Model compression using Bayesian Deep Learning

Devashish Prasad
Department of Computer Science
Purdue University
West Lafayette, IN 47096
prasadd@purdue.edu

Abstract

A prominent work on Bayesian Neural Network [2] introduces a type of network that is relatively easy to train and achieves great accuracy results when compared to normal SGD-based approaches. Apart from predicting the uncertainty, which is the primary reason for which authors had introduced their approach, they also claim that their network achieves great accuracy even after pruning 90% of the weights. Such an impressive result motivates us to dive deep into the approach and know its true efficacy by truly comparing it with other baseline methods. In this paper, we do a systematic analysis of the Bayes by Backprop method for model compression by reproducing it with several other baseline methods like vanilla stochastic gradient descent (SGD) and Dropout based SGD under the same training and testing environment. Our results show that Bayes by Backprop indeed outperforms all other methods by a huge margin. However, it pays a really heavy computational cost in terms of wall time but needs fewer epochs to fit the data. You can find the code at this link (https://github.com/DevashishPrasad/bayesian-nn).

1 Introduction

Deep learning has achieved unprecedented results on the most difficult problems in the world. However, the practicality of deep learning-based systems is often hindered because of the computational requirements. Model compression and computational efficiency in deep learning have become a problem of great significance. The existing approaches for deep learning model compression are often divided into five categories - pruning, quantization, knowledge distillation, neural architecture search, and others [4] [3] [9]. Out of these five categories, weight pruning is the most widely used because of its simplicity and efficacy. And also mainly because weight pruning works very well in practice as it can be applied to any neural network architecture as [1] suggests. Moreover, a lot of open-sourced deep learning packages like PyTorch support weight pruning off the shelf. But recently, some of the recent papers show that we can also use Bayesian deep learning techniques to prune large parts of the network [2] [7] [8]. But so far not much attention has been given to the Bayesian deep learning techniques. And hence, in this project, we want to systematically apply Bayesian deep learning techniques for deep learning model compression and evaluate its efficacy by comparing it with the weight pruning technique mentioned above. The end goal of the project is to get a fair comparison of Bayesian deep learning-based compression with weight pruning-based model compression techniques.

The primary motivation for applying Bayesian techniques for deep neural network compression tasks is the results reported by [2] [7] [8]. All of them show that using Bayesian deep learning methods, the resulting model weight can be pruned down up to 90% without sacrificing any significant error

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

rate or accuracy. [2] considers a Bayesian sparsity enforcing prior on the weights. They then prune down the weights that are closer to 0 after training with such prior. [7] argue that network pruning and reducing bit precision for the weights is aligned with the Bayesian perspective 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 (which leads to removing insignificant bits). They elaborate more on this in their paper. [8] proposes a novel Bayesian Optimization framework for neural network compression that assists in the speed and accuracy of the compression process.

In the next section, we present a more detailed literature survey to deeply investigate the techniques used by these papers. We discuss more details about [2] and [7] in the next sections of this report. [2] and [7]'s work align with each other very well and [7] directly continues on the weakness of [2]. However, in our experiments, we could not reproduce [7] and [8], and also to keep our analysis simple we keep both of these approaches for future work. We focus primarily on comparing [2] with [6] in this project. We will also present a brief literature survey of weight pruning.

Next, we carry out our experiments. The Bayesian approach [2] has not compared their methods for model compression performance with any of the other methods like Stochastic Gradient Descent (SGD) baselines. Hence, our main aim is to understand their methods more thoroughly and reproduce their experiments to fairly compare them with the prominent weight pruning methods. We also plan to focus on the inference speed and scalability of these methods in our evaluation as opposed to the three probabilistic papers that only report the evaluation in terms of the number of parameters. We have mentioned more details about our experiments in the last section of this report.

2 Literature Review

2.1 Weight uncertainty in neural networks [2]

In this paper, the authors introduce an efficient, principled, and backpropagation-compatible algorithm for learning a probability distribution on the weights of a neural network, called Bayes by Backprop. Essentially, their method can learn all weights in our neural networks as probability distributions over possible values, rather than learning a single fixed value (point estimate) as is the norm. This introduces uncertainty in model weights and has advantages such as 1) regularisation via a compression cost on the weights, and 2) richer representations and predictions from cheap model averaging.

After giving a brief background, the authors first explain how we train and formulate traditional non-bayesian Neural Networks which they call Point estimates of Neural Networks. Here, for classification, we learn a categorical distribution while for regression we learn a Gaussian distribution. Weights of the neural network are learned by maximum likelihood estimation which is typically achieved by gradient descent (e.g., backpropagation). And regularisation can be introduced by placing a prior upon the weights w and finding the maximum a posteriori (MAP) weights.

Next, they introduce their Bayesian approach. Bayesian inference for neural networks calculates the posterior distribution of the weights given the training data, P(w|D). Taking an expectation under the posterior distribution on weights is equivalent to using an ensemble of an uncountably infinite number of neural networks. Unfortunately, this is intractable for neural networks of any practical size. And so we need to employ approximations in order to get posterior distribution on weights. Variational inference operates by constructing a new distribution q(w|θ), over the weights w and parameterized by θ, that approximates the true posterior P(w|D). More concretely, it finds the setting of the parameters which minimizes the KL-divergence between q to P.

In the paper [2], the authors derive the cost function which is a balance between having a variational posterior that is close to the chosen prior, yet also able to explain the complexity of the data well. But we need to compute gradients of this cost function in order to learn the distribution. And so, the authors derive Proposition 1 which enables us to get a backpropagation-like algorithm. This is the obtained algorithm for variational Bayesian inference in neural networks – Bayes by Backprop –

which uses unbiased estimates of gradients of the cost to learn a distribution over the weights of a neural network.

Next, the authors present and explain the complete algorithm which is based on Gaussian priors and posteriors. The crucial part of the algorithm is the choice of the Gaussian prior. And hence authors suggest using a scale mixture of two Gaussian densities as the prior combined with a diagonal Gaussian posterior. Such a prior has several advantages and has empirically shown that it works much better for learning weights distribution which can enable us to prune it later. Its property of having a heavier tail and a spike at the center or mean also helps while pruning. For these reasons, this prior resembles and is called a spike-and-slab prior. Finally, they write the cost and algorithm in terms of mini-batches. They also come up with a strategy to re-weight the two terms of the KL cost function such that the first few mini-batches are heavily influenced by the complexity cost (KL term), whilst the later mini-batches are largely influenced by the data (data term).

They carry out the experiments by training their method on both classification and regression tasks. The accuracy and error-based comparison is done with the SGD and drop-out-based baselines. In table 2 of their paper [2], they show classification errors after different amounts of Weight pruning. It is very impressive that even after pruning 98% of weights the network attains a great score. However, the authors did not show how SGD or Dropout baselines perform in the same experiment. And lastly, there is no comparison study showing results for comparison with other model compression strategies such as weight pruning. These are the key things we will try to answer in this paper.

2.2 Bayesian Compression for Deep Learning [7]

This paper directly improves upon the weaknesses of [2] i.e. the chosen gaussian scale mixture prior. They propose hierarchical priors to prune nodes instead of individual weights, and propose using 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.

They first present the relation between Variational Bayes and Minimum Description Length. 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) and the data misfit (error cost or Data term of cost) with the minimum number of bits. They formulate equation 2 in their paper [7]. This way we get a single cost function that can help us optimize for both, pruning weight and choosing lower bit precision.

The major focus of the paper is on the new priors that they propose. First, they claim that several previously proposed priors have various disadvantages. And specifically, the spike-and-slab prior which is used in many previous works including [2] has the disadvantage of computationally expensive inference since we have to explore a space of 2^M models, where M is the number of the model parameters. Hence, to solve the disadvantages, they propose choosing two priors called the hyperparameter free log-uniform prior and the half-Cauchy prior, which results in a horseshoe distribution.

Next, they show how we can use both of these priors to achieve our goal of compressed neural network architecture with respect to weight pruning and lower bit precision. They show the reparameterization of log-uniform prior and how we can group horseshoe with half-Cauchy scale priors to achieve the final bit precision for the entire weight matrix. They also very well demonstrate their method's effectiveness using experiments using several prominent CNN architectures and comparing obtained compression rates between their algorithm and other methods.

2.3 Weight pruning in Neural Networks [6]

Neural network pruning is the task of reducing the size of a network by removing parameters and has been the subject of a great deal of work in recent years. Typically, the initial network is large and accurate, and the goal is to produce a smaller network with similar accuracy. Pruning has been used since the late 1980s but has seen an explosion of interest in the past decade thanks to the rise of

deep neural networks. [1] discuss more details about various pruning strategies and how the field has evolved.

In this paper, we use the [6] algorithm which was one of the first approaches that demonstrated great success in applying pruning to neural networks. A lot of subsequent works have cited and used the algorithm proposed by [6]. The algorithms begin with training a very large neural network with random initialization. Unlike conventional training, however, we are not learning the final values of the weights, but rather we are learning which connections are important. The second step involves pruning the low-weight connections. There are several strategies involved in this step and we use pruning based on the L1 norm of the weights. The authors use a threshold to remove all connections with weights below it. This converts a dense network into a sparse network. But in our approach, instead of using a threshold we sort all the weights based on the L1 norm in increasing order and prune a certain percentage of the lowest weights. The final step retrains the network to learn the final weights for the remaining sparse connections. The authors mention that if the pruned network is used without retraining, accuracy is significantly impacted. However, we do not carry out this final step because it will not be a fair comparison with Bayes by Backprop because they only prune their network once and do not fine-tune afterward.

Lastly, overall there are two high-level strategies to prune the weights called structured pruning and unstructured pruning. Structured pruning aims at pruning the entire structures of the network (parameters in groups) such as removing entire neurons, filters, or channels to exploit hardware and software optimized for dense computation. On the other hand, the unstructured pruning strategies remove individual weight parameters. In this paper, we consider unstructured pruning strategies because the [2] follow this strategy and they don't cut whole neurons. Thus for a fair comparison, we do unstructured weight pruning with our baselines too.

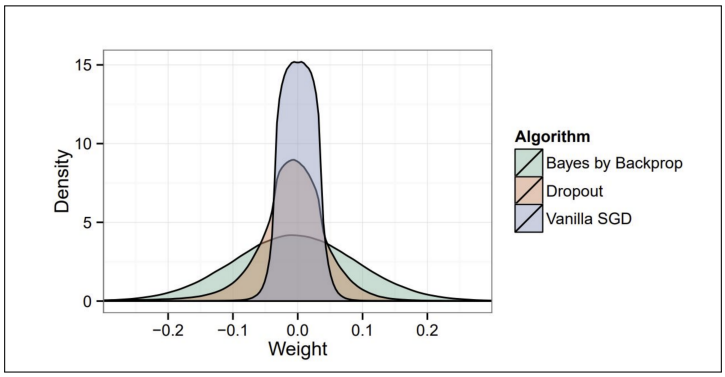

Figure 1: The figure taken from [2] shows the distribution of weights (individual parameters) of three approaches i.e. Bayes by Backprop, Dropout, and Vanilla SGD. We notice that the distribution of Bayes by Backprop has the highest variance while dropout has lesser and Vanilla SGD has the least.

3 Method

The primary basis for our approach is the figure that [2] presented in their paper in which they showed the distribution of weights (individual parameter values) of the three networks, Bayes by Backprop, Dropout, and Vanilla SGD. The authors argued that the reason behind their Bayes by Backprop Bayesian network's great accuracy even after removing 90% of weights is that their network has a very high variance in the distribution of weights. The motivation for our approach is that [2] did not perform pruning on the other two methods (Dropout and Vanilla SGD). And they also did not try different dropout rates (rate or probability of an element to be zeroed). We can see in the figure that Bayes by Backprop achieves the highest variance in the distribution of parameters of the model. This is a convincing result because of the Gaussian prior that is enforced on each of the weights. But the interesting result is the one with Dropout. We can see from the result that if we apply dropout to the vanilla SGD, then it distributes the weights and gets a higher variance in the weight distribution than

the vanilla SGD. [2] used a dropout rate of 0.5 and didn't try any rate higher than 0.5. And thus, our approach and motivation are using a dropout rate higher than 0.5 such that it will enable the neural network to get a much higher variance for its weights distribution.

In this work, for Bayes by BackProp we follow exact details and reproduce the results. And then as the authors mentioned that they used the lowest signal-to-noise ratio ($|\mu_i|/\sigma_i$) used to prune parameters, we do the same. For SGD and Dropout baselines, we prune the parameters after training based on the lowest L1-norm. And as mentioned previously, we do not fine-tune after pruning.

4 Experiments

For experiments, we use two classification datasets FMNIST [10] and MNIST [5]. Both FMNIST and MNIST both have 60000 training images and 10000 testing images of size 28x28. We do not carry out any data augmentation and we keep the mini-batch size at 100.

We reproduce the Bayes by Backprop using the scale mixture Gaussian prior on a relatively smaller network than the original paper. We show the visualization of the scale mixture prior in Fig. 2. We use a network with two layers with five hundred neurons each across all three approaches (Bayesian, SGD, and Dropout). Our networks take 28x28-sized images as input and output categorical distribution over 10 classes.

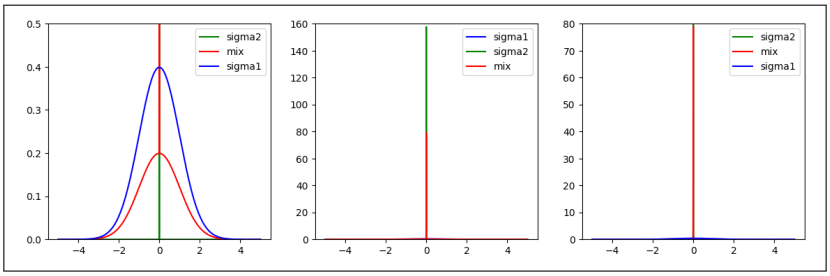

Figure 2: The figures show the original Scale Mixture prior used by [2]. We reproduce the code and show the effect of changing the scale of the y-axis to get a better understanding and visualization. We use the sigma of the first Gaussian as $\sigma = e^1$ and the sigma of the second Gaussian as $\sigma = e^{-6}$. In each figure, we plot first Gaussian, second Gaussian, and the mixture of both Gaussians.

To remain consistent and for a fair comparison, along with keeping the network architecture the same for all three approaches, we also train all the networks for exactly 50 epochs. One thing to note is that Bayes by Backprop network pays a huge computational cost when compared to SGD. On average, Bayes by Backprop takes about 50 minutes to complete 50 epochs of training on RTX 3080 GPU while SGD and Dropout networks only take about 5 minutes to complete the 50 epochs of training.

4.1 Bayes by Backprop training

We first initialize the network parameters randomly. During each forward propagation (train or test) the values of network parameters are sampled from the Scale Mixture Gaussian probability distributions. The parameters of these probability distributions are learned during the training time. In Fig. 3 we show the loss curve on both datasets.

4.2 SGD and Dropout training

We again first initialize the network parameters randomly. And then we use the cross entropy loss function to train the network without any Dropout rate (i.e. vanilla SGD) and with 3 dropout rates of 0.5, 0.75, and 0.9. Our motivation to go beyond the 0.5 dropout rate was to get the weight distribution variance higher. In Fig. 4, Fig. 5, Fig. 6, and Fig. 7 we show the loss curve on both datasets for SGD, Dropout rate 0.5, Dropout rate 0.75, and Dropout rate 0.9 respectively.

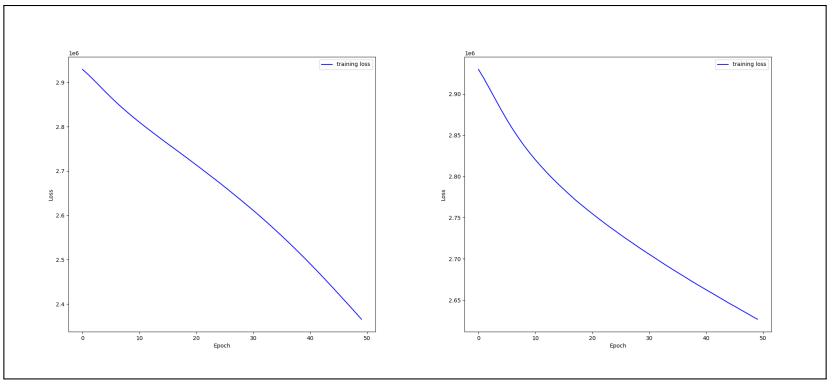

Figure 3: The figures show the training loss curves for Bayes by Backprop network on FMNIST and MNIST datasets. The left figure shows FMNIST and the right figure shows MNIST.

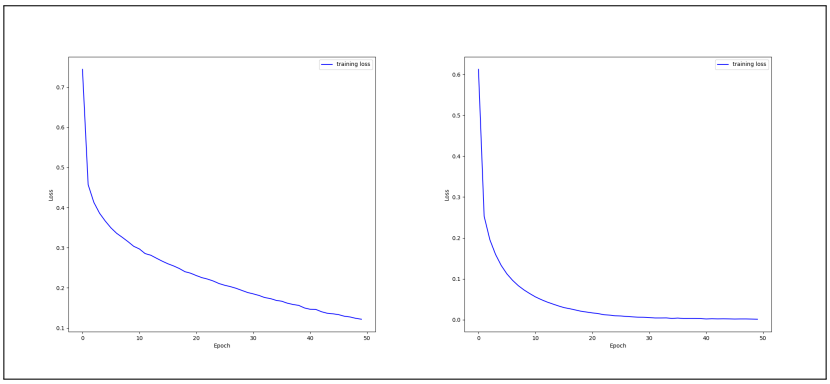

Figure 4: The figures show the training loss curves for the SGD network on FMNIST and MNIST datasets. The left figure shows FMNIST and the right figure shows MNIST.

4.3 Weight distribution

Just like [2] presented the weight distribution of their models in Fig 1. we too present the weight distribution for all of our trained models. In Fig 8. we present the weight parameter distributions of the 5 models trained on the MNIST dataset and in Fig 9. we present the weight parameter distributions of our 5 models trained on the FMNIST dataset. The results are not ideal because the authors reported that dropout networks attain higher weight distribution variance than SGD as opposed to our findings. We discuss our analysis in the next section.

4.4 Evaluation

Finally, we also present our evaluation of all trained models on both FMNIST and MNIST test sets. We prune all the networks for 3 thresholds such as 50%, 75%, and 90%. And report the results in Table 1. and Table 2. The details about our pruning techniques it described in previous sections.

Table 1: The table presents the test accuracy attained by all trained and pruned models on the MNIST test set

Method	0% pruned	50% pruned	75% pruned	90% pruned
SGD	0.9783	0.9753	0.9586	0.6943
Dropout (p=0.5)	0.9827	0.9763	0.9108	0.2216
Dropout (p=0.75)	0.9576	0.9290	0.6097	0.1287
Dropout (p=0.9)	0.8663	0.7493	0.2537	0.1960
Bayes by Backprop	0.9820	0.9819	0.9823	0.9822

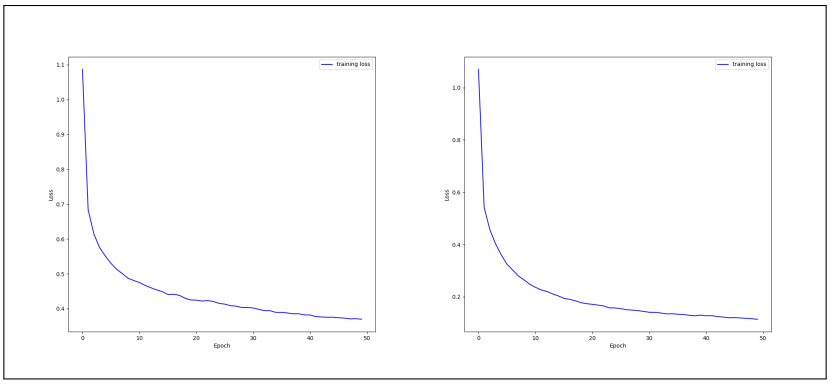

Figure 5: The figures show the training loss curves for the SGD network with a Dropout rate of 0.5 on FMNIST and MNIST datasets. The left figure shows FMNIST and the right figure shows MNIST.

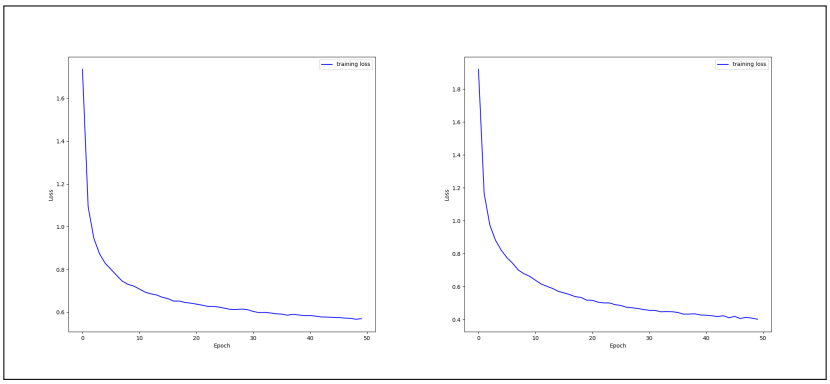

Figure 6: The figures show the training loss curves for the SGD network with a Dropout rate of 0.75 on FMNIST and MNIST datasets. The left figure shows FMNIST and the right figure shows MNIST.

Table 2: The table presents the test accuracy attained by all trained and pruned models on the FMNIST test set

Method	0% pruned	50% pruned	75% pruned	90% pruned
SGD	0.8967	0.8877	0.8003	0.4890
Dropout (p=0.5)	0.8756	0.8765	0.7595	0.3436
Dropout (p=0.75)	0.8277	0.8256	0.5790	0.2150
Dropout (p=0.9)	0.7198	0.6552	0.3479	0.1539
Bayes by Backprop	0.8925	0.8941	0.8934	0.8962

5 Discussion

As seen in the tables the dropout and SGD-based methods do not attain as good as Bayes by Backprop performance at any stage (before and after pruning). However, we can see that in terms of weight distribution variance SGD models show us a little more variance than dropout models. And that is why we can also see that after pruning, the SGD-based models perform a little better performance than the dropout models. This shows that the authors [2] argue that having a larger variance in model parameters distribution indeed helps to retain performance after pruning.

By looking at these results and training curves we can conclude that all models are underfitting if we keep training them for more epochs all of them will continue to improve. And because of this we also tried to train SGD and Dropout based networks for more than 50 epochs and trained the models to 150 epochs. We did see a minor improvement in the dropout models distribution variance and resulting accuracy gain. But it was not a major improvement and the models were still underfitting.

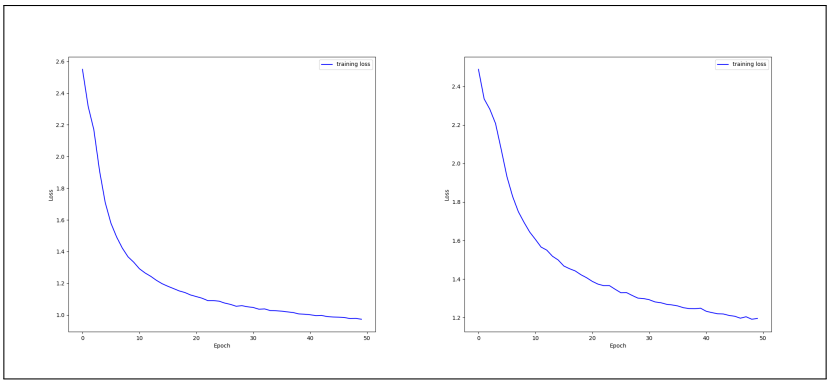

Figure 7: The figures show the training loss curves for the SGD network with a Dropout rate of 0.9 on FMNIST and MNIST datasets. The left figure shows FMNIST and the right figure shows MNIST.

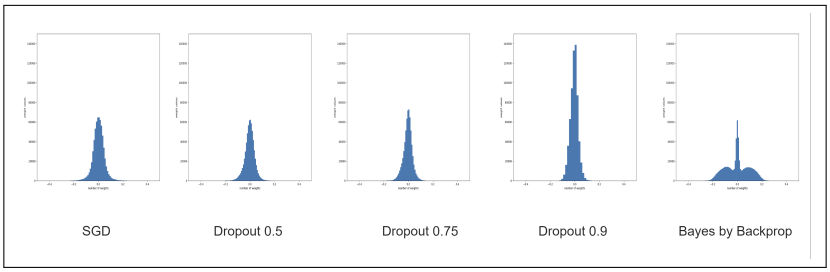

Figure 8: The figures show the weight distribution of all 5 models trained on the FMNIST dataset for 50 epochs

For this reason, we do not show the results for the 150 epochs experiments as it will be redundant. Thus, in future work, we plan to train the models for much more long. The authors [2] have trained a much larger network with 1200 hidden units and for 600 epochs and this also suggests that our models are underfitting.

But from our work, we can conclude one interesting property of the [2] Bayesian Neural Network is that it gets trained more quickly in terms of the number of epochs when compared with SGD and Dropout based models.

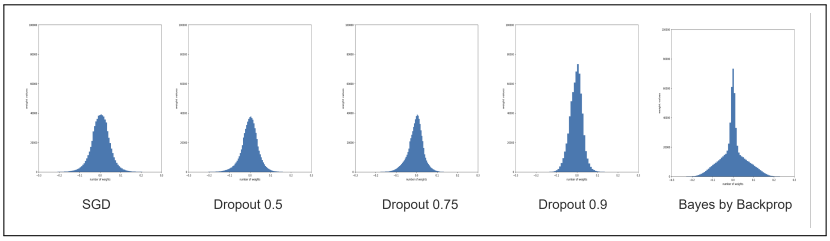

Figure 9: The figures show the weight distribution of all 5 models trained on the MNIST dataset for 50 epochs

6 Conclusion

In this paper, we carry out a systematic comparison of the Bayesian Neural Network introduced by [2] with SGD and Dropout-based models under the same environment which includes the same model architecture, dataset, optimization algorithms, etc. We also show that the Bayesian Neural Network [2] achieves great accuracy even after pruning large portions are parameters when compared to normal SGD-based approaches. And additionally, our results also show that it takes 10 times longer to train the Bayesian Neural Network in terms of wall time however it gets trained much more data-efficient way as it needs relatively fewer epochs to fit the overall data. Our future involves trying to replicate our experiments with more epochs to ensure SGD and Dropout based models do not under-fit.

References

[1] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning?, 2020. URL https://arxiv.org/abs/2003.03033.

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks, 2015. URL https://arxiv.org/abs/1505.05424.

[3] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks, 2017. URL https://arxiv.org/abs/1710.09282.

[4] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53(7):5113–5155, Oct 2020. ISSN 1573-7462. doi: 10.1007/s10462-020-09816-7. URL https://doi.org/10.1007/s10462-020-09816-7.

[5] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

[6] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks, 2015. URL https://arxiv.org/abs/1506.02626.

[7] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning, 2017. URL https://arxiv.org/abs/1705.08665.

[8] X. Ma, A. R. Triki, M. Berman, C. Sagonas, J. Cali, and M. Blaschko. A bayesian optimization framework for neural network compression. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10273–10282, 2019. doi: 10.1109/ICCV.2019.01037.

[9] R. Mishra, H. P. Gupta, and T. Dutta. A survey on deep neural network compression: Challenges, overview, and solutions, 2020. URL https://arxiv.org/abs/2010.03954.

[10] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. URL https://arxiv.org/abs/1708.07747.