Post-Training Neural Network Compression With Variational Bayesian Quantization

Zipei Tan
Massachusetts Institute of Technology
Cambridge, MA 02139
zipeitan@mit.edu

Robert Bamler
University of Tuebingen
Tuebingen, Baden-Wuerttemberg, Germany
robert.bamler@uni-tuebingen.de

Abstract

Neural network compression can open up new deployment schemes for deep learning models by making it feasible to ship deep neural networks with millions of parameters directly within a mobile or web app rather than running them on a remote data center, thus reducing server costs, network usage, latency, and privacy concerns. In this paper, we propose and empirically evaluate a simple and generic compression method for trained neural networks that builds on variational inference and on the Variational Bayesian Quantization algorithm [Yang et al., 2020]. We find that the proposed method achieves significantly lower bit rates than existing post-training compression methods at comparable model performance. The proposed method demonstrates a new use case of Bayesian neural networks (BNNs), and we analyze how compression performance depends on the temperature of a BNN.

1 Introduction

Many popular software products make use of deep neural networks for some of their functionality. Today, such deep-learning based functionality is often deployed as a service that runs in central data centers to which each end-user's device makes a remote procedure call over the internet each time the application needs to evaluate some model on some new data point. Such hybrid client/server deployments are popular as they allow operators to collect additional training data and because deep learning models are often too big to be realistically pushed to the client side.

However, deploying a deep neural network on the client side rather than on a server can also have significant advantages. First, executing expensive neural network operations directly on the end-user's device (e.g., smartphone or personal computer) reduces server costs. Second, eliminating the need for an internet round trip for every single model evaluation reduces latency and power consumption, and allows users to use an application regardless of internet connectivity. Finally, running deep learning models locally on the end-user's device eases privacy concerns, which can open up new opportunities for deep-learning applications in highly regulated areas such as healthcare or education.

In this work, we consider one major obstacle that arises when deploying deep neural networks to end-user devices: the file size of the trained network. Powerful deep neural networks have millions of parameters (or 'weights') and are thus expensive to store in their raw form. Several methods have been proposed to reduce network size (see Section 2 below). However, many of these methods focus on reducing the cost of computations rather than storage (often by *pruning*, i.e., completely discarding small weights), or they impose overly simplistic global constraints (e.g., layer-wise numerical precisions), resulting in a sudden drop of model performance as the bit rate decreases.

We propose and evaluate a simple alternative lossy compression method for deep neural networks, and we find that it retains significantly higher model performance at low bit rates (see main results in Figure 1a in Section 4). The method uses Variational Bayesian Quantization (VBQ) [Yang et al.,

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2020] to optimize a rate/distortion trade-off, taking into account not only the magnitude of network weights but also the network's sensitivity to distortions in each individual weight. VBQ was originally proposed for compressing latent representations in graphical models [Yang et al., 2020]. In this work, we extend VBQ to neural network compression by recasting the model as a Bayesian neural network (BNN) [Wilson and Izmailov, 2020]. Surprisingly, we find that, for optimal compression performance, one has to artificially inflate the posterior uncertainty estimates ("increase the temperature" of the BNN). This observation exhibits a contrast to traditional applications of BNNs, where one often has to *decrease* the temperature to obtain best performance [Wenzel et al., 2020].

Our proposed compression method can be applied to a wide variety of network architectures and requires no modification of the network other than recasting it as a BNN, which can be done in a generic way as described in Section 3 below. Only a single additional hyperparameter (the temperature T) is introduced at training time; once the BNN is trained, model compression via VBQ introduces one more hyperparameter that trades off between model performance and compression strength, and that can be tuned easily as each run of VBQ takes only a few seconds.

2 Related Work

Related work falls into the categories of Bayesian Neural networks and network compression/pruning.

Bayesian Neural Networks. Our proposed method builds on Bayesian neural networks (BNNs) trained with variational inference (VI) [Blei et al., 2017, Zhang et al., 2018], i.e., the Bayes by Backprop algorithm [Blundell et al., 2015, Shridhar et al., 2019]. While most work on BNNs discusses their generalization capability [Wilson and Izmailov, 2020], the present paper shows that BNNs can be used for the different task of network compression. BNN training introduces a scalar hyperparameter T called 'temperature'. While empirical evidence [Wenzel et al., 2020] suggests that uncompressed BNNs perform best for $T < 1$, we find that compression works best for $T > 1$.

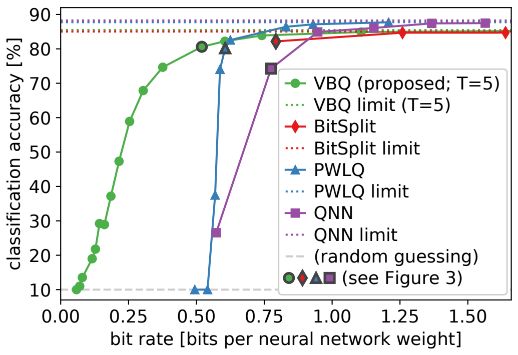
Network Compression/Pruning. There are two approaches to neural network compression [Neill, 2020]: compression during training and post-training compression. Compression during training through pruning can be done using variational dropout [Kingma et al., 2015, Molchanov et al., 2017]. Quantization is another compression method during training and it ranges from training binary [Gupta et al., 2015] or ternary [Courbariaux et al., 2014] networks to more complicated models [Achterhold et al., 2018, Van Baalen et al., 2020]. Post-training compression methods include piecewise linear quantization [Fang et al., 2020], BitSplit [Wang et al., 2020], and a greedy path-following quantization algorithm [Zhang et al., 2022]. These methods do not exploit posterior uncertainty estimates, and we find in our experiments in Section 4 that they perform significantly worse than our proposed method.

3 Method

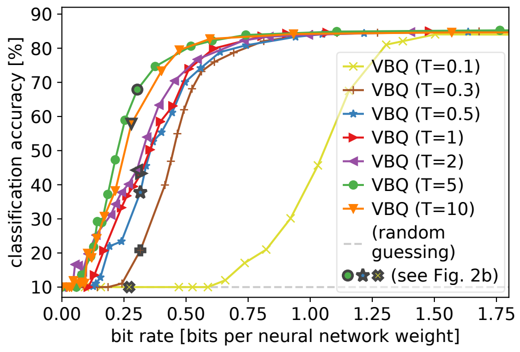
The lossy compression method for deep neural networks studied in this paper follows three steps: (1) model training with Variational Inference (VI) [Blei et al., 2017, Zhang et al., 2018]; (2) quantization with Variational Bayesian Quantization (VBQ) [Yang et al., 2020]; and (3) entropy coding.

Step 1: Model Training With VI. The VBQ algorithm used in step 2 below was originally proposed for compressing latent variables in probabilistic generative models. In order to extend VBQ to neural network compression, we first recast the network as a Bayesian neural network (BNN) [Blundell et al., 2015, Shridhar et al., 2019], which we then train with VI using the so-called Gaussian mean-field approximation [Blei et al., 2017]. This can be achieved with a simple generic transformation of the neural network where one injects random Gaussian noise at each layer and then learns optimal noise amplitudes (denoted $\boldsymbol{\sigma}$ below) together with the mean neural network weights (denoted $\boldsymbol{\mu}$ below).

In detail, assuming a neural network with weights $\mathbf{w} \equiv (w_\alpha)_\alpha$, training data points $\mathbf{x} \equiv (x_i)_{i=1}^N$, and a loss function of the form $\mathcal{L}(\mathbf{w}) = \sum_{i=1}^N \big(\ell(\mathbf{w}, x_i) + \psi(\mathbf{w})\big)$ where ψ is a regularizer, we reinterpret $\ell(\mathbf{w}, x_i)$ as a negative log-likelihood and $\psi(\mathbf{w})$ as a negative log-prior. VI with Gaussian mean-field



(a) Performance of VBQ compared to baselines.



(b) Performance of VBQ with various temperatures T.

Figure 1: Model performances at varying bit rates. VBQ (green) performs well even at low bit rates.

approximation then amounts to stochastic minimization of the following loss function over $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$,

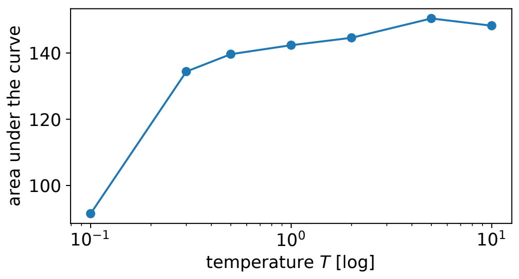
$$\mathcal{L}_T^{\mathrm{VI}}(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{i=1}^{N} \mathbb{E}_{\mathbf{w}\sim\mathcal{N}(\boldsymbol{\mu},\,\mathrm{diag}(\boldsymbol{\sigma}^2))} \left[\ell(\mathbf{w}, x_i) + T \left(\psi(\mathbf{w}) - \frac{1}{N} \sum_{\alpha} \log \sigma_\alpha \right) \right]. \tag{1}$$

Here, the temperature $T > 0$ is a hyperparameter (discussed in Sectino 4 below), $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are each of the same dimension as $\mathbf{w}$, and the notation $\mathbb{E}_{\mathbf{w}\sim\mathcal{N}(\boldsymbol{\mu},\,\mathrm{diag}(\boldsymbol{\sigma}^2))}[\,\cdot\,]$ denotes that each weight w_α is drawn at random in each gradient step from a normal distribution with mean μ_α and variance σ_α^2.

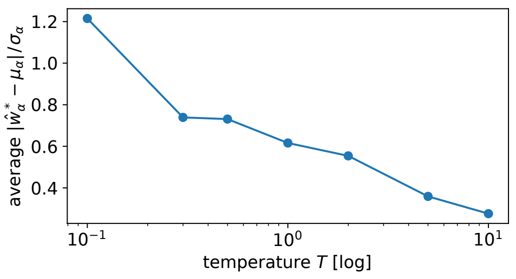
Step 2: Quantization With VBQ. Recasting the neural network as a BNN in step 1 above allows us to quantize it using the VBQ algorithm [Yang et al., 2020], which has previously only been used for compressing images and simple graphical models. Quantization approximates the *continuous* neural network weights with points from some *discrete* grid. VBQ uses an adaptive grid spacing that automatically adjusts to how sensitive the overall model performance is to each individual weight. Such an adaptive grid can be particularly beneficial in deep neural networks that contain both convolutional and fully connected layers, as each single weight in a convolutional layer affects many output activations, thus likely warranting a finer quantization than a weight in a fully connected layer.

VBQ estimates the relative sensitivities to each weight without having to refer back to training data by exploiting the posterior uncertainty estimates $\boldsymbol{\sigma}$ obtained from the minimization of Eq. 1. This leads to a very fast quantization method that can be run at various bitrate-settings without having to retrain the model. VBQ obtains a quantization $\hat{w}_\alpha^* = \arg\min_{\hat{w}_\alpha} \mathcal{L}_\lambda(\hat{w}_\alpha)$ for each weight w_α by minimizing a rate/distortion trade-off $\mathcal{L}_\lambda(\hat{w}_\alpha) = \lambda\mathcal{R}(\hat{w}_\alpha) + \mathcal{D}(\hat{w}_\alpha)$, where the minimization over $\hat{w}_\alpha$ runs over a (dense) discrete set of grid points, $\mathcal{R}(\hat{w}_\alpha)$ estimates how much a candidate grid point $\hat{w}_\alpha$ would contribute to the total bit rate in step 3 below (taking into account the required grid spacing to represent $\hat{w}_\alpha$); the distortion term $\mathcal{D}(\hat{w}_\alpha) = (\hat{w}_\alpha - \mu_\alpha)^2/(2\sigma_\alpha^2)$ approximates the negative log-posterior distribution and encourages small quantization errors $|\hat{w}_\alpha - \mu_\alpha|$ for weights with low posterior uncertainty σ_α; and the Lagrange parameter $\lambda > 0$ controls the trade-off between bit rate and distortion. An efficient algorithm for minimizing $\mathcal{L}_\lambda(\hat{w}_\alpha)$ was given in [Yang et al., 2020].

Step 3: Entropy Coding. The quantized weights can be encoded into a bit string using lossless entropy coding [MacKay, 2003], e.g., range coding [Pasco, 1976, Rissanen and Langdon, 1979] or asymmetric numeral systems [Duda et al., 2015, Bamler, 2022a]. Entropy coding requires a model of the frequencies with which each grid point occurs. We use each layer's empirical frequencies of quantized weights here. A real deployment would have to store a table of these empirical frequencies for each layer alongside the compressed weights, incurring an overhead that is negligible unless the model has only a handful of weights in each layer. The theoretically optimal expected bit rate (i.e., length of the resulting bit string) per weight is the entropy (to base 2) of the empirical frequency distribution [MacKay, 2003]. Open-source implementations of entropy coders [Ballé et al., 2022, Bamler, 2022b] reach this theoretical limit up to a negligible overhead ($< 0.1\,\%$ [Bamler, 2022a]).



(a) Areas under rate/distortion curves in Figure 1b (integrated from 0 to 2.0 bits per network weight).



(b) Average ratio of quantization error to weight uncertainty σ_α in VBQ for highlighted models in Figure 1b.

Figure 2: Compression performance and deviations for VBQ with different temperatures

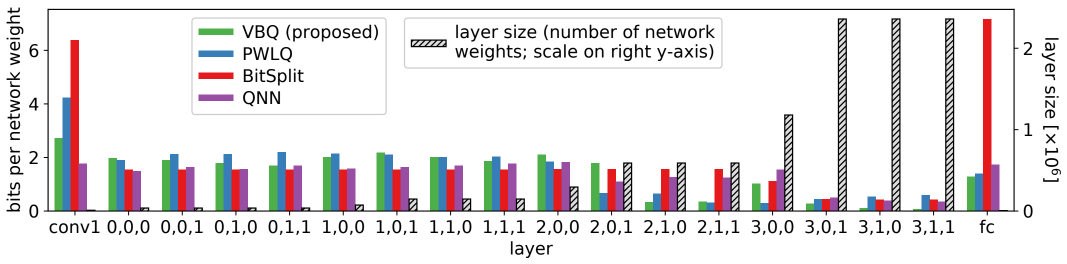


Figure 3: Average number of bits per network weight (colored bars) for highlighted models in Figure 1a. Gray hatched bars show layer sizes. Layers labeled according to [He et al., 2016]

4 Results

We empirically compare network compression methods and analyze VBQ at varying temperatures.

Experimental Setup. We analyze compression performances for a ResNet18 model taken from `PyTorch.torchvision` trained on CIFAR-10. For this model, we compare the proposed method to BitSplit [Wang et al., 2020], pointwise linear quantization (PWLQ) [Fang et al., 2020], and quantized neural nets (QNN) [Zhang et al., 2022]. Here, BitSplit required modifications to the model following the method's proposal in [Wang et al., 2020]. See appendix for more details and code.

Results. Figure 1a shows classification accuracy as a function of bit rate. We observe that the proposed method (green) retains a nontrivial classification accuracy up to low bit rates. By contrast, the accuracy with PWLQ and QNN drops sharply at ~ 0.5 bits per network weight, and BitSplit could not reach lower bit rates at all within its supported settings. Figure 1b shows that VBQ compression performance improves as temperature T increases, before it starts to decrease again. This trend is shown more quantitatively in Figure 2a and might be explained by Figure 2b, which shows that increasing T decreases the ratio of quantization error $|\hat{w}^*_\alpha - \mu_\alpha|$ to parameter uncertainty σ_α, i.e., the training procedure explores a wider range of quantization candidates. Figure 3 shows that bit rate savings in VBQ compared to baseline models with comparable accuracy occur mostly in the last three convolutional layers, which contain significantly more network weights (see gray bars).

Conclusions and Outlook. We proposed and empirically evaluated a compression method for deep neural networks. The method could make it feasible to deploy large machine learning models on the client side. In future work, it would be interesting to evaluate how rate/distortion performance changes if uncertainty estimates are obtained via Laplace approximation instead of variational inference.

References

Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *International Conference on Learning Representations*, 2018.

Johannes Ballé, Sung Jin Hwang, and Eirikur Agustsson. TensorFlow Compression: Learned data compression, 2022. URL `http://github.com/tensorflow/compression`.

Robert Bamler. Understanding entropy coding with asymmetric numeral systems (ANS): a statistician's perspective. *arXiv preprint arXiv:2201.01741*, 2022a.

Robert Bamler. constriction library: entropy coders for research and production, 2022b. URL `https://bamler-lab.github.io/constriction/`.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

Jarek Duda, Khalid Tahboub, Neeraj J Gadgil, and Edward J Delp. The use of asymmetric numeral systems as an accurate replacement for huffman coding. In *2015 Picture Coding Symposium (PCS)*, pages 65–69. IEEE, 2015.

Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun. Post-training piecewise linear quantization for deep neural networks. In *European Conference on Computer Vision*, pages 69–86. Springer, 2020.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015.

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

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

James O' Neill. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.

Richard Clark Pasco. *Source coding algorithms for fast data compression*. PhD thesis, Stanford University CA, 1976.

Jorma Rissanen and Glen G Langdon. Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162, 1979.

Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*, 2019.

Mart Van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. *Advances in neural information processing systems*, 33:5741–5752, 2020.

Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. Towards accurate post-training network quantization via bit-split and stitching. In *International Conference on Machine Learning*, pages 9847–9856. PMLR, 2020.

Florian Wenzel, Kevin Roth, Bastiaan Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the bayes posterior in deep neural networks really? In *International Conference on Machine Learning*, pages 10248–10259. PMLR, 2020.

Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

Yibo Yang, Robert Bamler, and Stephan Mandt. Variational bayesian quantization. In *International Conference on Machine Learning*, pages 10670–10680. PMLR, 2020.

Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

Jinjie Zhang, Yixuan Zhou, and Rayan Saab. Post-training quantization for neural networks with provable guarantees. *arXiv preprint arXiv:2201.11113*, 2022.

5 Appendix

5.1 Training Details

All experiments used the same dataset: PyTorch CIFAR10 dataset, 60000 32x32 colour images in 10 classes, with 6000 images per class; 50000 training images and 10000 test images. Training details for deterministic model: L_2 regularization through weight decay of 0.05; 200 epochs; learning rate 0.001; batch size 128; number of workers 8; accuracy on CIFAR10 87.75% Training details for Bayesian model (for our proposed compression method): L_2 regularization through weight decay of 1×10^{-4}; 200 epochs; learning rate 0.001; batch size 128; number of workers 8; accuracy on CIFAR10 85.41 when temperature is 1 (scaling factor of the KL divergence term) For the baseline methods, we tuned the regularization weight for each model, resulting in $L_2 = 0.01$ for bit-split, $L_2 = 0.05$ for piecewise linear, $L_2 = 0.05$ for Quantize Neural Nets; the batch size was 128 for all of these.

5.2 Code

Our code is available at: `https://anonymous.4open.science/r/bnn_vbq_submission-3F7B`