Bayesian Neural Networks

Aaditya Singh

Abstract

In this project, I decided to investigate Bayesian Neural Networks (BNNs). Specifically, I attempted to reproduce the work of Blundell et al., who developed an algorithm known as "Bayes by Backprop" for efficient training of BNNs (Blundell et al., 2015). Using a combination of Monte Carlo estimates and automatic differentiation, their approach leads to tractable variational inference on the weights of a neural network and removes the need for hand-derived variational updates (similar to ADVI for general variational inference (Kucukelbir et al., 2016)). Although the results boasted in the paper are quite impressive, they skirt over information about and justification for many hyperparameter choices. In this report, I investigate these various choices and how important they are to the great results in the paper.

1. Introduction

In the field of deep learning, a common issue and inability for widespread use in critical situations (like medical diagnoses) comes from the lack of uncertainty measures. Typical networks just perform maximum likelihood inference, or maximum a posteriori if regularization is used, for the model parameters. Bayesian Neural Networks (BNNs) solve this problem by inferring the posterior over model parameters (the weights of the network). However, as we see in all of Bayesian inference, inferring the posterior is a difficult task. A recent approach that has gained more traction is using variational inference, which is motivated by a desire to still be able to use the backpropogation algorithm that has been so critical to the field . Blundell et al. develop an algorithm known as "Bayes by Backprop" [1] and apply it to a variety of baseline tasks to show advantages of a Bayesian framework. However, throughout the paper there is a lack of discussion about training time (which is vastly larger for BNNs) as well as a lack of justification and information about seemingly arbitrary hyperparameter choices. Although the results from the paper look extremely promising, BNNs are still not the go-to method, which prompts the question: How important are these hyperparameters? In this report, we examine the effect these various hyperparameters have on performance by starting with the most basic model. As suspected, the amazing results shown in the paper are much harder to achieve than one woudl think at first glance.

1.1. Bayesian Neural Networks

Before diving into the workings of the model, some background in BNNs and a brief overview of the approach in [1] is necessary.

Simply put, BNNs are a way of specifying a very complex likelihood function, parameterized by a set of weights, using a neural network. Working with BNNs, as a result, involves performing posterior inference on the weights (the parameters) given the data and a prior on the weights. After posteriors are known, inference on new inputs can be done by sampling from the posterior and getting outputs (similar to ensemble techniques). Since a BNN represents an infinite number of neural networks, we can sample as many times as we want to get better and better estimates (as well as uncertainties).

However, for even the simplest networks, there are usually thousands, if not millions of weights involved, which makes pure sampling-based methods for approximating the posterior intractably slow. A common alternative to sampling-based methods for the posterior inference problem is variational inference (VI). In VI, we reframe posterior inference as an optimization problem where we parametrize a tractable approximating distribution, and then optimize its parameters to make it match the true posterior as closely as it can by minimizing the KL divergence. Since the true posterior is not known, this cannot be done directly, so instead a lower bound to the evidence is used (Blei et al., 2016). The optimization is usually done using gradient ascent on this evidential lower bound (ELBO). The difficulties of VI are: choosing an approximating family (which greatly limits the accuracy of your posterior) and deriving the variational updates (to do gradient ascent). For now, we choose a multivariate gaussian with diagonal covariance for the approximating family, as this model only has twice the number of parameters as the original network. For large networks, this is an appealing choice as more complicated distributions may be difficult to sample from, or require many more parameters (the storage of which may be infeasible).

1.2. Bayes by Backprop

For BNN's, Blundell et al. provide an automatatic differentiation approach to to performing these updates, which takes advantage of the reparametrization trick. For more details on proofs, I refer the reader to the original paper, but I give the key ideas of the algorithm below. Also, to clear up some notation, $\mathbf{w}$ denotes all weights in the network, the variational apprximation is denoted by $q(\mathbf{w}|\theta)$ where $\theta = (\mu, \rho)$ and each weight is independently distributed according to $\mathbf{w}_i \sim \mathcal{N}(\mu_i, \log(1+\rho_i))$[1], so q is a multivariate Gaussian.

1. Sample a mini-batch $\mathcal{D}$. We denote M as the number of mini-batches per epoch.
2. Sample a $\mathbf{w} \sim q(\mathbf{w}|\theta)$. Gradients can be propogated through this step by using the reparametrization trick.
3. For each network, calculate $f(\mathbf{w}, \theta) = \frac{1}{M}(\log q(\mathbf{w}|\theta) - \log P(\mathbf{w})) - \log P(\mathcal{D}|\mathbf{w})$. We can decompose this loss as the log probability of these weights given our approximation minus the log probability of these weights given the prior[2], minus the log probability of the data given the sampled weights.
4. Take advantage of automatic differentiation on f to update the parameters θ.

Again, this is just a high-level overview as the algorithm itself is not our main concern. For a more complete description, I urge the reader to consult the original paper.

1.3. Seemingly Arbitrary Choices

Although the algorithm above seems simple, there are many choices one has to make in implementing. A few typical ones (when dealing with neural networks) are: What should the learning rate be? How should the data be normalized? In addition to these, we have some algorithm specific decisions to make: How many samples of the network should one take for every update? What type of prior should be used[3]? How should θ be initialized? And finally, the authors introduce another choice in the method when they discuss minibatches. Specifically, the authors note that the loss consists of two terms: the KL divergence of the posterior to the prior, and the likelihood of the data. The authors argue that the KL loss does not need to be uniformly split over the minibatches in an epoch, as it does not depend on the data. They offer a reweighting scheme that replaces the $\frac{1}{M}$ in the above algorithm with $\frac{2^{M-i}}{2^M-1}$ for mini-batch i. In this report, I investigate many of these choices, and how necessary they are for the results in the paper.

2. Methods

All results shown in this report were generating using my own PyTorch code that implements the Bayes by Backprop algorithm described in the paper[4], and was run on an NVIDIA P100 GPU using Google Cloud. To evaluate the effects of the various choices, I focused on the fully-connected neural network models trained on MNIST. Specifically, I investigated the paper's results on test error and weight distribution (Table 1, Figures 2 and 3). After discussing the specific hyperparameters, I also evaluated the efficacy of the paper's weight pruning using signal-to-noise ratio (which they argue also make Bayesian Neural Networks worth the extra training cost). Although I was unable to exactly reproduce the paper's findings in most cases (due to computational limitations), there are many noteworthy findings.

3. Results

3.1. Compute Time

Before delving into the figures and results, I'd like to make an important point about the increased compute needed to train BNNs, even with Bayes by Backprop. First of all, the number of parameters doubles (we need a mean and standard deviation for every variable). Secondly, based on how many samples we take per iteration, the training time per epoch is also multiplied by a constant factor. On a two-layer fully-connected standard neural network with 400 hidden units, the training time per epoch on a P100 is signficantly smaller than it takes for a BNN. Furthermore, to get intermediate test error estimates, the BNN is even more expensive, as one likely wants to take more samples to make sure the test error is accurate. Even when running on a P100 GPU, this smallest network only gets through about 75 epochs an hour (which is pretty slow for just training on MNIST!). As a result, all the results shown in this paper use a learning rate of 10^{-3} and train for 600 epochs[5] (smaller learning rates were tried but didn't converge fast enough). One more experiment was conducted using the latent dimension of 1200 and as many hyperparameter settings the same as given in the paper. This larger latent dimension cut the speed down to about 35 epochs an hour. This run also had to be stopped at 600 epochs, due to depletion of my google cloud credits. Already, one can see that the casually stated hyperparameter tuning in the paper, which considers 324 different settings,

[1] This reparametrization of the standard deviation is quite common for numerical stability, so I do not question the paper's choice in doing this.

[2] This is actually the Monte Carlo estimate of the KL divergence of our approximating distribution to the prior

[3] Bayes by Backprop lifts the restraint of using a Gaussian prior. For more info, read the original paper.

[4] The code can be found at https://github.com/aadityasingh/Bayes-Nets

[5] this number is chosen to match Figure 2 from the paper

would take a very long time to run on a single GPU (and would be extremely expensive)!

3.2. KL reweighting

To determine whether or not KL-reweighting had a big impact on the results, a BNN with latent dimension 400, learning rate 10^{-3}, and SM prior with $\pi = 0.5, \sigma_1 = 1, \sigma_2 = e^{-6}$ was used with or without KL re-weighting. Throughout training, the test error for the KL-reweighted run was consistently 0.3% below the non-KL reweighted run. The weight distributions of the two runs did not differ significantly, except that the KL-reweighted run had a slightly more symmetric distribution of weights.

3.3. Different Priors

The paper mentions two different priors, and I decided to experiment with the same two. The first is just a standard Gaussian centered around 0. The second one, referred to as a Scale Mixture (SM) Prior , is a mixture of a broad Gaussian and a pointed Gaussian, both centered at 0. For the standard Gaussian prior, there is one hyperparameter σ_1, and for the SM prior, there are three hyperparameters: the mixture ratio π and the two standard deviations σ_1 and σ_2.[6] Although the paper considers many different combinations of hyperparameters, as noted above, I did not have the resources to perform this huge hyperparameter tuning. However, I did experiment with the different hyperparameters (as shown in Table 1).

First and foremost, I found that the SM prior always yields a spiked weight distribution (see Figure 1b), whereas the Gaussian prior yields the wide spread weight distribution (see Figure 1c) the paper boasts of. Furthermore, I found that the Gaussian distribution actually outperformed all three SM Prior runs significantly (which is in direct contrast to the paper). However, this may be due to the fact that the SM prior has more fittable hyperparameters (3 compared to 1). After fitting these hyperparameters, it is very likely that the SM Prior performs better. However, going back to the earlier note on compute time, I would assert (contrary to the paper), that a Gaussian prior is still the way to go for quick and decently accurate results. Also, surprisingly, my trained Gaussian prior network ended with an accuracy of 1.69%, which is less than the paper's 1.82%. This may be due to a lack of hyperparameter fitting for the Gaussian prior in the paper (since fitting is only discussed for the SM prior), and my choice of parameters may have just been a good choice. Also, this could possibly be due to lucky samples (I sample 10 networks to evaluate test error), as I've seen variation of up to 0.1% in the test error (just by running again).

[6] π is the weight given to the first, broader component

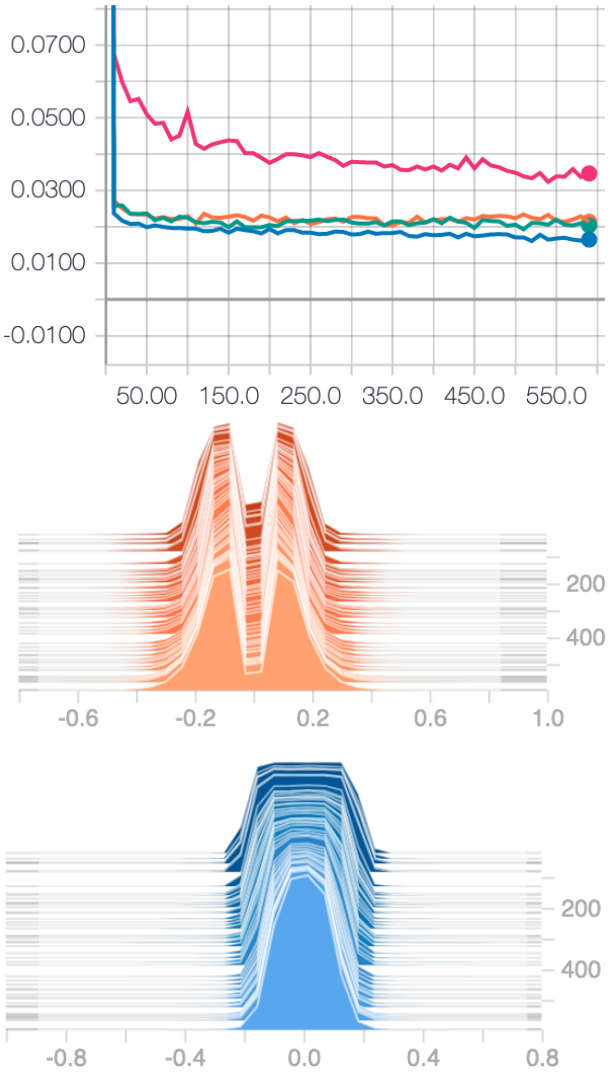

Figure 1. (a)-(c) top to bottom. Gaussion prior runs shown in blue. SM prior runs shown in orange, pink, green with prior parameters $(\pi, \sigma_1, \sigma_2)$ as (0.5, 1, e^{-6}), (0.5, 1, e^{-6}), (0.25, 1, e^{-6}), respectively. All modesl trained for (a) Test error as training progresses. Note that the Gaussian prior performs best. (b) Weight distribution over epochs during training for first SM prior network. This bimodal shape is seen in all SM prior runs. (c) Weight distribution over epochs during training for Gaussian prior BNN. Note the nice shape and length scale matching that of the results presented in the Blundell et al.

3.4. Normalization

Another weird choice in the paper is the normalization of MNIST data. The two canonical normalizations are either just rescaling to [0, 1] (dividing by 255), or normalizing according to the mean pixel value and the standard deviation of. However, the paper normalizes by dividing each pixel by 126, which seems extremely arbitrary. I experimented

training networks using the SM prior (like the paper), with the settings of $\pi = 0.5, \sigma_1 = 1, \sigma_2 = e^{-6}$, with the three different normalization schemes. The weird dividing by 126 normalization used by the paper yielded an accuracy $0.1 - 0.2\%$ better than the other two schemes on average[7].

4. Weight pruning

An interesting application of posteriors given in the paper is weight pruning. Very basic neural network weight pruning schemes just set a threshold on the absolute value of weights, and all weights under that value get set to 0 (I refer to this as naive pruning). Blundell et al. propose to use the signal-to-noise ratio (defined as $|\mu_i|/\sigma_i$ for weight w_i) to threshold the weights. In their experiments with latent dimension 1200 and SM prior, pruning up to $75 - 95\%$ of the weights seems to has barely any effect on performance, which they assert is due to this bimodal distribution of signal-to-noise.

For most of my runs, I did not observe a bimodal signal-to-noise distribution. I believe this is due to a lack of hyperparameter tuning as for one of my runs (using the weird divide-by-126 normalization, and SM prior with parameters $\pi = 0.5, \sigma_1 = 1, \sigma_2 = e^{-6}$), there is a bimodal distribution (see Figure 2). However, naive pruning[8] on this case seems to perform better than the fancy signal-to-noise pruning (see Table 1). A closer look at the weight distributions (see Figure 3) shows the difference between the two methods. The signal-to-noise pruning sometimes keeps weights whose means have smaller absolute values, but have less uncertainty, whereas the naive pruning abruptly chops the weights at the threshold. A possible explanation for signal-to-noise performing worse is the common underestimation of uncertainties in mean-field variational inference (where all the weights are independent). If the uncertainties are underestimated, the signal-to-noise will be overestimated leading to weights that should be pruned remaining in the network, causing higher error. However, the paper still shows promising signal-to-noise pruning results but they did not compare their weight pruning to another technique, so it may be a viable option, but not necessary.

5. Conclusion

Overall, the results presented by Blundell et al. are still very impressive, but nearly intractable without large amounts of compute power. Although Bayes by Backprop is much faster than any sampling-based technique and can be extended to any prior distribution, the amount of time needed to train a single network is still prohibitive if hyperparameter tuning is necessary (which, in most cases, it appears to be). With regards to weight pruning, there's no evidence that the signal-to-noise scheme performs better than a naive weight thresholding scheme, indicating that posterior estimates do not necessarily lead to better weight pruning.

[7] I reran the test script to make sure this wasn't just noise in the 10 samples being drawn

[8] Naive pruning is performed on BNNs by just filtering based on the posterior means. This intuitively makes sense and can be though of as training a separate standard neural network with some regularization scheme and pruning that

Table 1. All runs were performed on networks with a latent dimension of 400 and SM prior parameters $\pi = 0.5, \sigma_1 = 1, \sigma_2 = e^{-6}$. The best checkpoint from training based on validation error was used (instead of the last checkpoint at the end of training). Left column indicates percentage of weights pruned, next two columns indicate test errors from different pruning techniques.

% PRUNED	NAIVE	SIGNAL-TO-NOISE
0%	2.03%	2.03%
25%	2.41%	2.59%
50%	5.33%	5.89%
75%	30.67%	41.05%
95%	63.110%	82.50%

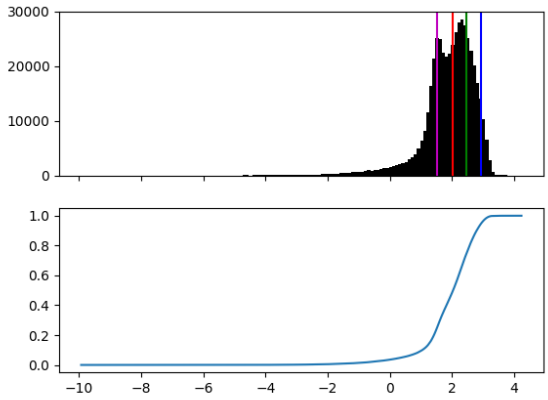

Figure 2. (a)-(b) top to bottom. (a) Histogram of noises with 25%, 50%, 75%, and 95% cutoffs indicated in magenta, red, green, and blue, respectively (b) Cumulative distribution for the histogram. The relative straightness of the incline indiates there's not much separation between the two modes

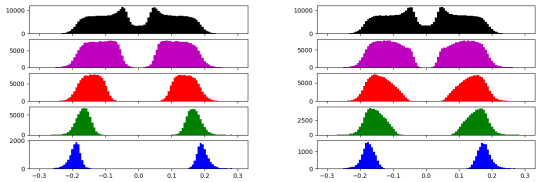

Figure 3. (a)-(b) left to right. Color scheme is the same as Figure 2, with black representing no weights pruned. (a) Naively pruned weight histograms (b) Signal-to-noise pruned weight histograms. The sharper drop-offs in plot (a) are indicative of thresholding on the posterior, whereas the declined dropoffs in (b) indicate weights with smaller $|\mu|$, but small uncertainties as well

References

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational Inference: A Review for Statisticians. *arXiv e-prints*, art. arXiv:1601.00670, Jan 2016.

Blundell, C., Cornebise, J., Kavukcuoglo, K., and Wierstra, D. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, Lille, France, 2015.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. Automatic Differentiation Variational Inference. *arXiv e-prints*, art. arXiv:1603.00788, Mar 2016.