Bayesian Approach to Giving Neural Networks the Power to Reject to Classify

Hyun Jae Cho
Department of Computer Science
University of Virginia
hc2kc@virginia.edu

Abstract

Artificial neural networks have proven to be powerful at identifying patterns between the distributions of input and output data. However, if an input is not within the distribution that the model was trained on, and hence the model is unsure which class it belongs to, the model should be able to "reject" to classify it rather than guessing with low confidence. In conventional neural networks, the values of the parameters after training are fixed scalars that the model uses to classify inputs. Hence, given a fixed input, the network will always classify it identically. On the other hand, in the Bayesian context, each parameter is a random variable with an associated probability distribution. As a result, training each parameter distribution initiates with a prior distribution and updates its latent variables, the mean and the standard deviation. Once trained, the values of the parameters are sampled from the corresponding distributions, allowing the model to make different inference for an identical input. This project compares the training process and evaluates the performance difference between conventional and Bayesian neural networks. To highlight, the Bayesian neural network containing one hidden layer with the ability to reject to classify achieved 65.69% inference accuracy on the CIFAR-10 dataset [2], whereas a conventional fully connected neural network with the same depth achieved 30.84% and a convolutional neural network achieved 45.79%.

1. Introduction

In the recent decades, a significant amount of research in Bayesian statistics has developed variational inference, an algorithm that has been shown to be effective at estimating the posterior distribution given the prior distribution and data. In contrast to Markov Chain Monte Carlo (MCMC) method, which is a sampling-based approach to estimate the posterior distribution, variational inference provides a much faster estimation. Such improvement has since been widely exploited, including its application in variational autoencoders (VAEs) [1]. However, until very recently, using varitional inference to train neural networks has been more sparsely studied. Since the parameters in a Bayesian neural network are random variables with distributions, an identical input data can be classified differently due to the stochastic approach to sample the parameters. In this project, I explain how the parameter distribution estimation in a Bayesian neural network can be accomplished with variational inference and using these parameter distributions to make inference. This is mainly accomplished using the probabilistic modeling language Pyro. Then, the performance difference between conventional neural networks and Bayesian neural networks are evaluated. The model structures are kept at minimum to purely demonstrate the difference in performance, avoiding the use of pooling or batch normalization. Finally, I discuss the potential implications and limitations to Bayesian neural networks.

2. Related Work

Very recently, the idea of using Bayesian methods, such as variational inference, to train neural networks' parameters has only been a rising area of research as seen in [[3], [4], [5]], to just name a few. Notice that the mentioned works are all published on arXiv in 2019. This project serves as a recreation of the idea of using variational inference to train neural networks and producing the certainty of inference as a probability, in addition to the inference itself using Pyro.

3. Methods

In a conventional neural network, the weight and bias parameters are numerical values that get updated by backward propagation to map the input to the output. In the Bayesian setting, each of the model parameters is a random variable that has a probability distribution associated with it. Figure 1 demonstrates the difference between the two models.[1] Therefore, a Bayesian network needs to be able to estimate

[1]Image source: https://courses.csail.mit.edu/18.337/2017/projects/morales_manuel/img/

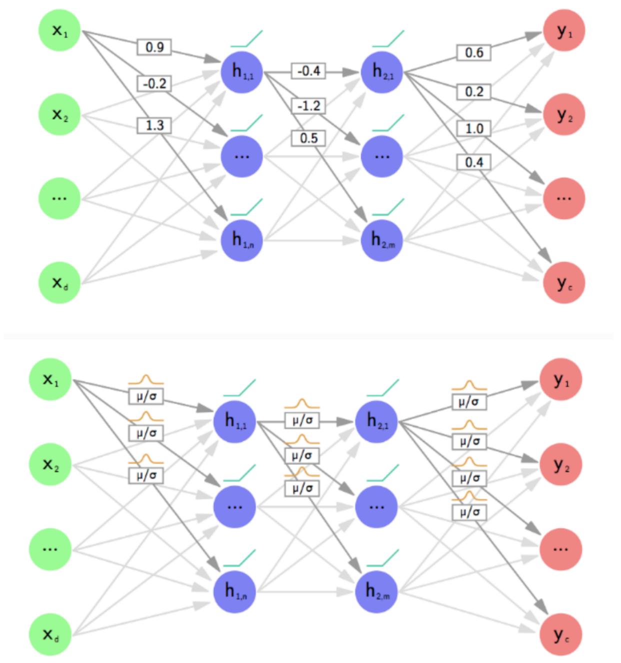


Figure 1: A comparison between a regular neural network where each parameter is a scalar value (top) and a Bayesian neural network whose parameters are represented by distributions with mean μ and the standard deviation σ.

the distributions well, since exactly calculating the true distribution for every parameter is often times computationally infeasible.

$$p(Z|\theta) = \frac{p(\theta|Z) \cdot p(Z)}{\int p(\theta, Z)\, dZ} \tag{1}$$

In Bayes' Theorem (1), where Z and θ respectively represent the latent variables of distributions and the parameters, the denominator is computationally infeasible to exactly calculate for several reasons. For example, it is computationally infeasible to sample every combination of parameters, where the number of parameters is very large, which is a common feature in neural networks. Furthermore, if the prior distributions are initialized with a standard Gaussian distribution $N(\mu = 0,\, \sigma^2 = 1)$, which is the case in this project, there are infinite number of possible samples for each parameter. In this section, I explain how estimating the posterior $p(Z|\theta)$ can be done using variational inference.

3.1. Learning using Variational Inference

This section illustrates the criterion for learning in a Bayesian neural network using variational inference. The objective is to estimate the posterior probability of the latent variables given observed parameter samples. This is because the parameters are represented by the latent variables. However, due to the aforementioned reasons, $\int p(\theta, Z)\, dz$ is often computationally intractable; hence, rather than directly trying to calculate the integral, we need to estimate it using $q(Z)$, where $q(\cdot)$ is a function whose distribution parameters are easy to calculate. Specifically, variational Bayes algorithm establishes this estimation step using Kullback-Leibler (KL) Divergence:

$$q^*(Z) = argmin_{q \in Q}\, KL(q(Z)\,||\,p(Z|\theta))$$

where Q is a family of potential functions. Expanding the KL-divergence term,

$$\begin{aligned} KL(q(Z)\,||\,p(Z|\theta)) &= \int q(Z) log(\frac{q(Z)}{p(Z|\theta)})\, dZ \\ &= \int q(Z) log(\frac{q(Z)\, p(\theta)}{p(Z,\theta)})\, dZ \\ &= log\, p(\theta) - \int q(Z) log(\frac{p(Z,\theta)}{q(Z)})\, dZ \end{aligned}$$

Since the objective is parametrized in terms of q, $log\, p(\theta)$ has no affect in the result. Therefore, maximizing $\int q(Z) log(\frac{p(Z,\theta)}{q(z)})\, dZ$ will minimize the divergence. This term is called the Evidence of Lower Bound (ELBO), and every term in ELBO can be calculated in closed form.

3.2. Certainty of Inference

The process for calculating the probability that a classification output is correct is as follows. For every input image, multiple forward passes are executed, say N times, sampling the parameters independently each time. The probability of classifying an image is determined by taking the median probability among the N forward passes. For example, for N=5, and an image was classified as class c with probabilities 15%, 18%, 20%, 26%, 30%, then the probability of classification being correct was set to be 20%. In this project, N was set to 30.

3.3. Implementation Details

This project used PyTorch to implement, train, and test the neural networks. During preprocessing, images were normalized, and no further preprocessing steps were applied. Throughout the models, the following details were followed: batch size=100, number of hidden neurons in hidden layers=512, learning rate=0.005, optimizer=Adam, number of epochs=5. The loss function for non-Bayesian neural networks was cross entropy loss, and the Bayesian neural network used the ELBO loss function provided by Pyro. In order to test the pure performance between the models, no pooling or batch normalization was used. Training was done on Google Colab, which uses Nvidia Tesla K80 GPU.

4. Validation of Performance

In Table 1, I present the differences in the performance between the Bayesian neural network and two conventional types of neural networks. The Bayesian model is decomposed into two separate models, one that can reject to classify and one that cannot. The non-Bayesian models are a fully connected network with one hidden layer and a convolutional neural network with a convolutional layer combined with a fully connected classification layer.

Because convolutional neural networks extract the features, which are inherent in image data, they tend to perform better than fully connected neural networks. This is verified by the difference between their performance 45.79% and 30.84% found in Table 1. The Bayesian neural network that cannot reject to classify performed between the two, achieving 37.77% accuracy. Lastly, the Bayesian neural network that can reject to classify inputs that it is unsure of achieved a significantly better performance compared to every other model, including the convolutional neural network. Because the images that the model decided to classify were easier for the model to classify, its accuracy was more than twice higher than that of fully connected neural network.

model	**validation accuracy (%)**
FC-layered network	30.84
Convolutional neural network	45.79
Bayesian neural network that can't reject	**37.77**
Bayesian neural network that can reject	**64.69**

Table 1: Allowing the network to classify only when it is more certain than a threshold value (20%) resulted in a much higher accuracy than its direct counterpart (30.84%) and even a convolutional neural network with the same network depth (45.79%). These models are implemented without techniques to improve the accuracy beyond the layers themselves; no batch normalization or pooling was applied. ReLu was used for nonlinearity.

Figure 2 illustrates that the Bayesian neural network learned to classify certain categories of images better than others. Since parameters are sampled, this distribution may have been different if different parameters were sampled. In comparison, Figures 3 and 4 are the distributions of images whose softmax value was below 20% in the baseline neural networks. Although softmax is a scaling function rather than a probability function, this demonstrates the different performance of the networks on images with different labels. Unlike the rejection distribution of the Bayesian model, the rejection distributions of the fully connected and convolutional networks do not change because the parameters are fixed scalar values after training.

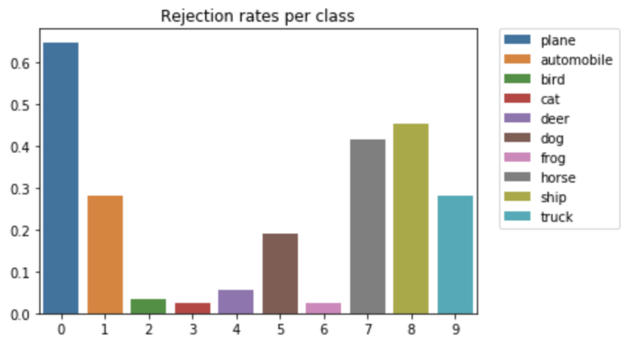


Figure 2: This distribution of rejected images shows that the Bayesian model with those parameters sampled to produce the output was more confident about bird, cat, and frog images than it was for plane, horse, and ship images.

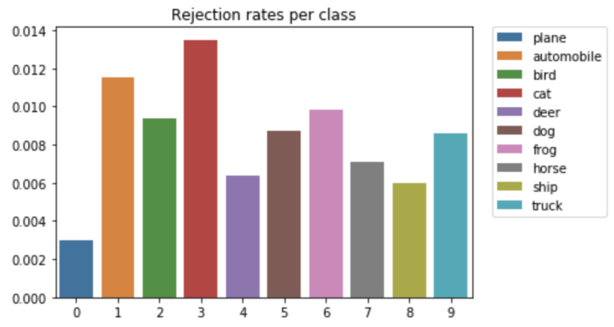


Figure 3: The distribution of images whose softmax value was below 20% during the inference time of the baseline fully connected model.

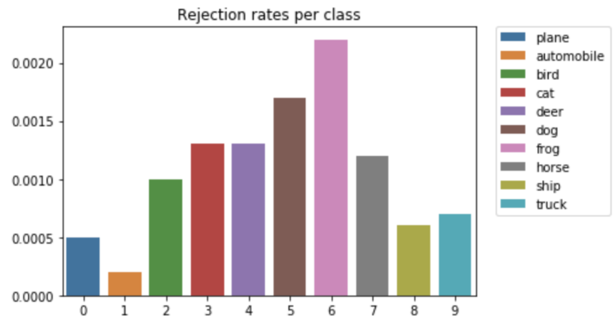


Figure 4: The distribution of images whose softmax value was below 20% during the inference time of the baseline convolutional neural network.

Finally, at the end of the paper, I provide examples of the images that the model classified and rejected. The model was more than 20% confident about the correct label for the images (a) through (f). On the other hand, it rejected to classify the images (g) through (l) due to uncertainty. The latter images indeed seem to be more ambiguous than the former.

5. Discussion

The implications for bayesian neural networks is diverse and potentially impactful. For instance, in self driving cars, camera images are processed to detect objects, such as vehicles, pedestrians, and traffic signs. In today's leading self driving companies, object detection algorithms such as Yolo, RCNN, and SSD are widely used to detect objects; however, the important cases arise when an image contains an object that is rarely contained in the training data or not contained at all. In such a case, the car should warn the driver and let him take over the control, without trying to decide what to do.

model	**Time took for inference (seconds)**
FC-layered network	4.78
Convolutional neural network	6.87
Bayesian neural network that can't reject	**396**
Bayesian neural network that can reject	**484**

Table 2: Note the significant amount of time required to run the Bayesian models compared to the non-Bayesian models. For the Bayesian networks, the number of forward passes ran to calculate the probability of making a correct prediction was 30.

There are a few limitations that Bayesian model suffered from. The first drawback was that the nature of the structure of Bayesian neural networks requires running the forward pass multiple times (N = 30 in this project) to determine the confidence probability. Referring to Table 2, when the model had to make inference for every image, the time took was 160 seconds, and when the model was given the ability to reject to classify, the time increased dramatically to 484 seconds. This difference can be ascribed to the additional computations required for the model that can reject to classify such as checking the confidence probability and calculating the median probability. On the other hand, the non-Bayesian models were much faster at making inference. This is a critical drawback (and perhaps the reason that Bayesian networks lose practical application) because in many cases, a fast, real-time analysis is required. For example, the rate at which images are classified is crucial in self driving cars because multiple control actions must be taken by the model every second.

Secondly, when the model was able to reject to classify, it did not classify around 85% of the images. In other words, the model was 20% or more certain for only 15% of the test images. Specifically, the model skipped 8499/10,000 images. This drawback illustrates another limitation of the Bayesian model.

Aside from drawbacks, I conclude by noting the differences between when N = 10 and N = 30. When N = 10, the Bayesian neural network that cannot reject took 160 seconds and achieved 35% accuracy, while the Bayesian neural network that can reject took 251 seconds, achieved 57% accuracy and skipped 8293/10,000 (83%) images. It is interesting to note that there is not much of accuracy difference between N = 10 and N = 30 when the model could not reject to infer (37.77% vs 35%), but a large difference when the model was able to reject (64.69% vs 57%). The ratio of skipped images was similar (85% vs 83%).

6. Future Research Directions

This project has proven the robustness of Bayesian neural networks compared to conventional neural networks when handling uncertainty. A natural next step would be checking if it adds any robustness benefit to conventional neural networks against adversarial input images. I presume the model is insufficiently robust against such attacks, but it would be an interesting research direction to add features to Bayesian networks to make it robust. In addition, to overcome the speed limitation of Bayesian networks, research in calculating the certainty measurement without having to run the forward pass multiple times will increase the practicality of the Bayesian networks. On possible method is utilizing the variance of the parameters, not the variance of the outputs of multiple forward passes.

7. Conclusion

In this report, I explained the idea of applying variational inference in a neural network so that the network only classifies an input image when the certainty is above a threshold. With this feature, the Bayesian model performed significantly better than other models with comparable model structure complexity. However, it should also be taken into consideration that this improvement incurred rejecting many images as well as the dramatic increase of the time complexity because multiple forward passes were required to determine the probability of correct inference. The code for this project is available at my github:

https://github.com/hyunjaecho94/Vision-course-project.

References

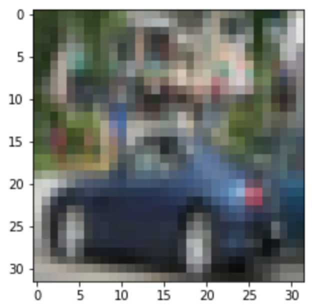
[1] D. Kingma and M. Welling. Auto-encoding variational bayes. In *arXiv preprint*, page arXiv:1312.6114., 2013.

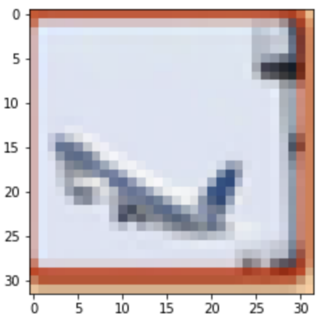
[2] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

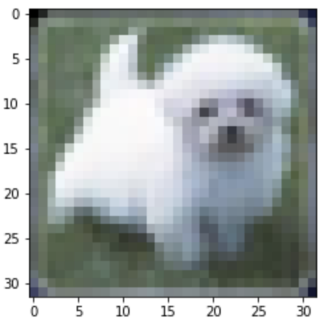
[3] Y. Li and Y. Zhu. Performance measurement for deep bayesian neural network. In *arXiv preprint*, page arXiv:1903.08674., 2019.

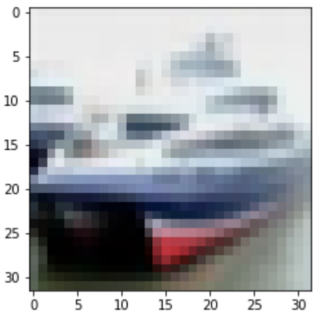
[4] K. Shridhar, F. Laumann, and M. Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. In *arXiv preprint*, page arXiv:1901.02731., 2019.

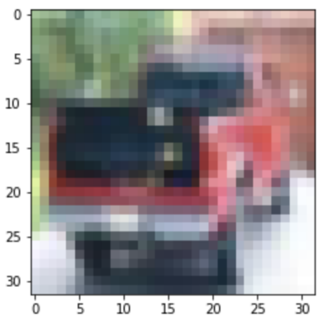
[5] S. Sun, G. Zhang, J. Shi, and R. Grosse. Functional variational bayesian neural networks. In *arXiv preprint*, page arXiv:1903.05779., 2019.

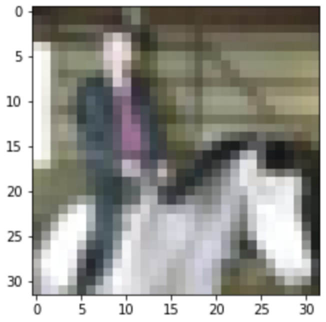

(a) automobile

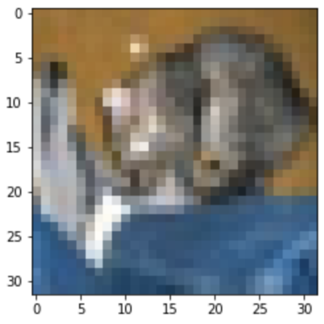

(b) plane

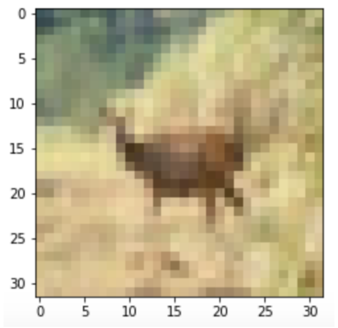

(c) dog

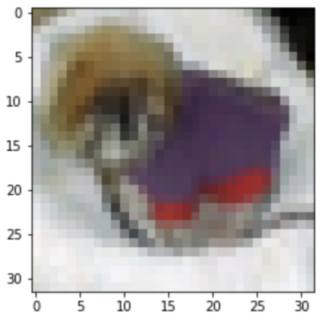

(d) boat

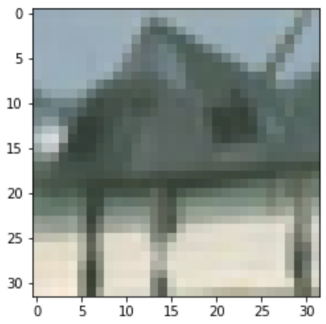

(e) truck

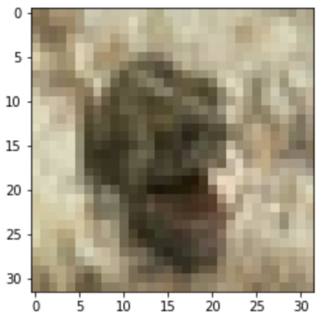

(f) horse

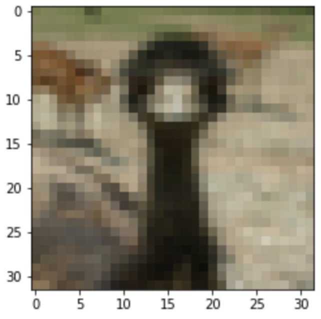

(g) cat


(h) deer


(i) dog


(j) plane


(k) frog


(l) bird