A Study on Bayesian Neural Network in Regression and Classification Tasks

Shaoyu Feng
Georgetown University
Washington, DC
sf865@georgetown.edu

Yunjia Zeng
Georgetown University
Washington, DC
yz682@georgetown.edu

Abstract

This paper shows the fundamental concepts of Bayesian Neural Network(BNN) and discusses in details about challenges in optimization of a BNN and overview a standard approach in training a BNN. The paper applied the BNN concepts into both regression and classification tasks using two different implementation methods. The results demonstrate that BNN is able to achieve similar predictive power of traditional MLP, and is capable of capturing epistemic uncertainties during training process. Another finding of the paper is that BNN is able to better handle unseen data in inference, which shows great potential practical applicability.

Keywords: Bayesian Neural Network, KL Divergence, ELBO, Reparametrization, Uncertainty, PyTorch, Pyro

1. Introduction

In advancement of computer hardware and computational power, there has been a tremendous research trend on employing deep learning techniques, represented by neural networks, to solve challenging problems like computer vision, natural language understanding etc. And we have observed unprecedented performance of those techniques in different functional areas. However, critiques have been demonstrated that traditional neural network architectures are prone to overfitting (N.Srivastava et al. 1), and thus various regularization methods has been developed such as early stopping, weight decay and dropout (N.Srivastava. et al. 1; Blundell et al., 1). Moreover, traditional neural networks do not address the uncertainties in model weights, and provide no useful measures in the form of confidence in predictions or results (Blundell et al., 1).

In light of those shortcomings for traditional neural networks, research works have been carried out to apply Bayesian paradigm into the neural network architecture, which make it possible to assign a probability distribution towards model weights of the network (Blundell et al., 2). Those types of neural networks with uncertainties over the weights is normally known as Bayesian Neural Network (BNN). However, there are also a few challenges in constructing an effective BNN, including setting up the reasonable prior, selecting the variational inference method and thus approximating the loss function and performing backpropagation on the network to perform gradient descent (Neal, 22).

This study focuses on the underlying essential theory foundations of BNN including Bayesian Backpropation in Neural Networks, the set up of loss functions, posterior inference and prediction, and etc. We will also build BNNs for both regression and classification tasks using simulated data and MNIST data sets with Pytorch and Pyro. At the end we make comparisons between the BNNs and traditional neural netwrok methods.

Section 2 of the paper will show the related work in BNNs. Section 3 will overview the essential theory setup for BNNs and crucial steps in training a BNN possible. Section 4 briefs about the data sets used in the application. Section 5 will show the application of BNNs built on top of Pytorch on both regression and classification tasks, and the relevant analysis of the results. The paper will conclude with further discussion in section 6.

2. Related Work

In Bayesian inference in BNN, the very starting point is to choose the prior distribution over the model parameters, which in BNN are the connections weights between different layers and the bias. Most past works on Bayesian inference for neural network have used independent Gaussian distribution, which make it possible to make the training process to converge to a Gaussian process (Neal, 34; MacKay, D, 448). Recent researchers have also focused on more intricate effect at unit level, state that the induced prior distribution becomes increasingly heavy-tailed with the depth of the neural network layers (Mariia et al ,1). However, the selection of priors remains ad-hoc (Neal, 17) and mixed priors could lead to a significant improvement in model performance (Blundell et al., 5).

A BNN normally uses standard normal as prior and calcu-

lates the posterior distribution of weights and biases using the training data. The entire posterior distribution of weights and biases are used to make predictions on test data by taking the expectations of the weights, which is equivalent to take an ensemble of infinite networks (Blundell et al., 3). Despite the theoretical advantages, this adds extra difficulty to train the model in practice. Bayesian inference on the weights of a neural network is intractable as the number of parameters is very large and the functional form of a neural network does not allow easy integration for weights update (Miller et al, 1). Therefore, Bayesian variational inference method has been applied in approximating posterior distribution on the weights. In particular, the usage of variational approximation attempts to find the distribution over model weights, which minimizes the Kullback-Leibler (KL) Divergence (Miller et al, 3; Blundell et al.,3).

However, KL divergence optimization is intractable as it directly depends on posterior distribution (Miller et al, 3). Another quantity called Evidence Lower Bound (ELBO) which can deduce a lower bound for KL divergence has been added, and we could utilize reparameterization gradient estimators (RGEs) to calculate the gradient of ELBO and can enable gradient descent optimization during network training (Miller et al, 3-6). The resulting cost function is variously known as the variational free energy or the expected lower bound (Blundell et al.,3).

Compare to traditional neural networks, BNNs win advantages in following ways: regularization over the weights to reduce overfitting (Blundell et al., 8), posterior inference on the weights for prediction, and useful measures of uncertainty attributed to unseen predictor space or heteroskedasticity (Neal, 47). BNNs have been recently adopted in various areas in deep learning research. The uncertainty in BNNs can be used to drive exploration in contextual bandit problems using Thompson sampling in reinforcement learning (Blundell et al.,5). Bayesian GAN has been shown to outperform many GAN architectures like DCGAN, which ensembles on many benchmark datasets (Sattchi, 1).

3. Bayesian Neural Networks (BNNs)

3.1 Overview

As discussed earlier, BNN normally employs standard normal as prior distributions of weights and biases and calculates the posterior distributions of weights and biases (together represented as w for simplicity) using the training data (X). In brief, the entire objective for BNN is to calculate the posterior distributions of weights and biases $P(w|X)$. The learned distribution can then be employed to make predictions on unforeseen observation: for an observation with unknown label y and predictors X, the prediction y^* is produced by a forward pass through the trained BNN network, using weights (w^*) corresponding to the specified loss function (posterior mean for MSE loss and posterior median for MAE loss).

$$\hat{y} = BNN(X, w^*) \tag{1}$$

Or we could leverage the entire posterior distribution of weights and bias: predicting by taking the expectation over the weights:

$$\hat{y} = E_{P(w|D)}\hat{y}|\hat{X}, w = \int BNN(\hat{X}, w)P(w|D)dw \tag{2}$$

However, it is clearly computationally exhaustive to calculate the entire posterior distribution over every single weight and bias, especially when there is a complex network structure. Therefore, Bayesian variational methods need to be applied to approximate a lower bound of KL Divergence between the true posterior distribution and our predicted posterior distribution.

3.2 Bayesian Backpropagation

As mentioned above, we need to find a variational approximation to the posterior distribution on the weights. In particular, the usage of variational approximation attempts to find the distribution over w, Q(w|θ), which minimizes the KL Divergence.

3.2.1 Kullback-Leibler (KL) Divergence

The KL Divergence is a numeric measure of the divergence between two distributions. For two continuous probability distributions Q and P, the KL divergence from Q to P is defined as:

$$\begin{aligned} D_{KL}(Q||P) &= \int_{-\infty}^{\infty} q(x) log \frac{q(x)}{p(x)} dx \\ &= E_{q(x)}[logq(x) - logp(x)] \end{aligned} \tag{3}$$

More specifically, we are trying to get a variational approximation $Q(w|\theta)$ to the posterior distribution $(P(w))$ on the weights of BNN. In short, we are looking for the best parameter such that the KL Divergence is minimized:

$$\theta^* = argmin_\theta E_{Q(w|\theta))}[logQ(w|\theta) - logP(w|\theta)] \tag{4}$$

However, this make the optimization using gradient descent intractable because the gradient is directly dependent on the posterior distribution. Therefore, we will have to introduce another quantity called Evidence Lower Bound (ELBO) for the KL divergence.

3.2.2 The Evidence Lower Bound (ELBO)

If we look at $logP(D)$, we know that it is logarithm of the

marginal likelihood of the data, therefore it is a fixed constant that is dependent on data.

$$\begin{aligned}
logP(D) &= log \int P(D,w)dw \\
&= log \int P(D) \int Q(w|\theta)d\theta dw \\
&= \int \int logP(D)Q(w|\theta)d\theta dw \\
&= \int \int log\frac{P(D)P(w|D)}{P(w|D)}Q(w|\theta)d\theta dw \\
&= \int \int log\frac{P(w,D)}{P(w|D)}Q(w|\theta)d\theta dw \\
&= \int \int [log\frac{P(w,D}{Q(w|\theta)} + log\frac{Q(w|\theta)}{P(w|D)}]Q(w|\theta)d\theta dw \\
&= \int \int log\frac{P(w,D)}{Q(w|\theta)}Q(w|\theta)d\theta dw \\
&+ \int \int log\frac{Q(w|\theta)}{P(w|D)}Q(w|\theta)d\theta dw \\
&= E_{Q(w|\theta)}[log(P(w,D) - logQ(w|\theta)] \\
&+ D_{KL}[Q(w|\theta)||P(w|)]
\end{aligned} \tag{5}$$

We observe from above that the second term is the KL Divergence that we would like to minimize, and given that $logP(D)$ is a constant, we can minimize the $D_{KL}(Q||P)$ by maximizing the Evidence Lower Bound:

$$\begin{aligned}
ELBO(\theta) &= E_{Q(w|\theta)}[log(P(w,D)) - logQ(w|\theta)] \\
&= E_{Q(w|\theta)}[log(P(w,D))] - E_{Q(w|\theta)}[logQ(w|\theta)]
\end{aligned} \tag{6}$$

Thus maximizing ELBO is equivalent to maximizing the first term $E_{Q(w|\theta)}[log(P(w,D))]$, which indicates that we need to wellfit the data. On the other hand, we will have to minimize the second term $E_{Q(w|\theta)}[logQ(w|\theta)]$, which discourages the distribution Q becomes too concentrated.

However, to make the gradient descent possible, we will have to calculate the gradient of ELBO with respect to θ. But according to Mill et al. (2015), the gradient inherits the ELBO's form as an expectation, which is in general an intractable quantity to compute. Therefore, we will have to apply reparameterization gradient estimators (RGEs) computed using the reparameterization trick to solve the problem.

3.2.3 Reparameterization Trick

To make the calculation of gradients become more tractable, we transform the form of representation of the variational distribution of the weights of BNN. We could perform a softplus transformation by rewriting $\sigma = log(1 + exp(\rho))$ for $\omega \sim N(\mu, diag(\sigma^2))$ to maintain non-negative σ, such that $\omega = \mu + log(1 + exp(\rho)) \bigodot \epsilon$, where $\epsilon \sim N(0, I)$. Note that the $\bigodot$ represents the element-wise product. Thus, we separated the original distribution of ω into two parts. In the following equations,

$$\theta = [\mu, \rho]\epsilon \sim N(0, I)\omega \sim T(\epsilon, \theta), \tag{7}$$

the second equation is independent of θ, the variational parameters, and the third equation is dependent on the first equation. Therefore, each step of optimization to maximize ELBO on a particular data point x will proceed as follows under our setup (Blundell et al., 2015):

1. Draw $\epsilon \sim N(0, I) = q(\epsilon)$. This is used to denote the distribution of ϵ.
2. Let $\omega = \mu + log(1 + exp(\rho)) \bigodot \epsilon$.
3. Let $\theta = [\mu, \rho]$.
4. Loss function
$$\begin{aligned}
f_x(\omega, \theta) &= E_{q(\epsilon)}(logP(\omega, x) - logQ(\omega|\theta)) \\
&= logQ(\omega|\theta) - logP(x, \omega) \\
&= logQ(\omega|\theta) - logP(\omega)P(x|\omega)
\end{aligned} \tag{8}$$
5. Compute the gradient with respect to μ,
$$\nabla_\mu(x) = \frac{\partial f(\omega, \theta}{\partial \omega} + \frac{\partial f(\omega, \theta)}{\partial \mu} \tag{9}$$
6. Compute the gradient with respect to ρ,
$$\nabla_\rho(x) = \frac{\partial f(\omega, \theta}{\partial \omega} \frac{\epsilon}{1 + exp(-\rho)} + \frac{\partial f(\omega, \theta)}{\partial \rho} \tag{10}$$
7. Update the variational parameters using the user-defined learning rate α:
$$\begin{aligned}
\mu' &= \mu - \alpha\nabla_\mu \\
\rho' &= \rho - \alpha\nabla_\rho
\end{aligned} \tag{11}$$

Similarly for training data, the calculation of gradient of mini-batch is to average the gradients computed using the sample, such that

$$\begin{aligned}
\hat{\nabla_\mu}(S) &\approx \frac{1}{||S||} \sum_{x \in S} \nabla_\mu(x) \\
\hat{\nabla_\rho}(S) &\approx \frac{1}{||S||} \sum_{x \in S} \nabla_\rho(x)
\end{aligned} \tag{12}$$

where S represents one mini-batch of training data. The unbiased reparameterization gradient (Miller et al., 2015) provides theoretical foundation for the use of stochastic gradient decent to perform gradient updates for BNN.

3.3 Posterior Inference and Prediction

We can now perform the posterior inference and predictions using the trained BNN on stack of the training process that we have discussed previously. We have approximated the actual posterior distribution with the variational distribution $Q(\omega|\theta^*)$ on weights given the training data $P(\omega|D)$. We will use posterior median of the weights to minimize MAE

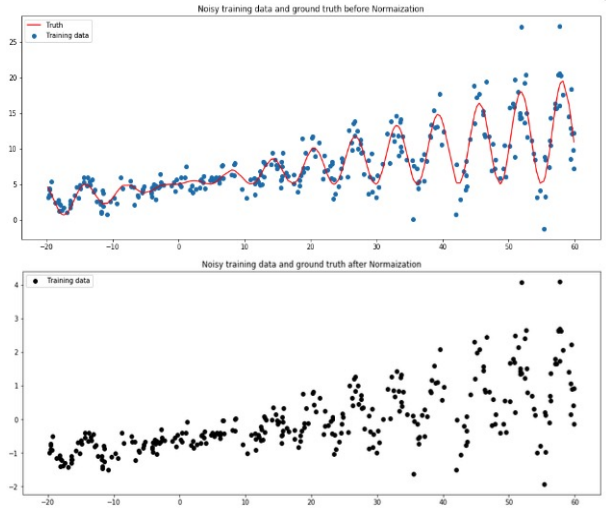

Figure 1: Regression Data Raw and After Nomalization

and use posterior mean of the weights for the BNN to minimize MSE depend on loss function. Furthermore, to access uncertainty and to potentially improve model generalization to unseen data by performing model ensembles, we could construct different models with randomly drawn weights from the posterior distributions in a very computationally efficient manner.

4. The DataSets

We will build BNNs for both regression and classification tasks for application.

4.1 Data Set for Regression

For the regression task, we use a simulated data set that created by

$$y = w * x * (1 + sin(x)) + b + \epsilon \tag{13}$$

where ϵ is a random noise term that dependent on x. Since there is a sinusoidal function involves in the data generation, we normalize the data before training. Figure 1 shows the raw data point and the normalized data.

4.2 Data Set for Classification

For the classification task, we chose to use the benchmark data set MNIST, which is gray scale images with size 28*28 and label 0-9, and has been built in Pytorch community. There are 50000 training images and 10000 testing images for the classification. In addition to the original dataset, in order to access the uncertainties of the model, we have introduced MNIST-alphabet, which is used to test whether the model is able to refuse prediction on the noise data set. We will further discuss this part in section 5.2.

Table 1: MLP and BNN on Regression Task

Model	Run Time	MSE Loss
MLP	7.8s	0.2513
BNN	38.5	0.2609

5. The Model and Results Analysis

Two models are built to apply BNNs on regression and classification tasks respectively. Both implementations are developed on top of PyTorch. The regression network is written in low-level Pytorch, we have customized the BNN layers and wraps the model using Pytroch. The classification implementation leveages on a higher-level language called Pyro, which is a probabilistic programming language written on top of Pytorch.

5.1 Regression On Simulated Data Set

The entire model is built on top of Pytorch, we've implemented a BNN-layer class to incorparte the reparameterization trick and calculate loss during each training batch.

For model architecture, we build a BNN model with one input layer, two hidden layers (each has a weight and a bias term) with ReLu Activation, and one output layer. For all the priors, we have assumed a Gaussian Distribution with zero mean and a variance of 0.001. We look at the results in two dimensions: (a) A comparison with Multi-layer Perceptron(MLP) (b) The Epistemic uncertainty of BNN over iterations.

5.1.1 Comparison with MLP

Despite the uncertainty measures that we are interested in, we first compare the regression results with BNN against the traditional neural network. Other than the BNN netwrok, we have also implement a MLP with exact same network structure. We have trained both of them using the same number of iterations and learning rate. Figure 2 and Table 1 shows the performance comparisons of the two models.

From the results, we see that the MSE for the two models are comparable, both models are able to achieve a reasonably good prediction on the same data. However, the run time for BNN is much longer than MLP. This is expected because of the variational methods set-up.

5.1.2 The Epistemic uncertainty of BNN

Epistemic uncertainty is the scientific uncertainty in the

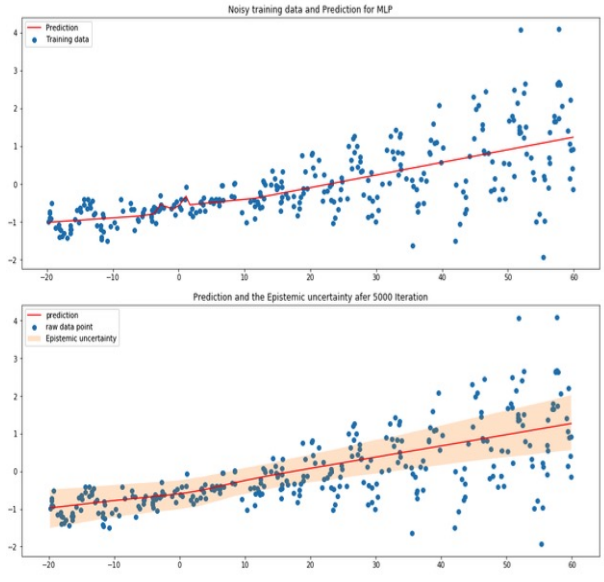

Figure 2: Prediction Results for MLP and BNN

model of the process. The epistemic uncertainty is characterized by alternative models. One of the greatest advantages of BNNs is that we are able to model the Epistemic uncertainty by plotting the confidence intervals in our prediction. we evaluate the trained model by taking 1000 samples per data point. These samples are used to approximate quantities of the posterior predictive distribution such as the mean and the $0.05, 0.95$ quantities. Intuitively, we can think of each 1 of the 1000 samples as a different neural network with slightly different parameters. Figure 3 demonstrates the epistemic uncertainty being modeled if we train the model 100, 500 and 5000 iteration respectively. As we observed, the confidence interval of the single data points' prediction is shrinking over iterations, which indicates that the model preserves great ability in capturing and describing the uncertainty in its predictions. Too few iterations will lead to poor fit of the data, while too many iterations, similar to its MLE counterpart, will make the model overfit the training data and become overconfident in its predictions (which could lead to poor generalizability to new data).

5.2 Classification on MNIST

Pyro is a universal probabilistic programming language (PPL) written in Python and supported by PyTorch on the backend. Pyro enables flexible and expressive deep probabilistic modeling, unifying the best of modern deep learning and Bayesian modeling. It helps to "automatically" convert NNs into bayesian counterparts. To examine the functionality of Pyro, we built a simple BNN model with single fully connected layers, and model all the priors with Gaussian distributions.

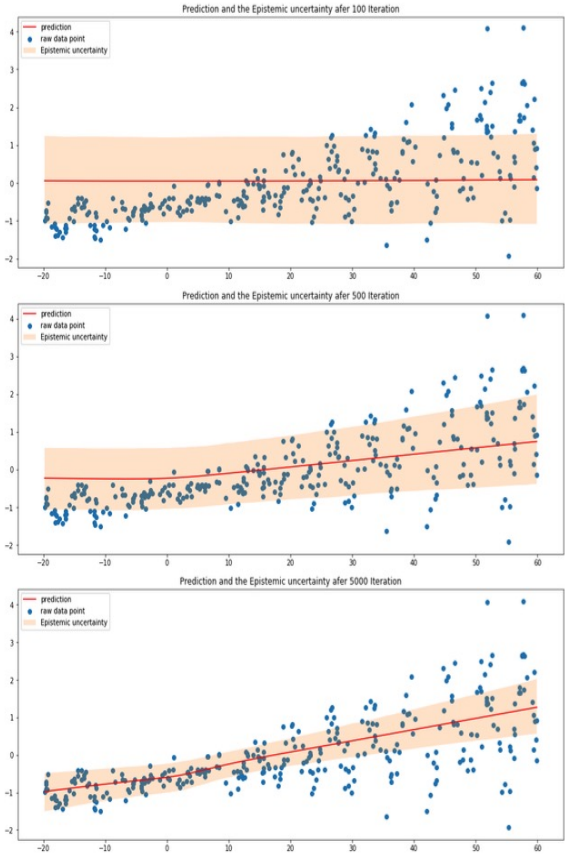

Figure 3: Epistemic Uncertainty at Different Iterations

5.2.1 Prediction Accuracy

Figure 4 demonstrates the ELBO loss over iterations, we observe that the model quickly converges. The model achieved 0.92 of accuracy in testing data, which is reasonably great given the simply network structure. To better make use of the uncertainty of the BNN model, we evaluate the trained model by taking 1000 samples per test data image. We also take the median of the prediction results per class, since if this quantities go lower than a certain threshold, we will know that the model is not confident in making prediction for this particular image. We set the threshold for this median value to be 0.2, and we retrained the model with same set up as above. We found out that the model output an 'I dont know' for 333 images with 10000 data samples, but the model is able to make predictions on all samples for 9667 images, and the prediction accuracy increased from 0.92 to 0.95. Figure 5 shows a case where the model made correct prediction and a case where the model is not able to predict.

5.2.2 Test on Noisy test data

Two more tests were ran on the model. Firstly, we randomly generated 1000 noisy $28 \cdot 28$ images to test the model. The

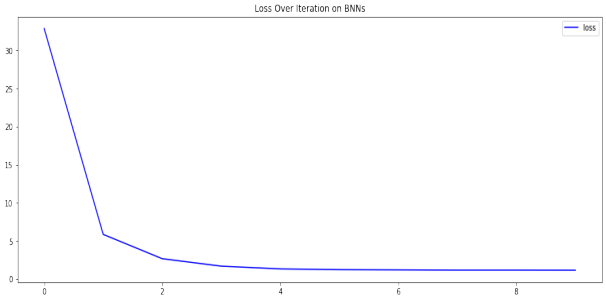

Figure 4: Loss Over Iteration

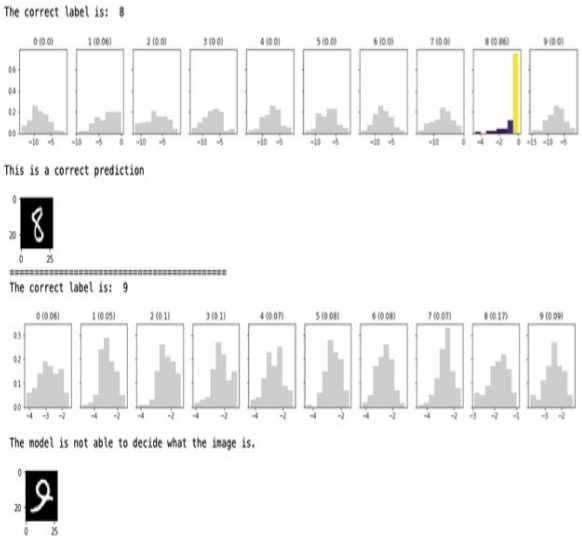

Figure 5: BNN Classification cases

model successfully rejected 998 of the test image and output 'I don't Know'.

We then extended the test on MNIST-alphabet data set. Same as above, we tested 455 images with label 'A' to 'J' on the trained model. Out of 455 test images, the model rejected 243 in predicting, which suggests that the model is still under-trained or the feature space is not able to distinguish between digits and alphabets. Therefore, more training is needed or a more complex model structured is required to improve the accuracy.

6. Conclusions and Future work

In this report, we have discussed in detail about how BNNs could be trained including the use of KL Divergence, ELBO maximization and the reparameterization tricks. We then applied BNNs to two different predictive tasks. On the regression task, we tried to compare the model performance between BNNs and MLP, and demonstrate the power of BNN to capture uncertainties when data is noisy and suffer from heteroskedasticity. And in the classification problem, we show that BNNs are capable of achieving similar performance in prediction with other NNs, and we can use the uncertainties captured to deal with unknow/unseen data, which is pretty common in practical setting.

For future explorations, we could try placing more complicated (mixed) prior on weights and biases of BNN, which may lead to significant improvement in model performance. We can also develop more complex network architecture including Convolutional BNN or BNN-GANs to deal with more sophisticated problems. Another interesting problem to research is how to better use the uncertainties to perform regularization on training of neural networks.

References

1. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research, 15(1):1929–1958, 2014.

2. C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. "Weight uncertainty in neural network." Proceedings of the 32nd ICML volume 37 of Proceedings of Machine Learning Research, pages 1613–1622. PMLR, 2015.

3. Alex Graves. "Practical variational inference for neural net- works". In Advances in Neural Information Processing Systems (NIPS), pages 2348–2356, 2011.

4. Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., and Blei, D. M.. "Edward: A library for probabilistic modeling, inference, and criticism." arXiv:1610.09787.

5. Neal, R. M. "BAYESIAN LEARNING FOR NEURAL NETWORKS." PhD thesis, University of Toronto. 1995

6. MacKay, D. "A practical Bayesian framework for backpropagation networks". Neural Computation, 4(3):448–472, 1992.

7. Mariia Vladimirova, Jakob Verbeek, Pablo Mesejo, and Julyan Arbel. "Understanding priors in Bayesian neural networks at the unit level". volume 97 of Proceedings of Machine Learning Research, pages 6458–6467, 2019.

8. Andrew C Miller, Nicholas J Foti, Alexander D'Amour, and Ryan P Adams. "Reducing reparameterization gradient variance. " arXiv preprint arXiv:1705.07880, 2015.

9. Saatchi, Y. and Wilson, A. G. "Bayesian gan". In Advances in Neural Information Processing Systems, pp. 3625–3634, 2017.