An Empirical Evaluation of Bayesian Inference Methods for Bayesian Neural Networks

Rui Zhao
RPI
zhaor@rpi.edu

Qiang Ji
RPI
qji@ecse.rpi.edu

Abstract

In this paper we perform an empirical evaluation of different inference methods for Bayesian neural networks (BNN). We compare two major approaches, including Hamiltonian Monte Carlo (HMC) and Variational Inference (VI). Both methods are implemented using Tensorflow Probability Toolbox. We first evaluate how different factors involved in HMC and VI affect the inference results, focusing on classification problem. We then show benefits of Bayesian inference in generalization over point estimation approach. Finally, we provide a way of quantifying uncertainty based on Bayesian inference.

1 Introduction

Over the last few years, there is a resurgence of interest in Bayesian neural networks (BNN) [20, 18] as it provides a promising direction to address the limitation of neural networks in generalization and overlook of uncertainty [6]. BNN is a probabilistic extension of neural networks. The key of BNN is treating parameters in original neural networks as random variables, whose distribution is specified by the prior distribution. The benefits of such extension is as follows. First, BNN reduces the risk of overfitting through model averaging. Second, BNN can quantify the uncertainty of model prediction. We consider two major approaches to perform Bayesian inference for BNN. The first one is Hamiltonian Monte Carlo (HMC). The second one is variational inference (VI). After a brief discussion of the Bayesian inference problem and the two methods, we empirically evaluate different factors that need to be considered in using HMC and VI during inference. We compare the results with point estimation approach. Finally, we provide an analysis of estimated uncertainty.

2 Bayesian Inference for BNN

We first provide a general description of BNN. Suppose we have the observation X and the associated target value Y, BNN defines a probabilistic conditional distribution $P(Y|X, \theta)$ using neural networks with parameter θ. Furthermore, θ is also treated as a random variable with a prior distribution $P(\theta|\alpha)$ and α is the hyperparameter. Given a set of training data $\mathcal{D} = \{X_i, Y_i\}_{i=1}^N$ and a query of testing data X', the goal of Bayesian inference is to compute the conditional posterior distribution of target variable Y' as follows.

$$P(Y'|X', \mathcal{D}, \alpha) = \int_\theta P(Y'|X', \theta)P(\theta|\mathcal{D}, \alpha)d\theta \tag{1}$$

A full Bayesian treatment can also be used where α is also modeled as a random variable and to be integrated out together with θ. To estimate the value of target variable, we solve the following optimization problem.

$$Y^* = \arg\max_y P(Y' = y|X', \mathcal{D}, \alpha) \approx \arg\max_y \frac{1}{M}\sum_{i=1}^{M} P(Y' = y|X', \theta_i), \theta_i \sim P(\theta|\mathcal{D}, \alpha) \tag{2}$$

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

The key to solve Eq. (2) is to generate samples from the posterior distribution $P(\theta|\mathcal{D}, \alpha)$. For BNN, the posterior is intractable due to the complex nonlinearity introduced by neural networks. Thus approximate inference methods are needed. For *classification*, the target variable $Y \in \{1, ..., K\}$ is a discrete-valued variable, representing the class label. K is the total number of classes. We assume Y follows categorical distribution.

$$P(Y|X, \theta) \triangleq [p_1, ..., p_K]^T = f_{NN}(X; \theta) \tag{3}$$

where $p_i \geq 0, \sum_{i=1}^{K} p_i = 1$ and $f_{NN}(X; \theta)$ is a neural network with a softmax output layer. Solving Eq. (2) is straightforward as we only need to evaluate the probabilities for each one of K classes and choose the one with the largest probability.

2.1 Computing Uncertainty

One of the major benefit of using BNN is to quantify the uncertainty associated with prediction. We quantify the uncertainty of inference results using marginal covariance of target variable, which can be shown to decompose into two terms.

$$V[Y|X] = E_\theta[V[Y|X, \theta]] + V_\theta[E[Y|X, \theta]] \tag{4}$$

The first term captures the intrinsic uncertainty in data (aleatoric uncertainty [13]). The second term captures the uncertainty among different model parameters (epistemic uncertainty [13]). In practice, $E[Y|X, \theta]$ and $V[Y|X, \theta]$ are readily available from the prediction of model given one particular instance of model parameter. Then $V[Y|X]$ can be computed using results of different model parameters. Finally, we compute the total uncertainty using the trace of total covariance *i.e.*, $U(X) = \sum_i V[Y|X]_{ii}$.

3 Inference Methods

Hamiltonian Monte Carlo (HMC) [21] is a variant of MCMC method, which uses gradient of Hamiltonian function to update the Markov chain. HMC alleviates random walk behavior and allows the chain to explore the target distribution more efficiently. HMC and its extensions are widely used for inference of BNN [21, 24, 3, 8]. In the context of sampling from posterior distribution, Hamiltonians $H(\theta, r)$ is a function defined over the BNN parameter θ and auxiliary variable r.

$$H(\theta, r) = U(\theta) + K(r),\ U(\theta) \triangleq \sum_{\{X,Y\} \in \mathcal{D}} \log P(Y|X, \theta) + \log P(\theta|\alpha) \tag{5}$$

where $K(r)$ is a quadratic function of r [21]. Following Hamilton's equations, we can map a state *i.e.*, value of θ, r at time t to a state at time $t + s$. Starting from some initial state, we can compute value of θ at different time t. Neal [21] proved that the stationary distribution simulated by Hamiltonian defined by Eq. (5) is the posterior distribution $P(\theta|\mathcal{D}, \alpha)$.

Variational Inference (VI) [11] is an approximate inference method that converts the inference problem into an optimization problem. The key idea is to introduce a variational distribution $q(\theta|\psi)$, where ψ is called variational parameters. q is usually chosen to be tractable and easy to compute. For example, mean-field variational method assumes fully factorized q over θ. The value of ψ is estimated by solving the following optimization problem.

$$\begin{aligned} \psi^* &= \arg\min_\psi KL(q(\theta|\psi)||P(\theta|\mathcal{D}, \alpha)) \\ &= \arg\max_\psi E_{q(\theta|\psi)}[\log P(\mathcal{D}, \theta|\alpha)] - E_{q(\theta|\psi)}[\log q(\theta|\psi)] \end{aligned} \tag{6}$$

where $KL(\cdot||\cdot)$ is the Kullback-Leibler divergence. After we find the optimal ψ^*, the samples of θ are obtained through sampling $\theta \sim q(\theta|\psi)$. In case of Gaussian mean-field method, $q(\theta|\psi)$ is fully factorized over individual parameter. The sampling of all parameters reduces to sampling each parameter from the respective Gaussian distribution. Variants of variational inference have been proposed for BNN inference [9, 19, 22, 2].

4 Experiments

In this section, we compare different inference methods on classification task. We use MNIST handwritten digits dataset [16] for all the experiments. We use 5500 images from training set and all the 10000 images from testing set. To test generalization, we add Gaussian noise with standard deviation 0.3 to the testing images whose values are scaled linearly between 0 and 1. We implemented a BNN with two convolutional layers and two fully connected layers using Tensorflow [1] and Tensorflow probability toolbox [4]. The details are in Appendix A. We summarize all the factors considered in Table 1. Based on our results, we also provide a guideline to configure all the factors. The detailed results are provided in Appendix B.

Table 1: Performance factors evaluated in different inference methods (See Appendix B for details).

Method	Factors	Guideline
HMC	Prior of model parameters	Standard Normal with small variance
	Initialization of MC state	Sample from prior
	Burn-in iterations	Determine by within-sliding window performance
	Number of samples collected	Proportional to parameter size
	Step number L and size ϵ	Critical to performance. Tune by running small length chain and observing acceptance rate and performance
Variational	Prior of model parameters	Standard Normal with small variance
	Initialization of variational parameters	Sample from prior
	Optimization termination criterion	Fixed threshold to the loss
	Number of samples collected	Moderate (parameter size independent)

4.1 Comparison among Different Inference Methods

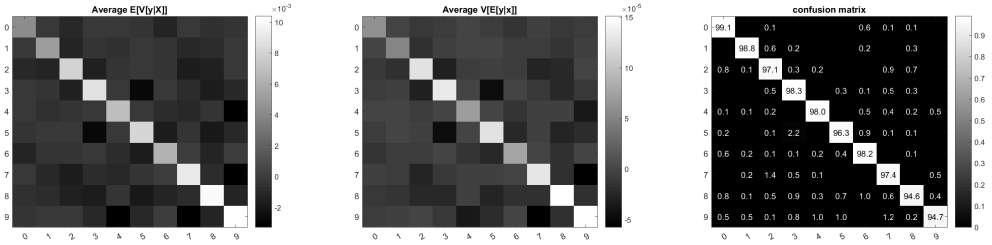
With all the factors in consideration, we perform a comparison in classification performance between HMC, VI, MC dropout [7], and two point estimate methods. We found that the performance of HMC is subject to the choice of leapfrog step size ϵ and number L. We tune these parameters by running the chain for a small number of iterations and observe the acceptance rate and performance. The final choice are $\epsilon = 0.001, L = 10$. In terms of speed, ML and MAP are the fastest. VI and MC Dropout are slightly slower due to the extra cost for performing inference using multiple samples. HMC is much slower than other four methods mainly due to the need to compute gradient over the entire training dataset at each leapfrog update. The time increases linearly with the number of leapfrog steps. One way to improve this is to use stochastic scheme such as SGLD [24] and SGHMC [3]. All the methods are repeated for 10 times with different random initializations. Overall, VI provides a more competitive performance than HMC. It also allows better scaling with dataset and model size. However, VI increases the amount of parameters by a factor of two due to the use of variational parameters. For HMC, stochastic scheme must be used in order to scale up to large dataset despite that it is easier to implement and does not require additional parameterization cost. SGHMC improves HMC in both speed and final accuracy. MC dropout is comparable with VI as it can be considered as an approximation to VI. We use a dropout rate 0.1 as a larger rate yields worse performance in our experiment. Compared to point estimation approaches, all Bayesian inference methods show better accuracy and generalization. In particular, in the noisy test, point estimation results drop significantly, while Bayesian inference results remain comparable with the clean test. This shows the superior generalization capability of BNN with Bayesian inference. Furthermore, Bayesian inference have additional benefits in providing uncertainty measure as discussed in the next section.

Table 2: Compare the classification performance of different methods under different testing data. Each method is repeated for 10 times with different random initializations.

Method	ML	MAP	VI	MC dropout	HMC	SGHMC
Clean test (Acc.±Std.)	97.26±0.22	97.51±0.19	97.76±0.26	97.71±0.16	97.30±0.20	97.67±0.08
Noisy test (Acc.±Std.)	79.65±0.28	82.70±0.15	96.60±0.21	93.93±0.25	96.15±0.11	96.56±0.10
Time per trial (h)	0.5	0.5	0.6	0.7	4	1.5

4.2 Analysis of Uncertainty

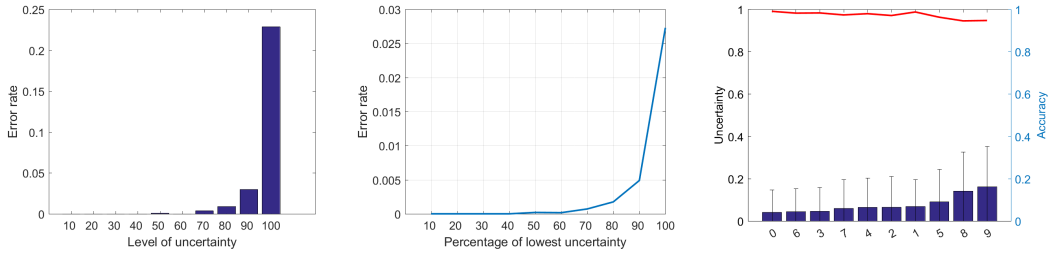
As shown in Section 2.1, the uncertainty consists of two terms, namely the data uncertainty and the model uncertainty (both are represented by the covariance). We first analyze each individual uncertainty, using the results obtained by variational method. In classification task, both terms in Eq. (4) are $K \times K$ matrices where $K = 10$ is the total number of classes. From Figure 1(a)-1(b), we see that the data uncertainty is much larger than model uncertainty, indicating a small difference among different model predictions. The major source of uncertainty comes from the data. We can also compare the covariance with confusion matrix in Figure 1(c). Unlike confusion matrix, computing covariance does not need the ground truth labels. Yet it provides indication of what classes are easy to be confused with others. For example, digit '9' has high covariance with digit '7' and '4' in Figure 1(a). This is consistent with the confusion matrix, where digit '9' has high confusion with '7' and '4'.



(a) Data uncertainty (b) Model uncertainty (c) Confusion matrix

Figure 1: (a) Average testing data uncertainty. (b) Average testing model uncertainty. (c) Confusion matrix.

We then analyze the uncertainty and model performance. Figure 2(a)-(b) show two different ways of analysis. In Figure 2(a), we quantize the uncertainty into 10 different levels. Then we plot the error rate within each different levels. In Figure 2(b), we plot the error rate among data with at most certain level of uncertainty. It is clear that as the uncertainty increases, the error rate increases as well. We can use this uncertainty value to help us with the decision of classification. For example, we can consider top-2 result if the uncertainty value is high. Finally, we plot the uncertainty within each individual class in Figure 2(c). We sort the uncertainty level from low to high and plot the corresponding accuracy of the class. This gives us an indication on which class is more likely to be mis-classified. For example, '9' has the highest uncertainty. This is consistent with intuition that hand-written '9' can look similar to '1', '4' and '7'. Again, the accuracy decreases as the uncertainty value increases.



(a) (b) (c)

Figure 2: (a) Error rate within each uncertainty level. (b) Error rate with uncertainty up to different levels. (c) Class-wise uncertainty value. Left vertical axis is for the bar chart. Right vertical axis is for the curve plot.

5 Conclusion

We evaluated two commonly used approximate inference methods for BNN inference, based on which a guideline to adjust different factors of inference methods is developed. We believe this is

a useful guideline when applying different approximate inference methods for BNN. We further demonstrate the benefit of performing Bayesian inference by showing better performance in accuracy and generalization in classification task. In addition, the proposed estimation of uncertainty provides useful information for decision-making.

References

[1] M. Abadi et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv*, 2015.

[3] T. Chen, E. Fox, and C. Guestrin. Stochastic gradient hamiltonian monte carlo. In *ICML*, 2014.

[4] J. V. Dillon et al. Tensorflow distributions. *arXiv*, 2017.

[5] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[6] Y. Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

[7] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation. *arXiv preprint arXiv:1506.02157*, 2015.

[8] Z. Gan, C. Li, C. Chen, Y. Pu, Q. Su, and L. Carin. Scalable bayesian learning of recurrent neural networks for language modeling. In *ACL*, 2017.

[9] A. Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

[10] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[11] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[12] A. Joshi, S. Ghosh, M. Betke, S. Sclaroff, and H. Pfister. Personalizing gesture recognition using hierarchical bayesian neural networks. In *CVPR*, 2017.

[13] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *NIPS*, 2017.

[14] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv preprint arXiv:1705.07115*, 2017.

[15] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[17] C. Louizos and M. Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

[18] D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 1992.

[19] A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. *arXiv*, 2014.

[20] R. M. Neal. *Bayesian learning for neural networks*. 2012.

[21] R. M. Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

[22] R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *AISTATS*, 2014.

[23] K.-C. Wang, P. Vicol, J. Lucas, L. Gu, R. Grosse, and R. Zemel. Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317*, 2018.

[24] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

[25] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.

Appendices

A Implementation

We implement the BNN model using Tensorflow [1] and Tensorflow probability toolbox [4]. Specifically, we use a model with two convolution layers with ReLU activation function, followed by a max-pooling layer. The output of the last convolution layer is flattened to a vector and input to two fully connected layers. The last fully connected layer is the actual output. The total number of parameters may vary depending on the number of filters in convolution layer and nodes in fully connected layers. We use a model with about 20K parameters for most of the experiments except when we evaluate the effect of model size. The model architecture is shown in Figure 3.

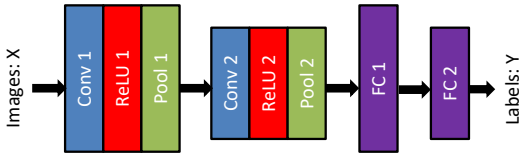


Figure 3: Architecture of CNN model. For classification task, a softmax transformation is further applied to FC2 to produce the output. For regression task, FC2 is the final output.

B Experiment Details

B.1 Experiment with HMC

The use of HMC requires careful tuning of the leapfrog step size and step number. We postpone the tuning for comparison among different inference methods and for now we fix the HMC parameter which provides reasonable results and evaluate other factors. First, we evaluate the burn-in iterations. Determine the mixing condition of MCMC remains an open research question. Here we use an empirical criterion based on final performance of the model to make decision. The intuition is that once the burn-in period is completed, The chain should be mixed in a stationary distribution. Therefore, we expect a relative stable performance after the burn-in period. In order to measure the stableness of performance, we used a sliding window based scheme, which is illustrated in Figure 4.

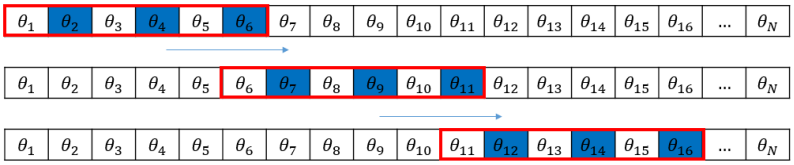
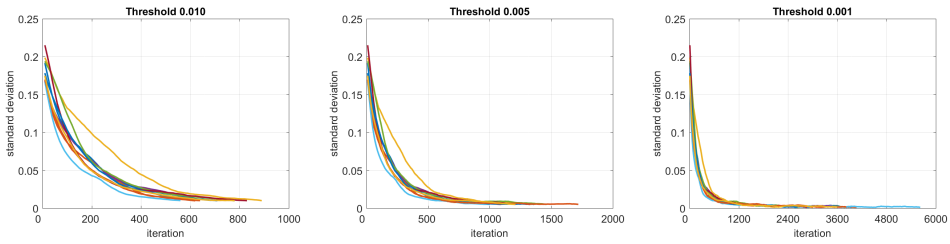


Figure 4: Sliding window along an MCMC of length N. The red bounding box indicates a sliding window. The samples in blue are the ones selected to perform Bayesian inference. The index of starting sample of the sliding window is called *position*. The total number of samples contained in a sliding window is called *length*. The difference between consecutive sliding windows is called *gap*. The difference between consecutive selected samples are called *skip*.

By varying the *length* of sliding window, we can get different inference results at the same *position*. We fix the value of *gap* and *skip* to be 10. If the chain is mixed, we expect the variation of results among different sliding window lengths to be small. We determine the burn-in iteration if the variation is smaller than some pre-defined threshold.

Figure 5 shows the change of variation versus the number of MCMC iterations with 10 different initializations of MCMC chain. The accuracy and burn-in iterations under different thresholds are listed in Table 3. The results indicate that the performance becomes stable when we reach the threshold $1e-4$. Later, we use this threshold for burn-in.



(a) threshold = 1×10^{-2} (b) threshold = 5×10^{-3} (c) threshold = 1×10^{-3}

Figure 5: HMC inference burn-in iterations under different termination threshold. Different color curves indicate different random initializations. (Best view in color)

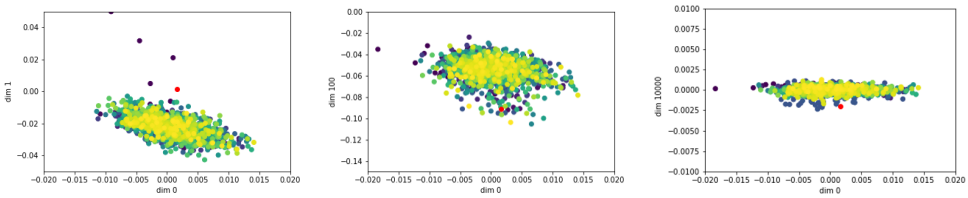
Table 3: Results of 10 runs of HMC in the format of average value $\pm$ standard deviation.

Threshold	1e-2	5e-3	1e-3	5e-4	1e-4	5e-5
Accuracy	89.24±0.81	90.98±1.05	93.36±0.77	94.01±0.41	95.01±0.30	95.01±0.30
Burn-in #	730±102	1305±201	3706±833	5329±1850	15498±4679	15498±4680

We also visualize the samples at different iterations of HMC in Figure 6. We visualize the scatter plot of two selected dimensions of the entire set of parameters. We color coded the the sample so that the brighter the color, the later the iterations of the sample. We use red to indicate the MAP estimate of the parameters. From the plot we observe a distribution of samples containing the MAP estimate. The inference is performed using multiple samples of parameters instead of a point estimate.

Second, we evaluate the number of samples used to perform inference. We continue to run the chain after the burn-in iterations. Then we compute the performance of the model with different numbers of samples collected after burn-in. Again, the experiment is repeated for 10 times with different random initializations and the results are listed in Table 4. We observe that 1) in general, the larger number of samples, the better the performance. While the performance increases 0.21% from 500 to 1000, the increase becomes 0.02% as we go from 5000 to 10000. 2) the standard deviation remains steady around 0.25%, indicating different initializations will roughly affect the final results uniformly under different numbers of samples. This also indicates that the performance change within 0.25% is not likely to be statistically significant. For the remaining experiment, we use 10000 examples to perform inference.

Third, we vary the value of prior distribution on parameters with fixed burn-in iterations and number of samples to perform inference. Typically, NN parameters are small numbers that are perturbed around 0. So for the prior distribution, we consider a Gaussian prior distribution with mean 0 and standard deviation σ. We only vary the value of σ. The results are summarized in Table 5. We see that a prior with smaller standard deviation is preferred. We can also observe that in Figure 6, the variance of sample values are indeed very small.



(a) (b) (c)

Figure 6: Sample value of selected dimensions along the chain simulated by HMC. Darker color dots are obtained at earlier iterations. Red dot is the MAP estimate (Best view in color).

Table 4: Results of different number of samples used for inference after burn-in.

Sample #	100	500	1000	2000	5000	10000
Acc.±Std.	94.75±0.29	95.02±0.22	95.23±0.23	95.34±0.25	95.43±0.24	95.45±0.27

Table 5: HMC performance under different prior of parameters with 10 random initializations.

σ	0.01	0.1	1	10
Acc. ± Std.	95.45±0.27	95.50±0.33	95.17± 0.27	94.57±0.36

Finally, we vary the size of the model and the results are listed in Table 6. Generally speaking, a larger model has a better performance. But the improvement becomes marginal once the model is large enough, considering the fact that we only used 5500 training images.

Table 6: HMC performance of different model size with 10 random initializations.

Parameters #	524	1922	3578	7370	13818	20494
Acc. ± Std.	84.83±2.33	94.39±0.46	94.68±0.35	95.25±0.33	95.17±0.17	95.50±0.32

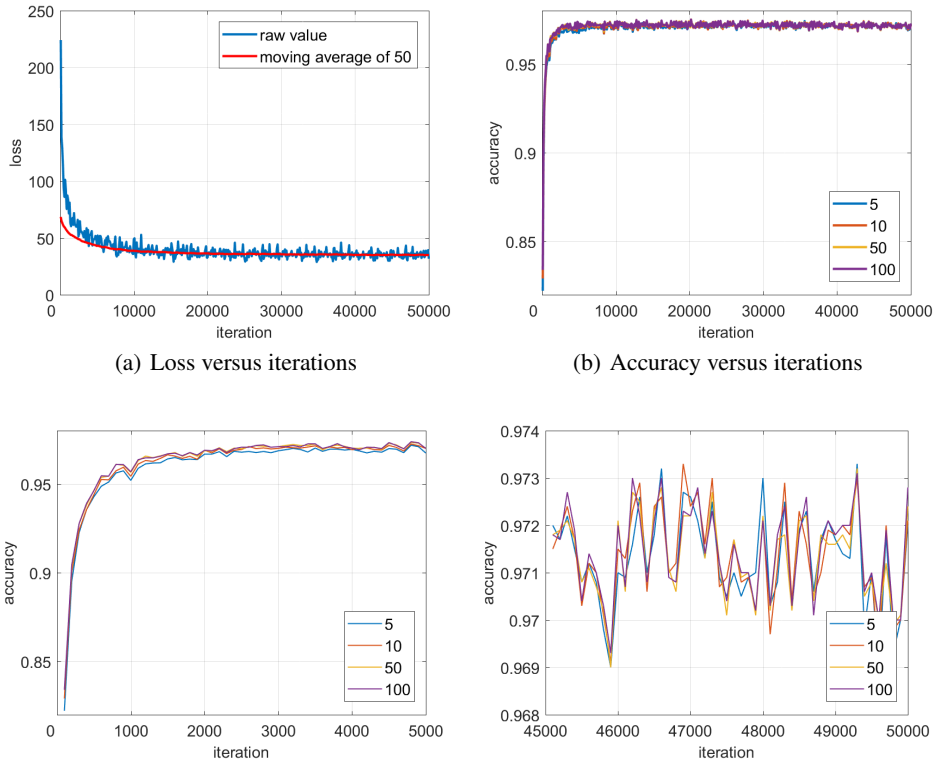
B.2 Experiment with Variational Method

We use one particular variant of variational inference method called Flipout [25], which has the advantage of performing inference efficiently on large-scale neural network models. The algorithm is implemented in Tensorflow probability toolbox [4]. In the first experiment, we evaluate how does the performance change with the number of samples and the stopping criterion of variational optimization. We use the same model architecture as used in HMC experiment. Figure 7(a) shows the -ELBO value versus the optimization iterations. We can see from the red moving average curve that the optimization converges between 20000-30000 iterations. Figure 7(b-d) shows the classification accuracy on testing data under different numbers of samples that are used to perform inference. The Bayesian inference follows the description in Section 2. We observe that the performance becomes stationary around 5000 iterations. Towards the end of the optimization, the inference results obtained by different numbers of samples are similar.

Then we evaluate how initialization on the variational parameters affect the performance by repeating the experiment for 10 times with random initialization on variational parameters. Table 7 summarizes the average accuracy and standard deviation at different numbers of optimization iterations and different numbers of samples used to perform Bayesian inference. We see the results become stable after 10000 iterations. In addition, the performance becomes stable with 50 samples as the difference between 50 and 100 samples is very small. Due to the use of mean-field assumption, 100 independent samples for each parameter seems sufficient.

For the second experiment, we evaluate the effect of prior distribution on parameters. We fix the number of iterations to be 50000 and the number of samples to be 100. Same as HMC experiments, we only vary the variance of prior distribution. The results are summarized in Table 8. Again, each experiment is repeated 10 times to account for random initializations. We observe that a smaller prior is preferred for better performance, which is similar to HMC.

We further visualize the distribution of parameters at the end of the optimization. As we assume Gaussian mean-field q function. Therefore, the posterior distribution of θ is approximated by a Gaussian with variational parameters $\psi = \{\mu, \sigma\}$. We plot the histogram of the values of μ and σ for parameters in each layer under different prior σ_0 as shown in Figure 8. These figures provide a rough estimate of how the model parameters will distribute. We observe that 1) the majority of the weights has an expected value that is close to 0. 2) the majority of the weights has a small standard deviation. 3) the choice of prior has more noticeable effect on some of the layers than the others. In particular, the last convolution layer and the first fully connected layer. The standard deviation of convolution layer tends to become larger if the prior σ_0 is larger. The standard deviation of fully connected layer has a small portion of mass at the initial prior value. Overall, the standard deviation of the majority of parameters are very small regardless the choice of prior. This also explains the close performance obtained under different numbers of sampled parameters.



(a) Loss versus iterations

(b) Accuracy versus iterations

(c) First 5000 iterations of accuracy

(d) Last 5000 iterations of accuracy

Figure 7: Variational inference based classification results on MNIST dataset.

Table 7: Results of 10 repeat experiments with randomly initialized variational parameters.

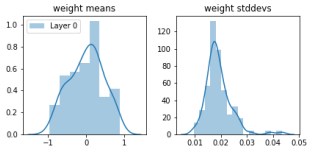
Iteration #		1	1000	2000	5000	10000	20000	50000
Sample #	5	10.59±1.65	95.86±0.21	96.53±0.25	97.00±0.08	97.13±0.07	97.18±0.11	97.18±0.07
	10	10.79±1.89	96.12±0.27	96.66±0.24	97.07±0.08	97.20±0.09	97.21±0.13	97.20±0.08
	50	10.82±1.86	96.30±0.20	96.79±0.23	97.17±0.09	97.25±0.09	97.22±0.09	97.21±0.10
	100	10.90±1.91	96.32±0.22	96.83±0.23	97.19±0.08	97.24±0.08	97.22±0.10	97.21±0.09

Table 8: VI results under different prior of parameters with 10 random initializations.

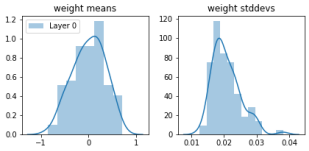
σ	0.01	0.1	1	10	100
Acc.±Std.	97.21±0.09	97.76±0.26	97.54±0.14	97.58±0.16	97.52±0.15

B.3 Use of Uncertainty

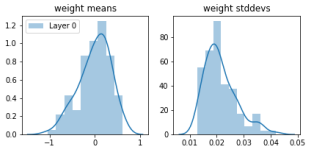
There are several directions of using uncertainty. First, through analysis and visualization, uncertainty can be used to facilitate decision making. Multiple possible results should be considered in case the prediction has a high uncertainty. Second, uncertainty can be used to improve model performance such as using uncertainty to weight the loss function in multi-task learning problem [14]. Third, uncertainty can be used in active learning problems to query a small set of labels from a large set of unlabeled data [12, 10]. Fourth, uncertainty can be used to detect anomaly. The anomaly can be viewed as data that is from a distribution outside the training data distribution [15]. Fifth, uncertainty can be used to train model against adversarial attacks, which are data designed to force mis-classification. Some recent work along this direction includes [5, 17, 23].



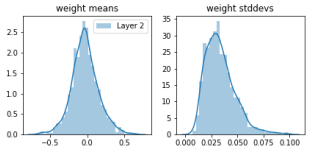
(a) Convolution layer 1. σ_0=0.1



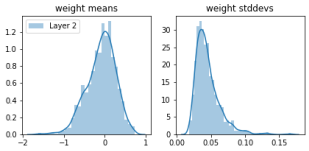
(b) Convolution layer 1. σ_0=1.0



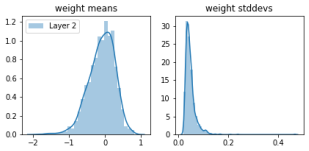
(c) Convolution layer 1. σ_0=10.0



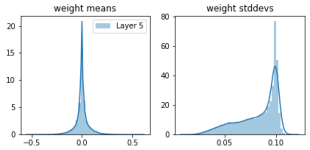
(d) Convolution layer 2. σ_0=0.1



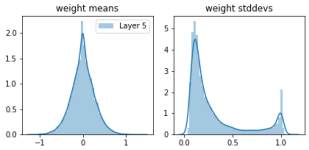
(e) Convolution layer 2. σ_0=1.0



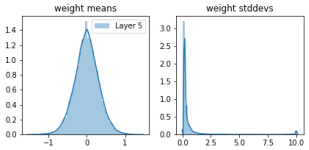
(f) Convolution layer 2. σ_0=10.0



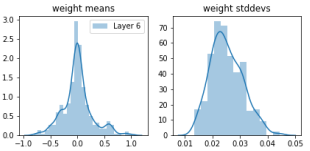
(g) Fully connected layer 1. σ_0=0.1



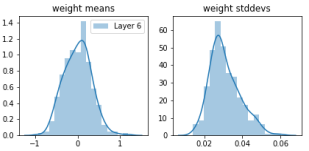
(h) Fully connected layer 1. σ_0=1.0



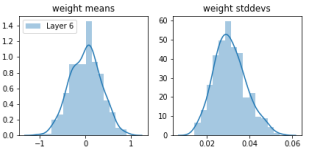
(i) Fully connected layer 1. σ_0=10.0



(j) Fully connected layer 2. σ_0=0.1



(k) Fully connected layer 2. σ_0=1.0



(l) Fully connected layer 2. σ_0=10.0

Figure 8: Variational parameters distribution after optimization.