Empirical Studies on Sensitivity to Perturbations and Hyperparameters in Bayesian Neural Networks

Homa Fashandi **Darin PW Graham**
LG Electronics Toronto AI lab
Toronto, ON, Canada
{homa.fashandi, darin.graham}@lge.com

Abstract

This paper examines the performance of two Bayesian Neural Networks, namely Monte-Carlo Dropout (MC-Dropout) and functional variational Bayesian Neural Networks (fBNN). The former specifies the prior distribution on weights, and the latter places the prior on stochastic processes (i.e., function space). We investigated their sensitivity to the change of hyperparameters, training set composition, and their ability to handle perturbed data. Their performances have been evaluated through empirical studies on the regression tasks. Moreover, their ability to detect out-of-distribution samples are examined.

1 Introduction

Deep neural network represents a parameterized function. The learning process is to obtain the point estimation of the parameters of the network. Therefore, these models are unable to provide predictive uncertainty estimates, which is crucial when they are deployed into the real world. The trained models should be able to know what they do not know rather than making overly confident decisions. Bayesian Neural Networks (BNN) combine the predictive performance of neural networks with the uncertainty estimate of the Bayesian formalism. The difficulty of employing Bayesian formalism into the neural network is the intractability of the posterior distribution. To clarify, let's specify priors on the weights of the neural network, $p(w|\alpha)$, where α represents the parameters of the prior. Let $\mathcal{D}$ be the training set, then $p(w|\mathcal{D}, \alpha)$ would be the posterior distribution. Due to the number of parameters in DNNs, the posterior distribution is intractable. One approach is to approximate the posterior with a family of distributions, $Q(\lambda)$, so that we can sample from it more efficiently or calculate the posterior analytically. Therefore, the goal is to find a member $q(w;\lambda) \in Q$ which is closest to the true posterior. Closeness is measured with Kullback-Leibler($\mathcal{KL}$) divergence between the variational posterior and the true posterior:

$$\mathcal{KL}[q(w;\lambda)||p(w|\mathcal{D},\alpha)] = \int q(w;\lambda) log \frac{q(w;\lambda)}{p(w|\mathcal{D},\alpha)} dw \tag{1}$$

The issue with equation 1 is that the true posterior is unknown. However, we have access to the joint probability distribution, $p(w, \mathcal{D}, \alpha)$, which leads to the definition of Evidence Lower Bound ($\mathcal{ELBO}$). Minimizing the $\mathcal{KL}$ divergence is equivalent to maximizing the $\mathcal{ELBO}$:

$$\mathcal{ELBO}_\lambda = \int q(w;\lambda)[log\ p(w,\mathcal{D},\alpha) - log\ q(w;\lambda)]dw \tag{2}$$

There are different implementations and interpretations of BNNs. Some of these approaches are summarized in Table A1 and Figure 1. The methods are chronologically sorted to demonstrate the technical trends in BNNs. Table A1 provides some details on the selected BNNs in terms of the prior

4th workshop on Bayesian Deep Learning (NeurIPS 2019), Vancouver, Canada.

and posterior distributions, their loss functions and the optimization processes. Early works on BNNs started by Neal and MacKay as they specified the prior distribution on weights, (Mackay, 1992), (Neal, 1992). Hinton and Van Camp (1993) added Gaussian noise to the weights to limit the amount of information they contain and applied the Minimum Description Length (MDL) principle to allow very noisy weights to be communicated efficiently. Later, Graves (2011) introduced a stochastic variational method based on the MDL. Introduction of Variational Auto-Encoders (VAE), reparametrization trick and dropout influenced the direction of BNNs, (Kingma and Welling, 2014), (Srivastava et al., 2014), (Rezende et al., 2014), (Gregor et al., 2014). Blundell et al. (2015) introduced Bayes By Backprop which uses the reparametrization trick to back propagate the gradients of the loss function to update the fully factorized Gaussian approximation to the posterior distribution. Gal and Ghahramani (2016) introduced MC-Dropout which applies dropout layer before every network unit in each layer. This work later improved with Concrete dropout, (Gal et al., 2017). Noisy versions of the off-the-shelf optimizers, Vprop, Noisy Adam and Noisy Natural Gradient, are also interpreted as BNNs, (Khan et al., 2017), (Zhang et al., 2018).

More recently, some works place the prior distributions (implicit or explicit) on the function space. Garnelo et al. (2018b) modeled conditional distribution with input data in Neural processes (NP) and Conditional NP, (Garnelo et al., 2018a). Sun et al. (2019b) introduced functional variational Bayesian Neural Networks (fBNN) which works with structured and implicit stochastic processes as priors and uses Spectral Stein Gradient Estimator (SSGE) for the gradient estimations. The work further developed into a method called infer-Net, (Shi et al., 2019). Gradient estimators for variational inference like Stein Variational Gradient Descent and SSGE helped with the transition of placing the priors on the function space, (Liu and Wang, 2016), (Shi et al., 2018).

In this paper, we take a closer look at two examples of BNNs: MC-Dropout, (Gal and Ghahramani, 2016), and fBNN, (Sun et al., 2019b). The former specifies the prior on weights and the latter places the prior over the functions. We perform a series of experiments on these two BNNs to investigate their generalization and sensitivity to hyperparameters, training set composition, and perturbations. Moreover, we examined the performance of these two methods on the detection of out-of-distribution samples. This paper is the extension and summarization of our technical reports, (Fashandi, 2019a), Fashandi (2019b). We utilized the MC-dropout and fBNN Github repositories, (Gal, 2018), (Sun et al., 2019a) for implementing the experiments. A slim version of LG Electronics Auptimizer has been used for some of the experiments presented in this paper, (Liu et al., 2019).

2 Background Information

This section describes the MC-dropout and fBNN methods in more details. The differences between the two approaches make them proper candidates for our experiments. Gal and Ghahramani (2016) used dropout in neural networks to build a BNN. They showed that a neural net with dropout applied before every weight layer is equivalent to an approximation to a probabilistic deep Gaussian process. During the test time, the weights are sampled T times and the first two moments of the outputs represent the prediction and the uncertainty. Randomly drawn masked weights can be interpreted as a function draw. The proper value of the dropout rate can be obtained with grid-search method. To obtain the dropout probability, p, with a gradient-based method instead of the grid-search approach, Gal et al. (2017) proposed the concrete dropout. MC-dropout is easy to utilize in any deep neural net architecture and with any off-the-shelf gradient estimator.

fBNN performs variational inference on the distribution of functions, (Sun et al., 2019b). fBNN is trained to produce the posterior distribution of functions with small KL divergence to the true posterior over functions; refer to Table A1 for the formulation of fELBO, the loss function of fBNN. The Spectral Stein Gradient Estimator (SSGE) is used for the gradient calculations, (Shi et al., 2018). SSGE estimates the gradient function; therefore, the gradient estimation is not limited to the sample points, and gradients can be estimated at both in-distribution samples and out-of-distribution samples. In each iteration, log-likelihood gradients and KL gradients are estimated for a set of points called measurement set, and the parameters of the fBNN are updated based on the gradients values at these points. Measurement set contains a random subset of training data, D_S, and M independently drawn point from a sampling distribution c, X^M. The reason to include X^M into the measurement set is not to overfit at the training points due to the presence of log-likelihood term in the objective. The

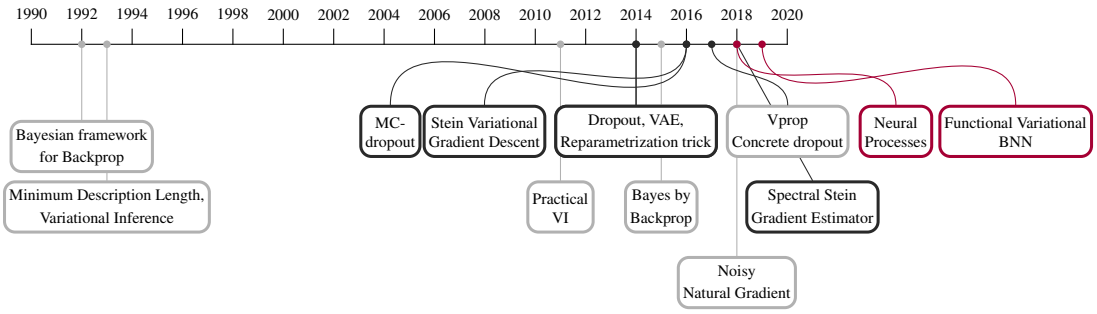


Figure 1: Timeline of Bayesian Neural Networks. The list is not exhaustive. The blocks are color coded. BNNs that put prior on weights are shown in light gray. The techniques that influence the trends are shown in darker gray. BNNs that put prior on functions are shown in red color.

priors to fBNN could be defined explicitly (e.g. Gaussian Processes) or implicitly (e.g., piecewise linear or piecewise constant functions).

3 Sensitivity to Hyperparameters

In this section, we examine the sensitivity of MC-Dropout and fBNN to their hyperparameters. Inspired by Novak et al. (2018), we performed a series of experiments to address the followings:

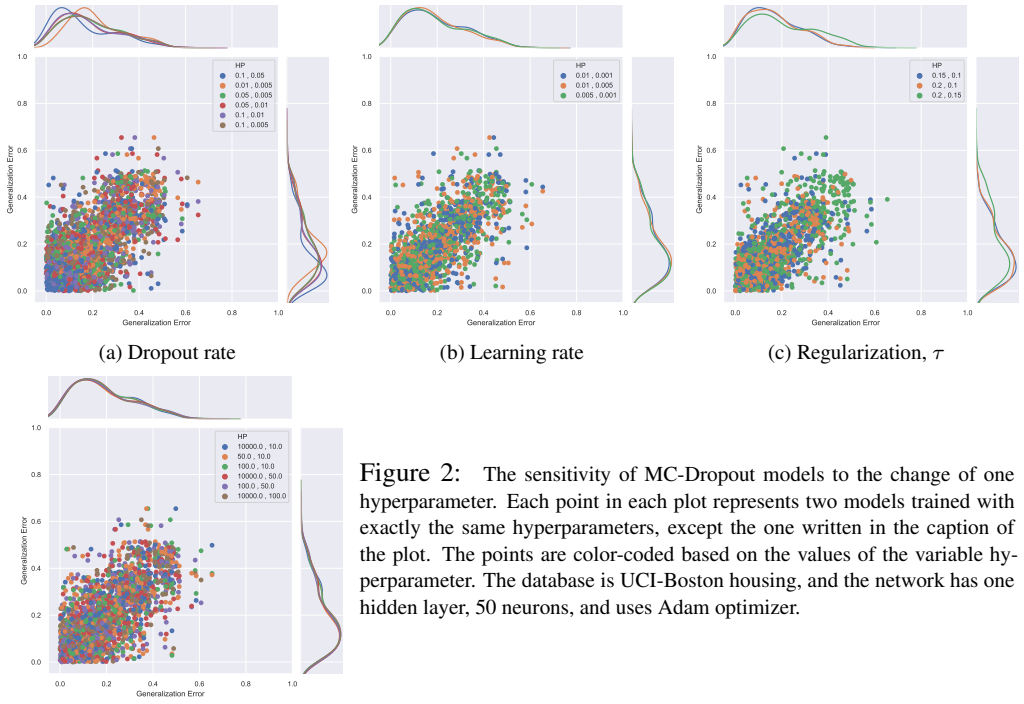
- How sensitive is the generalization error to the change of one hyperparameter?
- How sensitive is the generalization error to the hyperparameters in an unrestricted manner?

To address these questions, we trained models with different values of the hyperparameters on the UCI-Boston housing dataset, (Dua and Graff, 2019). Table 1 and Table 2 list the hyperparameters and their corresponding values. We kept the network architecture the same as the architectures used in MC-Dropout and fBNN papers for this dataset, (Gal and Ghahramani, 2016), (Sun et al., 2019b). The plots in Figure 2 and Figure 3 are to address the first question. Each plot shows the sensitivity of the model w.r.t the change of one hyperparameter, written in the plot's caption, called the variable hyperparameter. Each point, p, on these plots represent two models, A and B. These models have the same hyperparameters except for the variable hyperparameter ($h_A \neq h_B$). The point p on these plots is of the form $p = (\mathcal{G}_A, \mathcal{G}_B)$, where $\mathcal{G}_X$ is the generalization error of model X, approximated as follows:

$$\mathcal{G}(f, \mathcal{D}, \mathcal{T}) = |\mathcal{E}_{empirical}(f, \mathcal{D}) - \mathcal{E}_{empirical}(f, \mathcal{T})| \tag{3}$$

f is a trained model, $\mathcal{E}$ is the empirical error, $\mathcal{D}$, and $\mathcal{T}$ are training set and test set, respectively. The points are color-coded based on the values of the variable hyperparameter. We expect to see the points close to the identity line (i.e., $x = y$), which means that the change of one hyperparameter does not affect the generalization error too much. Based on the values of hyperparameters reported in Tables 1 and 2, there are 12096 and 1008 possible combinations of hypermeters for the fBNN and MC-dropout models, respectively. We trained all the 1008 MC-dropout models and trained 1068 fBNN models. The hyperparameter combinations for fBNN were selected semi-randomly, as we wanted to have some of the combinations that differ only in one hyperparameter.

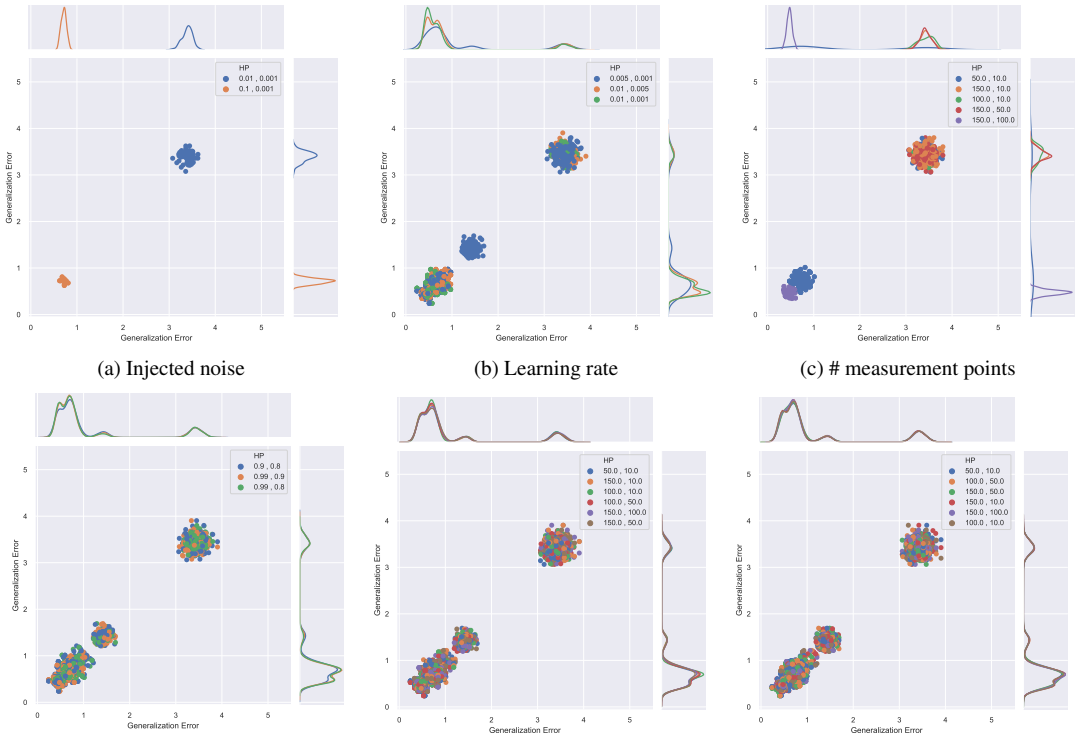
Based on the plots shown in Figures 2 and 3, fBNN and MC-dropout behave differently to the change of hyperparameters. For some of fBNN's hyperparameters, we see the clusters of points. For instance, in Figure 3-(a or c), we see single color clusters. Although the points are close to the identity line, different pairs of variable hyperparameter form separate clusters, which suggests that the models are sensitive to the different values of that particular hyperparameter. We also plotted the violin plots of the absolute difference of generalization errors between the models that only differ in one hyper parameters (i.e., $\mathcal{S} = \frac{1}{N}\sum_i^N |\mathcal{G}_{A_i} - \mathcal{G}_{B_i}|$). Figure 4-(a,b) show the average sensitivity for different hyperparameters of MC-dropout and fBNN models. Please note that the plots shows the sensitivity to each hyperparameter in isolation.



(a) Dropout rate (b) Learning rate (c) Regularization, τ

(d) Number of test samples

Figure 2: The sensitivity of MC-Dropout models to the change of one hyperparameter. Each point in each plot represents two models trained with exactly the same hyperparameters, except the one written in the caption of the plot. The points are color-coded based on the values of the variable hyperparameter. The database is UCI-Boston housing, and the network has one hidden layer, 50 neurons, and uses Adam optimizer.



(a) Injected noise (b) Learning rate (c) # measurement points

(d) Eigen threshold (e) # train samples (f) # test samples

Figure 3: The sensitivity of fBNN models to the change of one hyperparameter. Each point in each plot represents two models trained with exactly the same hyperparameters, except the one written in the caption of the plot. The points are color-coded based on the values of the variable hyperparameter. The database is UCI-Boston housing, and the network has two hidden layer, 50 neurons, and uses Adam optimizer. Not all the combinations of different hyperparameters are shown here.

Table 1: FBNN hypermarametrs and their corresponding values

fBNN, (Sun et al., 2019b)		
Hyperparameter	Values	Description
Injected noise	0.001, 0.01, 0.1	Injected noise for Gaussian process priors
Number of random measurement points	10, 50, 100, 150	In each training iteration, measurement points include a mini-batch, $\mathcal{D}_s$, from training data and random points ,X_M, from a distribution.
Learning rate	0.001,0.005,0.01	Learning rate of the optimizer
Eigen threshold	0.8, 0.9, 0.99	This is a hyperparameter of SSGE. It is a threshold on the magnitude of eigenvalues, (Shi et al., 2018).
Train samples	10, 50, 100,150	Number of samples from the posterior distribution during the training
Test samples	10, 50, 100, 150	Number of samples from the posterior distribution during the testing
Composition of training set (split number)	$0, \ldots, 6$	7-fold cross validation

Table 2: MC-dropout hypermarametrs and their corresponding values

MC-Dropout, (Gal and Ghahramani, 2016)		
Hyperparameter	Values	Description
Learning rate	0.001, 0.005, 0.01	Learning rate of the optimizer
Dropout rate	0.005, 0.01, 0.05,0.1	
Regularization rate,τ	0.1, 0.15, 0.2	
Test samples	10, 50, 100, 10000	Number of samples from the posterior distribution during the testing
Composition of training set (split number)	$0, \ldots, 6$	7-fold cross validation

To address the second question, we not only look at how the points spread from the unity line but also we consider their spread along the x or y-axis. As we would like to see the points close to the unity line and gathered around a point, rather than spreading along the unity line. We calculated the average generalization error for all the models we trained for each of the experiments. $\hat{\mathcal{G}}_{mc} = 0.21 \pm 0.13$ and $\hat{\mathcal{G}}_{fBNN} = 1.43 \pm 1.18$. If we look at the change to the hyperparameters in an unrestricted manner, fBNN shows higher sensitivity to its hyperparameters for these sets of experiments. Moreover, on average the fBNN models showed higher generalization error compared to MC-dropout models.

4 More Experiments

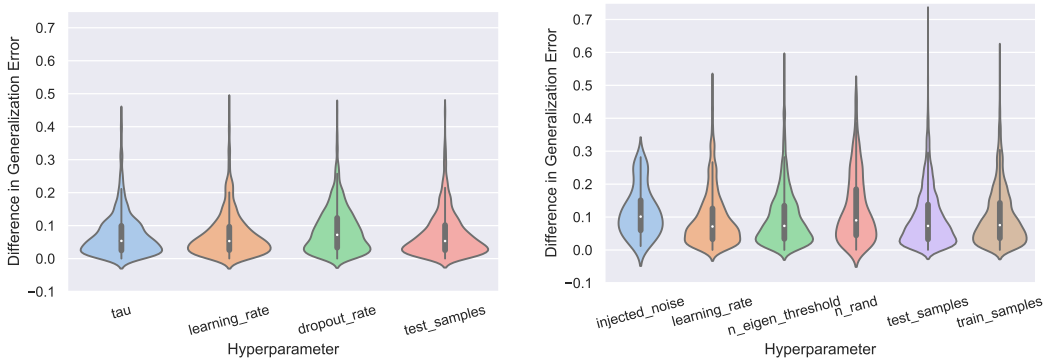
In this section, we address the sensitivity of MC-dropout and fBNN models to the network architecture, data perturbations, and composition of the training set. We selected a set of values for the number of layers, number of neurons, and weight of the noise for these experiments. For the training set composition related experiments, we used 7-fold cross validation. The split number refers to each of the 7 runs. We trained 168 fBNN models and the same number of MC-Dropout models based on the values presented in Table 3. The task is the regression task on the UCI-Boston housing dataset. The results are described in the following subsections. For the networks' hyperparameters, we used the default values in their referenced implementations, (Gal and Ghahramani, 2016), (Sun et al., 2019b).

4.1 Sensitivity to Network Architecture

We organized the results in similar plots as discussed before; Refer to Figure 5. Based on the plots presented in Figure 4-(c), fBNN shows higher sensitivity to the network architecture than MC-dropout. Addressing the generalization ability of BNNs in networks with different capacities would be an interesting investigation. As addressed in the conventional deep learning models, even very high capacity networks do not suffer from over-fitting and maintain their generalization abilities. It is

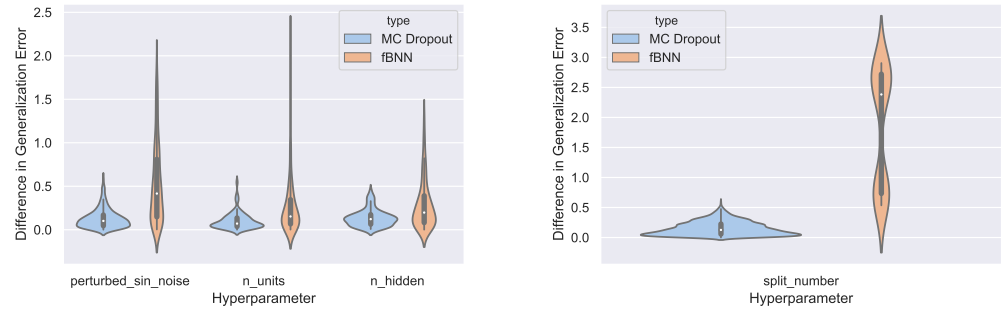
Table 3: Details of the experiments related to the network architecture and perturbation

(Hyper)parameters	Values	Description
Number of layers	1,2	Number of fully conncected layers
Number of neurons	100, 200, 300	Number of neurons in each layer
Sinusoidal noise weight,$\mathcal{A}$	0.0, 0.1, 0.5, 0.7	
Composition of training set (split number)	$0, \ldots, 6$	7-fold cross validation



(a) MC-Dropout:Hyperparams

(b) fBNN: Hyperparams



(c) Network architecture and data perturbations

(d) Training set composition

Figure 4: Average sensitivity (average difference in the generalization error) to the change of one hyperparameter, architecture, weight of the perturbation and training set composition. Split_number shows the sensitivity to the training set composition and n_rand is the number of random measurement points.

expected that BNNs achieve higher generalization abilities. It would be interesting to perform similar experiments presented in (Neyshabur et al., 2019) on different BNN implementation, as well.

4.2 Sensitivity to Data Perturbation and Training Set Composition

To explore the sensitivity of the two BNNs to the perturbations, we considered applying Sinusoidal and Gaussian perturbations to the data (to both inputs and output values). The Sinusoidal noise is defined as follows: $\hat{x} = x + \mathcal{A}sin(2\pi r)$, where $\hat{x}, x \in \mathbf{R}^n$ are the perturbed data and original data, respectively. $\mathcal{A} \in (0, 1]$ is a constant, and $r \in \mathbf{R}^n$, $r_i \in [0, 1]$, Yang et al. (2015). The Gaussian perturbation is the defined by: $\hat{x} = x + \mathcal{N}(0, \mathcal{A})$, where the mean value is zero and standard deviation is $\mathcal{A}$. We call $\mathcal{A}$ the weight of perturbation in both cases.

To closely investigate how fBNN and MC-dropout models learn in a noisy environment, we performed a series of experiments on a toy example presented in (Sun et al., 2019b). Figures 7 and 8 shows samples from $y = x^3$ with added Sinusoidal and Gaussian noise to the training data, with a different value of $\mathcal{A}$ in each plot. The network architecture for both BNNs is one hidden layer of 50 neurons. The blue regions show the uncertainty up to ± 3.75 times the standard deviation. The training data partitioned the input space into different regions: outer partitions, inner partition, and partitions with training samples.

fBNN model shows higher uncertainty in the inner region than the MC-dropout model. The uncertainty values in the outer regions are higher for MC-dropout model than fBNN model. It seems that MC-dropout treats the inner partition differently that the outer partitions. This observation needs further analysis, and it would be an interesting investigation specially when we move to a higher dimensional space. Novak et al. (2018) demonstrated that conventional (i.e., non-Bayesian) trained neural networks behave differently on and off the data manifold. Those studies are in-line with our

experiments and the predicted uncertainty values in both BNNs. Around the training data, fBNN shows higher confidence than MC-Dropout which results in a less accurate predicted mean function in the presence of lower signal to noise ratio; e.g., compare figure 7-(c and e) with figure 8-(c and e). This issue could be due to the presence of the log-likelihood term in the fBNN's objective function and due to updating the objective function based on the gradient values in a finite measurement set. Refer to section 2 for more details. We also perturbed the UCI-Boston housing data with Sinusoidal noise and plot the generalization errors of the models that only differ in the amount of perturbations; Refer to Figure 6-(a and c).

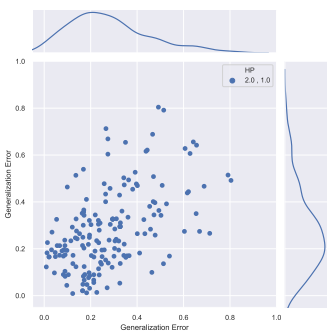
Similar plots are shown for the different split number (i.e., cross-validation folds). Each point shows the generalization errors of two models with the same hyperparameters, except different training set subset (i.e., fold). The points are color-coded. It is interesting to see that points with different colors are more separated in fBNN than MC-Dropout models. Compare Figure 6-(b) and Figure 6-(d). This shows the higher sensitivity of the fBNN models to the training set composition. Again, perhaps it is related to the loss function formulation. To further investigate the sensitivity to the training set composition, we summarized the sensitivity to the training set composition of all the experiments in Figure 4-(d).

5 Out-of-distribution Samples

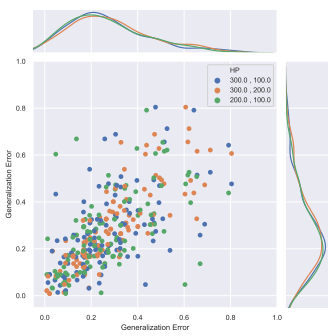
One application of the uncertainty measurements in BNNs is the detection of out-of-distribution (OOD) samples. We expect to see a highly uncertain output for an OOD sample. To empirically investigate the performance of MC-Dropout and fBNN, we chose sets of two different regression databases from UCI-datasets (Dua and Graff, 2019), with the same number of input features; one as in-distribution data and the other one as an out-of-distribution data. We trained the BNNs on concrete dataset, (Yeh, 1998). The out-of-distribution dataset is Energy efficiency Data Set, (Tsanas and Xifara, 2012). We only considered the heating load output of this dataset, as the training set has only one output value. Figure 9 shows the average uncertainty for the in-distribution test data and out-of-distribution test data during the training process. We trained the networks for 10 runs. The uncertainty value of a test sample is measured with the variance of the multiple (=100 draws) predictions for the test sample. The average of these uncertainty values for the test sets for each run are plotted in Figure 9. As expected, the uncertainty values of the OOD samples are higher than the in-distribution test samples for both implementations. It is interesting to note that for all the 10 runs the MC-dropout uncertainty values for the OOD samples are higher ($\sim$ order of 10) than the fBNN models. Another interesting observation is that for one of the MC-dropout runs the average uncertainty values of OOD samples were initially close to the in-distribution test samples, but the gap increased as the training continued; See Figure 9-(b).

6 Conclusion

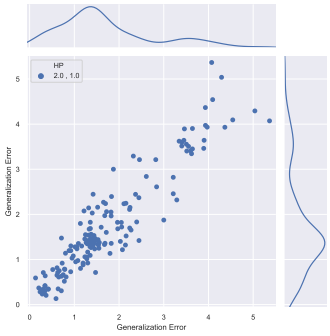
In this paper, we examined two implementations of BNNs. The nature of gradient estimation, loss function, prior specification, and posterior approximation were different in the two selected BNN implementations. We designed the set of experiments to compare the sensitivity and generalization abilities of them. Overall, we concluded that MC-dropout showed lower generalization error in these experiments than fBNN. It was less sensitive to its hyperparameters, the data perturbation, and the training set composition. Perhaps, the formulation of the loss function of fBNN plays a role in this, as the KL-term measured in fELBO is based on the finite measurement sets. This set contains randomly drawn samples in addition to the subset of the training set to overcome the over fitting to the training points. However, the finite nature of this set affects the generalization ability of the network in our experiments. Moreover, we showcased the quality of their predictive uncertainty in out-of-distribution samples.



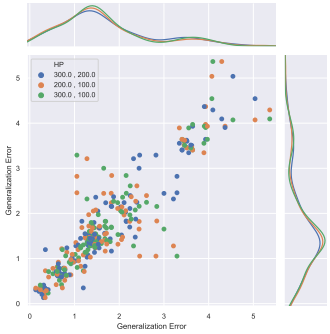
(a) # hidden layers



(a) # neurons per layer

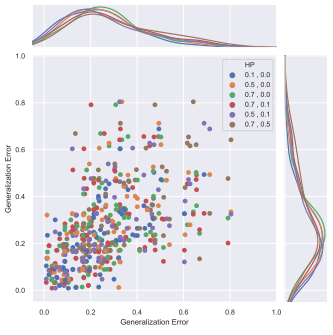


(c) # number of hidden layers

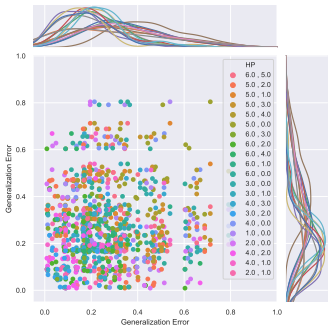


(d) # neurons per layer

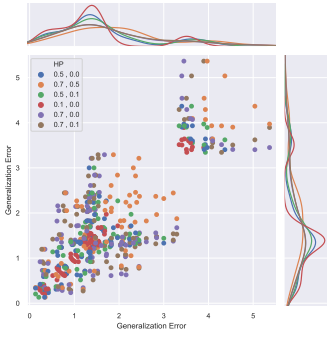
Figure 5: The sensitivity of (a-b) MC-Dropout and (c-d) fBNN models to the model architecture (i.e., number of hidden layers and number of neurons in each layer). Please note the difference in the plots' axis limits.



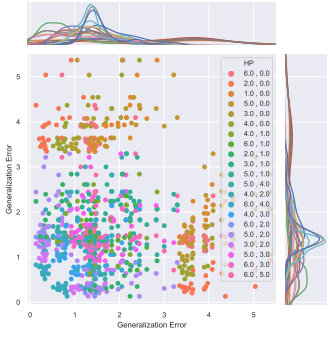
(a) Sinusoidal noise perturbation



(b) split number

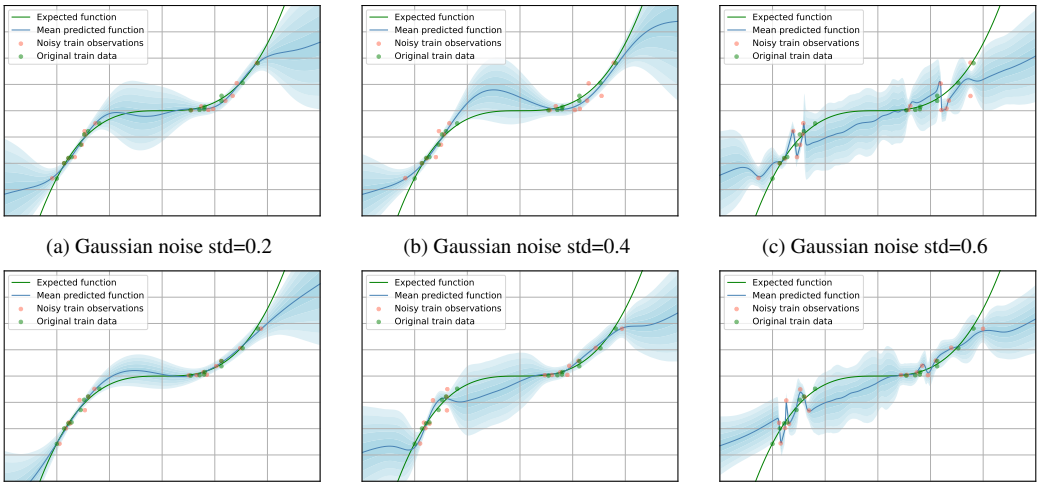


(c) Sinusoidal noise perturbation



(d) split number

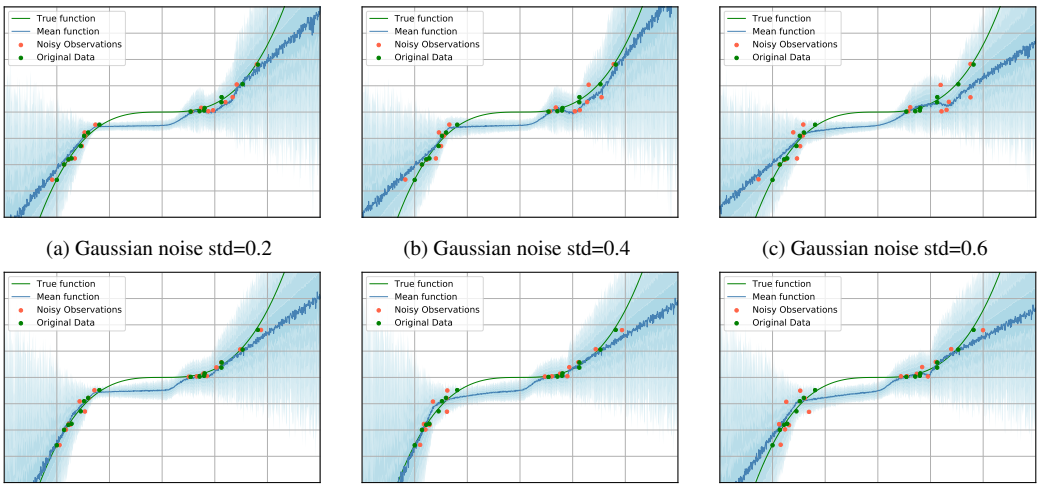
Figure 6: The sensitivity of (a-b) MC-dropout and (c-d) fBNN models to input perturbation, and training set composition. Please note the difference in the plots' axis limits.



(a) Gaussian noise std=0.2 (b) Gaussian noise std=0.4 (c) Gaussian noise std=0.6

(d) Sinusoidal noise, A=0.2 (e) Sinusoidal noise, A=0.4 (e) Sinusoidal noise, A=0.6

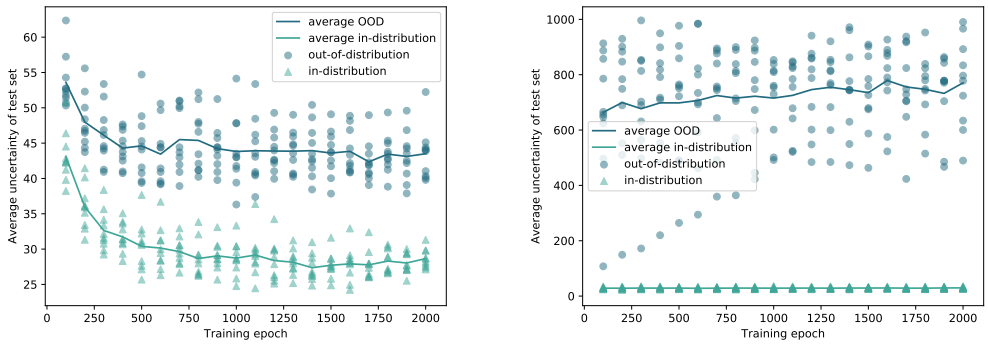
Figure 7: Sensitivity of fBNN models to data perturbation



(a) Gaussian noise std=0.2 (b) Gaussian noise std=0.4 (c) Gaussian noise std=0.6

(d) Sinusoidal noise, A=0.2 (e) Sinusoidal noise, A=0.4 (e) Sinusoidal noise, A=0.6

Figure 8: Sensitivity of MC-Dropout models to data perturbation



(a) FBNN (b) MC-Dropout

Figure 9: Uncertainty measurements of OOD samples

References

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wiestra, D. (2015). Weight uncertainty in neural networks. In *ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning*, pages 1613–1622.

Dua, D. and Graff, C. (2019). UCI machine learning repository. Available at: http://archive.ics.uci.edu/ml.

Fashandi, H. (2019a). Analysis of functional variational bayesian neural networks: Phase one. Technical report, LG Electronics Toronto AI Lab.

Fashandi, H. (2019b). Analysis of MC-dropout. Technical report, LG Electronics Toronto AI Lab.

Gal, Y. (2018). github page: Dropout uncertainty exps. Available at: https://github.com/yaringal/DropoutUncertaintyExps.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on Machine Learning(ICML)*.

Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3581–3590. Curran Associates, Inc.

Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Whye Teh, Y., Rezende, D. J., and Eslami, S. M. A. (2018a). Conditional neural processes. In Dy, Jennifer and Krause, A., editor, *Proceedings of the 35th International Conference on Machine Learning*, pages 1704–1713, Stockholmsmässan, Stockholm Sweden. PMLR.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. (2018b). Neural processes. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*.

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

Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1242–1250.

Hinton, G. E. and Van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *COLT '93 Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, Santa Cruz, California, USA. ACM New York, NY, USA.

Khan, M. E., Liu, Z., Tangkaratt, V., and Gal, Y. (2017). Vprop: Variational inference using rmsprop. In *Neural Information Processing Systems*.

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

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, Banff, AB, Canada.

Liu, J., Tripathi, S., Kurup, U., and Shah, M. (2019). Auptimizer - an extensible, open-source framework for hyperparameter tuning. In *IEEE International Conference on Big Data*.

Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2378–2386, USA. Curran Associates Inc.

Mackay, D. J. C. (1992). A practical bayesian framework for backprop networks. *Neural Computation*, 4(3):448–472.

Neal, R. M. (1992). Bayesian training of backpropagation networks by the hybrid monte carlo method.

Neyshabur, B., Li, Z., Bhojanapalli, S., Lecun, Y., and Srebro, N. (2019). Towards understanding the role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations (ICLR)*.

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: an empirical study. In *6th International Conference on Learning Representations, ICLR*, Vancouver, BC, Canada.

Plaut, D. C., Nowlan, S. J., Hinton, G. E., and Nowlan, S. (1986). Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Computrt Science Dpartment, Carnegie-Mellon University, Pittsburgh, PA, 15213.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, pages II–1278–II–1286, Beijing, China.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE International Conference on Neural Networks*, San Francisco, CA, USA, USA.

Shi, J., Khan, M. E., and Zhu, J. (2019). Scalable training of inference networks for gaussian-process models. In *International Conference on Machine Learning*, pages 5758–5768.

Shi, J., Sun, S., and Zhu, J. (2018). A spectral approach to gradient estimation for implicit distributions. In *Thirthy-fifth International Conference on Machine Learning (ICML)*, Stockholmsmässan, Stockholm, Sweden.

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

Sun, S., Zhang, G., and Shi, J. (2019a). github page: Functional variational bayesian neural networks. Available at: https://github.com/ssydasheng/FBNN#functional-variational-bayesian-neural-networks.

Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019b). Functional variational bayesian neural networks. In *Seventh International Conference on Learning Representations (ICLR)*, pages 1–22.

Tsanas, A. and Xifara, A. (2012). Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567.

Yang, J., Zhang, P., and Liu, Y. (2015). Robustness of classification ability of spiking neural networks. *Nonlinear Dynamics*, 82(1):723–730.

Yeh, I. C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808.

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. (2018). Noisy natural gradient as variational inference. *International Conference on Machine Learning*.

Appendices

A Bayesian Neural Networks

Table A1 provides an overview of BNNs. The table is chronologically sorted to demonstrate the technical trends in the field. Please note that the list is not exhaustive.

Table A1: Overview of Bayesian Neural Networks

Reference	Prior	Posterior	Loss Function	Optimization/Sampling
(Neal, 1992)	Similarity shaped logistic distribution: $w_i = \frac{s g_i}{1-s g_i} . log(\frac{a_i}{1-a_i})$ g_i identifies a group of weights that w_i belongs to, s_g is the scaling parameter.	$P(y_{n+1}\|x_{n+1}, (x_1, y_1), \ldots, (x_n, y_n))$ $= \int P(y_{n+1}\|x_{n+1}, w) \times$ $P(w\|(x_1, y_1), \ldots, (x_n, y_n) dw$		Hybrid Monte Carlo & Simulated Annealing
(Graves, 2011)	Uniform, Laplace, or Gaussian distribution	Delta distribution, when the prior is Uniform, Laplace or Gaussian. Gaussian when prior is Gaussian.	$\mathcal{L}(\alpha, \beta, \mathcal{D}) =$ $\Big\langle - \sum_{(x,y)\in\mathcal{D}} \ln Pr(y\|x, w) \Big\rangle_{w\sim Q(\beta)}$ $+ KL(Q(\beta)\|\|P(\alpha))$ where KL is the Kullback-Leibler divergence between $Q(\beta)$ and $P(\alpha)$. Q is variationl posterior, P is the prior distribution, α and β are the parameters of the distributions.	Back propagation with momentum, (Plaut et al., 1986), and Resilient Propagation(RPROP), (Riedmiller and Braun, 1993), with weight pruning
Bayes By Backprop (BBB) (Blundell et al., 2015)	Gaussian prior on weights: $P(w) = \prod_i \mathcal{N}(w\|0, \sigma^2)$ or, Mixture of Gaussians: $P(W) = \prod_i \pi\mathcal{N}(w_i\|0, \sigma_0^2)$ $+(1-\pi)\mathcal{N}(w_i\|0, \sigma_1^2)$	Fully Factorized Gaussian: $q(w\|\theta) = \prod_i \mathcal{N}(w_i\|\mu, \sigma^2)$	$\mathcal{L}(\mathcal{D}, \theta) \approx \sum_{i=1}^n log q(w_i\|\theta)$ $- log P(w_i) - log P(\mathcal{D}\|w_i)$ where $\mathcal{D}$ is the training set, and w_i is the $i-$the MC sample drwan from $q(w\|\theta)$	Stochastic gradient descent in conjunction with reparameterization trick, (Kingma and Welling, 2014)
MC-Dropout, (Gal and Ghahramani, 2016)	Gaussian prior on weights	$q(w\|\theta) = \prod_i q_{M_i}(W_i) = \prod_i M_i \cdot diag([z_{i,j}])$ where $z_{i,j} \sim Bernoulli(p_i)$, and $M_i = mean(W_i)$	$\mathcal{L}_{dropout} = \frac{1}{N} \sum_{i=1}^N E(y_i, \hat{y}_i) +$ $\lambda \sum_{i=1}^L (\|\|W_i\|\|_2^2 + \|\|b_i\|\|_2^2)$	Adam optimizer, (Kingma and Ba, 2015)
Vprop, (Khan et al., 2017)	Gaussian prior over weights, $p(w) = \mathcal{N}(0, \frac{I}{\lambda})$, where $\lambda > 0$	Fully factorized Gaussian , $q(w) = \mathcal{N}(\mu, diag(\sigma^2))$	$\mathcal{L}(\mu, \sigma) = \max_{\mu,\sigma} \mathbb{E}_q [log p(y\|X, w) + log p(w)$ $- log q(w)]$	Vprop adds weight noise to RMSprop
Noisy Natural Gradient (NNG), Noisy Adam, (Zhang et al., 2018)	Gaussian prior on weights	Fully factorized Gaussian for Noisy Adam, Matrix variate Gaussian for Noisy K-FAC	$\mathcal{L}(q) = \mathbb{E}_q [log p(\mathcal{D}\|w)] - \lambda KL(q(w)\|\|p(w))$ $KL(q\|\|p)$ is the Kullback-Leibler divergence between q and p, λ is a regularization parameter.	Noisy Adam, and Noisy K-FAC
Neural Process, (Garnelo et al., 2018b)	Data driven priors	$g(., \phi)$, where g is the network and ϕ are network's parameters	$\mathcal{L}(q) = \mathbb{E}_{q(z\|s_T)} [log p(y_T\|x_T, r_C, z)$ $- KL(q(z\|s_T)\|\|q(z\|s_C)]$ where z is the latent variable, s_C represent context points, r_C is the representation of context points, and s_T represent target points.	Adam optimizer, (Kingma and Ba, 2015)
Functional Variational BNN (fBNN), (Sun et al., 2019b)	• Explicit GP prior on functins, $\mathcal{GP}(\mu_p(.), k_p(., .))$ • Implicit prior on functions: Piecewise constant or piecewise linear functions	$g(.; \phi)$, where g is the network and ϕ are the parameters.	$\mathcal{L}(q) = \mathbb{E}_q [log p(\mathcal{D}\|f)]$ $- \sup_{n\in\mathbb{N}, X\in\mathcal{X}^n} KL\left(q(f^X)\|\|p(f^X)\right)$ where $\mathcal{D}$ is the training set. X is a finite measurement set. $KL(q\|\|p)$ is the Kullback-Leibler divergence between q and p.	Spectral Stein gradient estimator, (Shi et al., 2018) and Adam optimizer, (Kingma and Ba, 2015)