. Armagan, M. Clyde, and D. B. Dunson. Generalized beta mixtures of gaussians. In *Advances in neural information processing systems*, pages 523–531, 2011.

E. Azarkhish, D. Rossi, I. Loi, and L. Benini. Neurostream: Scalable and energy efficient deep learning with smart memory cubes. *arXiv preprint arXiv:1701.06420*, 2017.

J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

E. Beale, C. Mallows, et al. Scale mixing of symmetric distributions with zero means. *The Annals of Mathematical Statistics*, 30(4):1145–1151, 1959.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015.

C. M. Carvalho, N. G. Polson, and J. G. Scott. The horseshoe estimator for sparse signals. *Biometrika*, 97(2): 465–480, 2010.

S. Chai, A. Raghavan, D. Zhang, M. Amer, and T. Shields. Low precision neural networks using subband decomposition. *arXiv preprint arXiv:1703.08595*, 2017.

W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks. *arXiv preprint arXiv:1506.04449*, 2015.

M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to $+1$ or -1. *arXiv preprint arXiv:1602.02830*, 2016.

M. Courbariaux, J.-P. David, and Y. Bengio. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3105–3113, 2015.

M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

X. Dong, J. Huang, Y. Yang, and S. Yan. More is less: A more complicated network with less inference complexity. *arXiv preprint arXiv:1703.08651*, 2017.

M. A. Figueiredo. Adaptive sparseness using jeffreys' prior. *Advances in neural information processing systems*, 1:697–704, 2002.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *ICML*, 2016.

Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *ICLR*, 2015.

A. Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

P. D. Grünwald. *The minimum description length principle*. MIT press, 2007.

Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. *CoRR, abs/1502.02551*, 392, 2015.

P. Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *Master's thesis, University of California*, 2016.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

G. E. Hinton and D. 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*, pages 5–13. ACM, 1993.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

A. Honkela and H. Valpola. Variational learning and bits-back coding: an information-theoretic view to bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810, 2004.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *ICLR*, 2017.

J. B. Ingraham and D. S. Marks. Bayesian sparsity for intractable distributions. *arXiv preprint arXiv:1602.03807*, 2016.

T. Karaletsos and G. Rätsch. Automatic relevance determination for deep generative models. *arXiv preprint arXiv:1505.07765*, 2015.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR), San Diego*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014.

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparametrization trick. *Advances in Neural Information Processing Systems*, 2015.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

N. D. Lawrence. Note relevance determination. In *Neural Nets WIRN Vietri-01*, pages 128–133. Springer, 2002.

Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998b.

D. D. Lin and S. S. Talathi. Overcoming challenges in fixed point training of deep convolutional networks. *Workshop ICML*, 2016.

D. D. Lin, S. S. Talathi, and V. S. Annapureddy. Fixed point quantization of deep convolutional networks. *arXiv preprint arXiv:1511.06393*, 2015.

C. Louizos. Smart regularization of deep architectures. *Master's thesis, University of Amsterdam*, 2015.

C. Louizos and M. Welling. Multiplicative Normalizing Flows for Variational Bayesian Neural Networks. *ArXiv e-prints*, Mar. 2017.

D. J. MacKay. Probable networks and plausible predictions a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.

N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462*, 2017.

P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, and D. Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.

T. J. Mitchell and J. J. Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032, 1988.

D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

E. Nalisnick, A. Anandkumar, and P. Smyth. A scale mixture perspective of multiplicative noise in neural networks. *arXiv preprint arXiv:1506.03208*, 2015.

R. M. Neal. *Bayesian learning for neural networks*. PhD thesis, Citeseer, 1995.

S. E. Neville, J. T. Ormerod, M. Wand, et al. Mean field variational bayes for continuous sparse signal shrinkage: pitfalls and remedies. *Electronic Journal of Statistics*, 8(1):1113–1151, 2014.

O. Papaspiliopoulos, G. O. Roberts, and M. Sköld. A general framework for the parametrization of hierarchical models. *Statistical Science*, pages 59–73, 2007.

C. Peterson. A mean field theory learning algorithm for neural networks. *Complex systems*, 1:995–1019, 1987.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286, 2014.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

J. Rissanen. Stochastic complexity and modeling. *The annals of statistics*, pages 1080–1100, 1986.

S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. Group sparse regularization for deep neural networks. *arXiv preprint arXiv:1607.00485*, 2016.

S. Shi and X. Chu. Speeding up convolutional neural networks by exploiting the sparsity of rectifier units. *arXiv preprint arXiv:1704.07724*, 2017.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

M. Sites. Ieee standard for floating-point arithmetic. 2008.

C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. *arXiv preprint arXiv:1602.02282*, 2016.

S. Srinivas and R. V. Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint arXiv:1703.09039*, 2017.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. *ICLR*, 2017.

G. Venkatesh, E. Nurvitadhi, and D. Marr. Accelerating deep convolutional networks using low-precision and sparsity. *arXiv preprint arXiv:1610.00324*, 2016.

C. S. Wallace. Classification by minimum-message-length inference. In *International Conference on Computing and Information*, pages 72–81. Springer, 1990.

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances In Neural Information Processing Systems*, pages 2074–2082, 2016.

T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *CVPR*, 2017.

S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *ICLR*, 2017.

Appendix

A. Detailed experimental setup

We implemented our methods in Tensorflow Abadi et al. [2016] and optimized the variational parameters using Adam Kingma and Ba [2015] with the default hyperparameters. The means of the conditional Gaussian $q_\phi(\mathbf{W}|\mathbf{z})$ were initialized with the scheme proposed at He et al. [2015], whereas the log of the standard deviations were initialized by sampling from $\mathcal{N}(-9, 1e-4)$. The parameters of $q_\phi(\mathbf{z})$ were initialized such that the overall mean of $\mathbf{z}$ is ≈ 1 and the overall variance is very low ($\approx 1e-8$); this ensures that all of the groups are active during the initial training iterations.

As for the standard deviation constraints; for the LeNet-300-100 architecture we constrained the standard deviation of the first layer to be ≤ 0.2 whereas for the LeNet-5-Caffe we constrained the standard deviation of the first layer to be ≤ 0.5. The remaining standard deviations were left unconstrained. For the VGG network we constrained the standard deviations of the 64 and 128 feature map layers to be ≤ 0.1, the standard deviations of the 256 feature map layers to be ≤ 0.2 and left the rest of the standard deviations unconstrained. We also found beneficial the incorporation of "warm-up" [Sønderby et al., 2016], i.e we annealed the negative KL-divergence from the prior to the approximate posterior with a linear schedule for the first 100 epochs. We initialized the means of the approximate posterior by the weights and biases obtained from a VGG network trained with batch normalization and dropout on CIFAR 10. For our method we disabled batch-normalization during training.

As for preprocessing the data; for MNIST the only preprocessing we did was to rescale the digits to lie at the $[-1, 1]$ range and for CIFAR 10 we used the preprocessed dataset provided by Zagoruyko and Komodakis [2016].

Furthermore, do note that by pruning a given filter at a particular convolutional layer we can also prune the parameters corresponding to that feature map for the next layer. This similarly holds for fully connected layers; if we drop a given input neuron then the weights corresponding to that node from the previous layer can also be pruned.

B. Standards for Floating-Point Arithmetic

Floating points values eventually need to be represented in a binary basis in a computer. The most common standard today is the IEEE 754-2008 convention [Sites, 2008]. It defines x-bit base-2 formats, officially referred to as binary$x$, with $x \in \{16, 32, 64, 128\}$. The formats are also widely known as half, single, double and quadruple precision floats, respectively and used in almost all programming languages as a standard. The format considers 3 kinds of bits: one sign bit, w exponent bits and p precision bits.

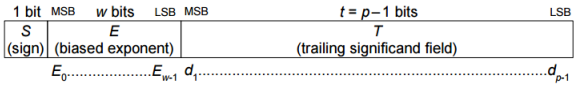


Figure 2: A symbolic representation of the binary$x$ format [Sites, 2008].

The Sign bit determines the sign of the number to be represented. The exponent E is an w-bit signed integer, e.g. for single precision $w = 8$ and thus $E \in [-127, 128]$. In practice, exponents range from

Table 3: Floating point formats

Bits per Float	Exponent width [bit]	Significand precision [bit]	underflow level	overflow level	unit roundoff
64	11	52	2.22×10^{-308}	1.79×10^{308}	2.22×10^{-16}
32	8	23	1.17×10^{-38}	3.40×10^{38}	1.19×10^{-7}
16	5	10	6.10×10^{-05}	6.54×10^{4}	9.76×10^{-4}

is smaller since the first and the last number are reserved for special numbers. The true significand or mantissa includes t bits on the right of the binary point. There is an implicit leading bit with value one. A values is consequently decomposed as follows

$$\text{mantissa} = 1 + \sum_{i=1}^{t} b_i 2^{-i} \tag{22}$$

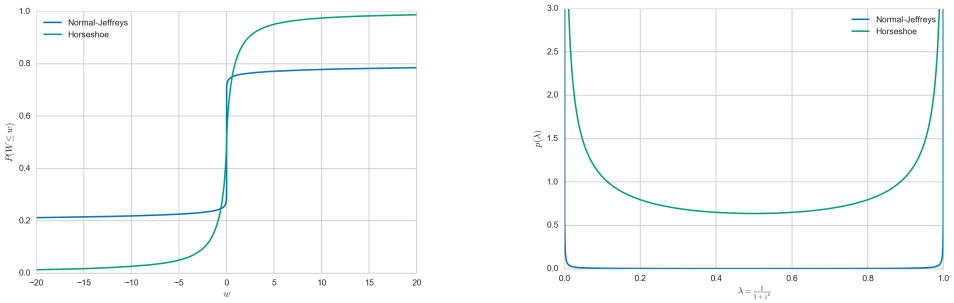
$$\text{value} = (-1)^{\text{sign bit}} \times 2^{E} \times \text{mantissa}. \tag{23}$$

In table 3, we summarize common and less common floating point formats.

There is however the possibility to design a self defined format. There are 3 important quantities when choosing the right specification: overflow, underflow and unit round off also known as machine precision. Each one can be computed knowing the number of exponent and significant bits. in our work for example we consider a format that uses significantly less exponent bits since network parameters usually vary between [-10,10]. We set the unit round off equal to the precision and thus can compute the significant bits necessary to represent a specific weight.

Beyond designing a tailored floating point format for deep learning, recent work also explored the possibility of deep learning with mixed formats [Lin and Talathi, 2016, Gysel, 2016]. For example, imagine the activations having high precision while weights can be low precision.

C. Shrinkage properties of the normal-Jeffreys and horseshoe priors



(a) Empirical CDF

(b) Prior on shrinkage coefficient

Figure 3: Comparison of the behavior of the log-uniform / normal-Jeffreys (NJ) prior and the horseshoe (HS) prior (where $s = 1$). Both priors behave similarly at zero but the normal-Jeffreys has an extremely heavy tail (thus making it non-normalizable).

In this section we will provide some insights about the behavior of each of the priors we employ by following the excellent analysis of Carvalho et al. [2010]; we can perform a change of variables and express the scale mixture distribution of eq.3 in the main paper in terms of a shrinkage coefficient, $\lambda = \frac{1}{1+z^2}$:

$$\lambda \sim p(\lambda); \qquad w \sim \mathcal{N}\left(0, \frac{1-\lambda}{\lambda}\right). \tag{24}$$

It is easy to observe that eq. 24 corresponds to a continuous relaxation of the spike-and-slab prior: when $\lambda = 0$ we have that $p(w|\lambda = 0) = \mathcal{U}(-\infty, \infty)$, i.e. no shrinkage/regularization for w, when $\lambda = 1$ we have that $p(w|\lambda = 1) = \delta(w = 0)$, i.e. w is exactly zero, and when $\lambda = \frac{1}{2}$ we have that

$p(w|\lambda = \frac{1}{2}) = \mathcal{N}(0, 1)$. Now by examining the implied prior on the shrinkage coefficient λ for both the log-uniform and the horseshoe priors we can better study their behavior. As it is explained at Carvalho et al. [2010], the half-Cauchy prior on z corresponds to a beta prior on the shrinkage coefficient, $p(\lambda) = \mathcal{B}(\frac{1}{2}, \frac{1}{2})$, whereas the normal-Jeffreys / log-uniform prior on z corresponds to $p(\lambda) = \mathcal{B}(\epsilon, \epsilon)$ with $\epsilon \approx 0$. The densities of both of these distributions can be seen at Figure 3b. As we can observe, the log-uniform prior posits a distribution that concentrates almost all of its mass at either $\lambda \approx 0$ or $\lambda \approx 1$, essentially either pruning the parameter or keeping it close to the maximum likelihood estimate due to $p(w|\lambda \approx 1) = \mathcal{U}(-\infty, \infty)$. In contrast the horseshoe prior maintains enough probability mass for the in-between values of λ and thus can, potentially, offer better regularization and generalization.

D. Negative KL-divergences for log-normal approximating posteriors

Let $q(z) = \mathcal{LN}(\mu, \sigma^2)$ be a log-normal approximating posterior. Here we will derive the negative KL-divergences to $q(z)$ from inverse gamma, gamma and half-normal distributions.

Let $p(z)$ be an inverse gamma distribution, i.e. $p(z) = \mathcal{IG}(\alpha, \beta)$. The negative KL-divergence can be expressed as follows:

$$-KL(q(z)||p(z)) = \int q(z) \log p(z) dz - \int q(z) \log q(z) dz. \tag{25}$$

The second term is the entropy of the log-normal distribution which has the following form:

$$\mathcal{H}_q = -\int q(z) \log q(z) dz = \frac{1}{2} \log \sigma^2 + \mu + \frac{1}{2} + \frac{1}{2} \log(2\pi). \tag{26}$$

The first term is the negative cross-entropy between the log-normal posterior and the inverse-Gamma prior:

$$-\mathcal{CE}_{qp} = \int q(z) \bigg(\alpha \log \beta - \log \Gamma(\alpha) - (\alpha + 1) \log z) - \frac{\beta}{z} \bigg) dz \tag{27}$$

$$= \alpha \log \beta - \log \Gamma(\alpha) - (\alpha + 1) \, \mathbb{E}_{q(z)}[\log z] - \beta \, \mathbb{E}_{q(z)}[z^{-1}]. \tag{28}$$

Since the natural logarithm of a log-normal distribution $\mathcal{LN}(\mu, \sigma^2)$ follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$ we have that $\mathbb{E}_{q(z)}[\log z] = \mu$. Furthermore we have that if $x \sim \mathcal{LN}(\mu, \sigma^2)$ then $\frac{1}{x} \sim \mathcal{LN}(-\mu, \sigma^2)$, therefore $\mathbb{E}_{q(z)}[z^{-1}] = \exp(-\mu + \frac{\sigma^2}{2})$. Putting everything together we have that:

$$-\mathcal{CE}_{qp} = \alpha \log \beta - \log \Gamma(\alpha) - (\alpha + 1)\mu - \beta \exp(-\mu + \frac{\sigma^2}{2}). \tag{29}$$

Therefore the negative KL-divergence is:

$$-KL(q(z)||p(z)) = \alpha \log \beta - \log \Gamma(\alpha) - \alpha\mu - \beta \exp(-\mu + 0.5\sigma^2) + \\ + 0.5(\log \sigma^2 + 1 + \log(2\pi)). \tag{30}$$

Now let $p(z)$ be a Gamma prior, i.e. $p(z) = \mathcal{G}(\alpha, \beta)$. We have that the negative cross-entropy changes to:

$$-\mathcal{CE}_{qp} = \int q(z) \bigg(- \alpha \log \beta - \log \Gamma(\alpha) - \frac{z}{\beta} + (\alpha - 1) \log z \bigg) dz \tag{31}$$

$$= -\alpha \log \beta - \log \Gamma(\alpha) - \beta^{-1} \, \mathbb{E}_{q(z)}[z] + (\alpha - 1) \, \mathbb{E}_{q(z)}[\log z] \tag{32}$$

$$= -\alpha \log \beta - \log \Gamma(\alpha) - \beta^{-1} \exp(\mu + \frac{\sigma^2}{2}) + (\alpha - 1)\mu. \tag{33}$$

Therefore the negative KL-divergence is:

$$-KL(q(z)||p(z)) = -\alpha \log \beta - \log \Gamma(\alpha) + \alpha\mu - \beta^{-1} \exp(\mu + 0.5\sigma^2) + \\ + 0.5(\log \sigma^2 + 1 + \log(2\pi)). \tag{34}$$

Now, by employing the aforementioned we can express the KL-divergence between $p(s_a, s_b, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}})$ and $q_\phi(s_a, s_b, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}})$ as follows:

$$-KL(q_\phi(s_a)||p(s_a)) = \log \tau_0 - \tau_0^{-1} \exp\left(\mu_{s_a} + \frac{1}{2}\sigma_{s_a}^2\right) + \frac{1}{2}\left(\mu_{s_a} + \log \sigma_{s_a}^2 + 1 + \log 2\right) \quad (35)$$

$$-KL(q_\phi(s_b)||p(s_b)) = -\exp\left(\frac{1}{2}\sigma_{s_b}^2 - \mu_{s_b}\right) + \frac{1}{2}\left(-\mu_{s_b} + \log \sigma_{s_b}^2 + 1 + \log 2\right) \quad (36)$$

$$-KL(q_\phi(\tilde{\boldsymbol{\alpha}})||p(\tilde{\boldsymbol{\alpha}})) = \sum_i^A \left(-\exp\left(\mu_{\tilde{\alpha}_i} + \frac{1}{2}\sigma_{\tilde{\alpha}_i}^2\right) + \frac{1}{2}\left(\mu_{\tilde{\alpha}_i} + \log \sigma_{\tilde{\alpha}_i}^2 + 1 + \log 2\right)\right) \quad (37)$$

$$-KL(q_\phi(\tilde{\boldsymbol{\beta}})||p(\tilde{\boldsymbol{\beta}})) = \sum_i^A \left(-\exp\left(\frac{1}{2}\sigma_{\tilde{\beta}_i}^2 - \mu_{\tilde{\beta}_i}\right) + \frac{1}{2}\left(-\mu_{\tilde{\beta}_i} + \log \sigma_{\tilde{\beta}_i}^2 + 1 + \log 2\right)\right), \quad (38)$$

with the KL-divergence for the weight distribution $q_\phi(\tilde{\mathbf{W}})$ given by eq.8 in the main paper.

E. Visualizations

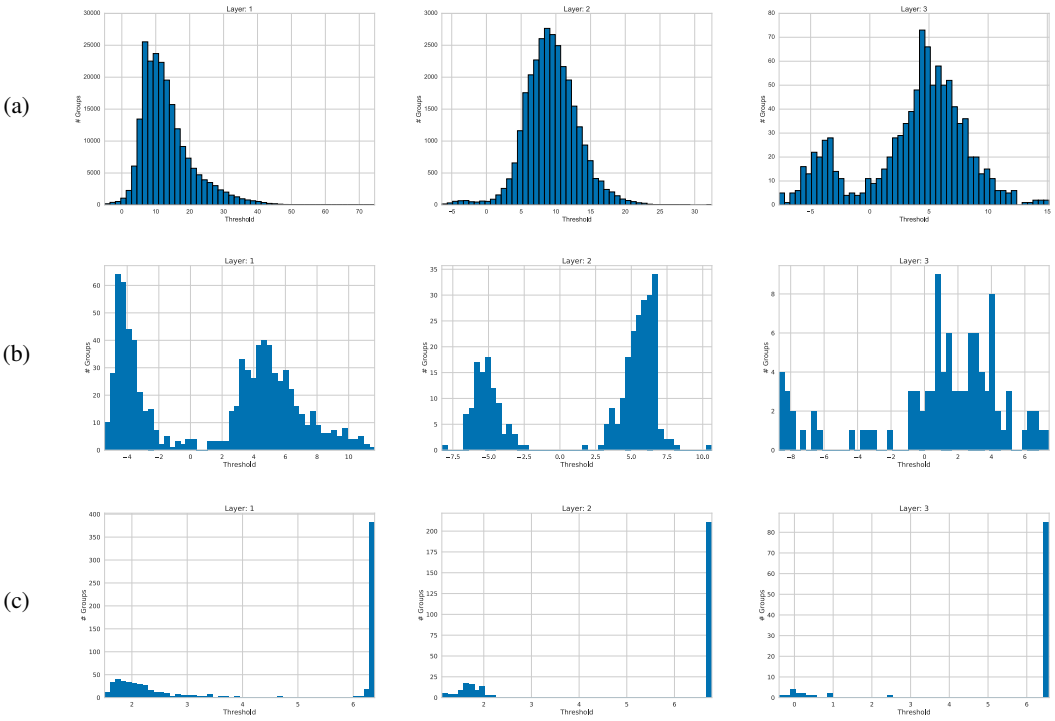


Figure 4: Distribution of the thresholds for the Sparse Variational Dropout 4a, Bayesian Compression with group normal-Jeffreys (BC-GNJ) 4b and group Horseshoe (BC-GHS) 4c priors for the three layer LeNet-300-100 architecture. It is easily observed that there are usually two well separable groups with BC-GNJ and BC-GHS, thus making the choice for the threshold easy. Smaller values indicate signal whereas larger values indicate noise (i.e. useless groups).

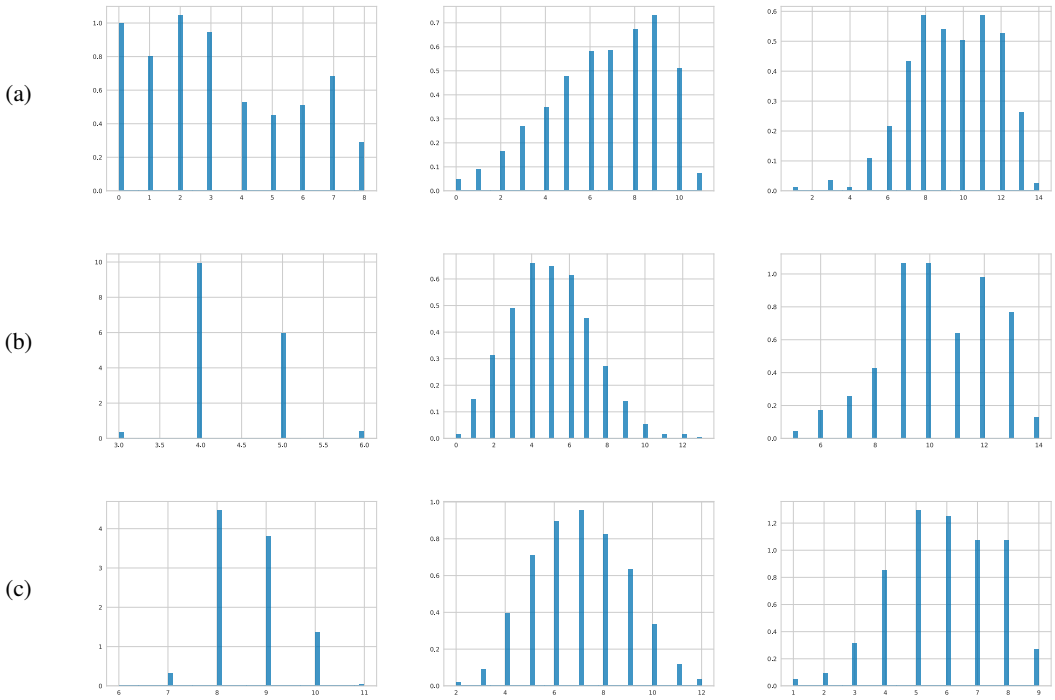


Figure 5: Distribution of the bit precisions for the Sparse Variational Dropout 5a, Bayesian Compression with group normal-Jeffreys (BC-GNJ) 5b and group Horseshoe (BC-GHS) 5c priors for the three layer LeNet-300-100 architecture. All of the methods usually require far fewer than 32bits for the weights.

F. Algorithms for the feedforward pass

Algorithms 1, 2, 3, 4 describe the forward pass using local reparametrizations for fully connected and convolutional layers with the approximate posteriors for the Bayesian Compression (BC) with group normal-Jeffreys (BC-GNJ) and group Horseshoe (BC-GHS) priors employed at the experiments. For the fully connected layers we coupled the scales for each input neuron whereas for the convolutional we couple the scales for each output feature map. $\mathbf{M}_w, \mathbf{\Sigma}_w$ are the means and variances of each layer, $\mathbf{H}$ is a minibatch of activations of size K. For the first layer we have that $\mathbf{H} = \mathbf{X}$ where $\mathbf{X}$ is the minibatch of inputs. For the convolutional layers N_f are the number of convolutional filters, $*$ is the convolution operator and we assume the [batch, height, width, feature maps] convention.

Algorithm 1 Fully connected BC-GNJ layer h.

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

$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$

$\mathbf{Z} = \boldsymbol{\mu}_z + \boldsymbol{\sigma}_z \odot \hat{\mathbf{E}}$

$\hat{\mathbf{H}} = \mathbf{H} \odot \mathbf{Z}$

$\mathbf{M}_h = \hat{\mathbf{H}}\mathbf{M}_w$

$\mathbf{V}_h = \hat{\mathbf{H}}^2\mathbf{\Sigma}_w$

$\mathbf{E} \sim \mathcal{N}(0, 1)$

return $\mathbf{M}_h + \sqrt{\mathbf{V}_h} \odot \mathbf{E}$

Algorithm 2 Convolutional BC-GNJ layer h.

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

$\mathbf{M}_h = \mathbf{H} * \mathbf{M}_w$

$\mathbf{V}_h = \mathbf{H}^2 * \mathbf{\Sigma}_w$

$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$

$\hat{\boldsymbol{\mu}}_z = \text{reshape}(\boldsymbol{\mu}_z, [K, 1, 1, N_f])$

$\hat{\boldsymbol{\sigma}}_z = \text{reshape}(\boldsymbol{\sigma}_z, [K, 1, 1, N_f])$

$\mathbf{Z} = \hat{\boldsymbol{\mu}}_z + \hat{\boldsymbol{\sigma}}_z \odot \hat{\mathbf{E}}$

$\mathbf{E} \sim \mathcal{N}(0, 1)$

return $\mathbf{M}_h \odot \mathbf{Z} + \sqrt{\mathbf{V}_h \odot \mathbf{Z}^2} \odot \mathbf{E}$

Algorithm 3 Fully connected BC-GHS layer h.

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

$\hat{\boldsymbol{\epsilon}} \sim \mathcal{N}(0, 1)$

$\mu_s = .5\mu_{s_a} + .5\mu_{s_b}$

$\sigma_s = \sqrt{.25\sigma^2_{s_a} + .25\sigma^2_{s_b}}$

$\log \mathbf{s} = \mu_s + \sigma_s \odot \hat{\boldsymbol{\epsilon}}$

$\boldsymbol{\mu}_{\tilde{z}} = .5\boldsymbol{\mu}_{\tilde{\alpha}} + .5\boldsymbol{\mu}_{\tilde{\beta}} + \log \mathbf{s}$

$\boldsymbol{\sigma}_{\tilde{z}} = \sqrt{.25\boldsymbol{\sigma}^2_{\tilde{\alpha}} + .25\boldsymbol{\sigma}^2_{\tilde{\beta}}}$

$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$

$\mathbf{Z} = \exp(\boldsymbol{\mu}_{\tilde{z}} + \boldsymbol{\sigma}_{\tilde{z}} \odot \hat{\mathbf{E}})$

$\hat{\mathbf{H}} = \mathbf{H} \odot \mathbf{Z}$

$\mathbf{M}_h = \hat{\mathbf{H}}\mathbf{M}_w$

$\mathbf{V}_h = \hat{\mathbf{H}}^2\mathbf{\Sigma}_w$

$\mathbf{E} \sim \mathcal{N}(0, 1)$

return $\mathbf{M}_h + \sqrt{\mathbf{V}_h} \odot \mathbf{E}$

Algorithm 4 Convolutional BC-GHS layer h.

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

$\mathbf{M}_h = \mathbf{H} * \mathbf{M}_w$

$\mathbf{V}_h = \mathbf{H}^2 * \mathbf{\Sigma}_w$

$\hat{\boldsymbol{\epsilon}} \sim \mathcal{N}(0, 1)$

$\mu_s = .5\mu_{s_a} + .5\mu_{s_b}$

$\sigma_s = \sqrt{.25\sigma^2_{s_a} + .25\sigma^2_{s_b}}$

$\log \mathbf{s} = \text{reshape}(\mu_s + \sigma_s \odot \hat{\boldsymbol{\epsilon}}, [K, 1, 1, 1])$

$\boldsymbol{\mu}_{\tilde{z}} = \text{reshape}(.5\boldsymbol{\mu}_{\tilde{\alpha}} + .5\boldsymbol{\mu}_{\tilde{\beta}}, [K, 1, 1, N_f])$

$\boldsymbol{\sigma}_{\tilde{z}} = \text{reshape}(\sqrt{.25\boldsymbol{\sigma}^2_{\tilde{\alpha}} + .25\boldsymbol{\sigma}^2_{\tilde{\beta}}}, [K, 1, 1, N_f])$

$\hat{\mathbf{E}} \sim \mathcal{N}(0, 1)$

$\mathbf{Z} = \exp(\boldsymbol{\mu}_{\tilde{z}} + \log \mathbf{s} + \boldsymbol{\sigma}_{\tilde{z}} \odot \hat{\mathbf{E}})$

$\mathbf{E} \sim \mathcal{N}(0, 1)$

return $\mathbf{M}_h \odot \mathbf{Z} + \sqrt{\mathbf{V}_h \odot \mathbf{Z}^2} \odot \mathbf{E}$