Deep Learning Project: Dropout Bayesian Neural Networks for Prediction Uncertainty Estimation

Tianyu Cui, Yi Zhao

April 3, 2020

1 Problem and Data Description

1.1 Problem Description

Deep learning methods have become the state of art tool in both regression problems and classification problems. However, these approaches cannot capture the uncertainty of the model, i.e, these models directly return one prediction or classification result without offering any useful information to decide the reliability of this result. In many cases, this uncertainty is extremely important. For example, in cancer diagnosis, facing a special case that the model never seen before, a desirable model should return the diagnostic result with large uncertainty, which means the model should know what it does not know. To capture the uncertainty, the Bayesian Neural Networks (BNN) offers a recipe, which is introduced in the methodology part. By adopting BNN, the model can return prediction or classification results as well as the uncertainty. However, there also arise another problem that needed to consider: the calibration of the output uncertainty, which describes the difference between the predictive uncertainty we obtained and the true uncertainty. In other words, the well-calibrated is that the predictive uncertainty is closed to the true uncertainty, and the more detailed criterion is introduced in the experiment part. In a nutshell, this project introduces two models that capture the uncertainty of the model and then evaluates its calibration.

1.2 Data Description

In our project, we use the "California Housing Prices" as our dataset. The dataset contains 20360 sets of data from the 1990 California census. In one set of the data, it includes 10 element, which are the longitude, latitude, housingMedianAge, totalRooms, totalBedrooms, population, households, medianIncome, medianHouseValue and oceanProximity. All the data is double floating point, except the "oceanProximity", which is an object.

In our case, we serve the "medianHouseValue" as the lable, and we use 80% data points (16346 elements) as our training set, and the remaining 20% (4014 elements) as the test set. Then, in order to train the model more feasible, we drop the "oceanProximity" as it is an object and not offer much information and also scale the data from its original range into range 0.5-1.5.

2 Methodology

2.1 A Brief Review of Bayesian Neural Network

Bayesian Neural Networks (BNN) [MacKay, 1992, Neal, 2012] have been studied since the 90s, but they were not generally applicable to large datasets, because the inference algorithms such as mean-field VI are computational hungry even for small datasets. After scalable variational inference algorithms [Hoffman et al., 2013, Kingma and Welling, 2013, Ranganath et al., 2014] were proposed, BNN became much popular in many applications. More importantly, Gal and Ghahramani [2015] show that the standard dropout training in NNs can be interpreted as a form of variational inference in BNNs, and that the prediction uncertainty can be obtained by simply using dropout during testing. This approach can be applied to most neural network architectures, such as RNNs [Gal and Ghahramani, 2016], as long as we can train the NN with dropout. Based on this technique, many critical problems that require uncertainty estimation of predictions have been well studied, such as self-driving car [McAllister et al., 2017, Michelmore et al., 2018] and medical analysis [Gal et al., 2017].

In the original paper of Gal and Ghahramani [2015], the dropout probability of each layer is fine-tuned by using grid search, which is cumbersome. More importantly, pre-specified dropout probabilities can heavily influence the estimated prediction uncertainty, so how to choose the dropout probability is critical in uncertainty estimation problems. Meanwhile, Kingma et al. [2015] propose *variational dropout* to **learn** the variance of Gaussian dropout probability directly from data, which avoid heuristic searching. In this project, we implemented these two Bayesian Neural Network, and compared their uncertainty estimation on a real-world dataset.

2.2 Dropout Bayesian Neural Network

Bayesian neural networks are defined by placing a prior distribution on the weights of a neural network. Then, instead of finding a **point estimate** of the weights by minimizing a cost function in classical NNs, a **posterior distribution** of the weights is calculated conditionally on the data. Let $g^{\mathbf{w}}(\mathbf{x})$ denote the output of a BNN and $p(\mathbf{y}|g^{\mathbf{w}}(\mathbf{x}))$ the likelihood. Then, given a dataset $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$, $\mathbf{Y} = \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(N)}\}$, training a BNN is equivalent to learning the posterior distribution $p(\mathbf{W}|\mathbf{X}, \mathbf{Y})$.

One popular way to find $p(\mathbf{W}|\mathbf{X}, \mathbf{Y})$ is using variational inference (learned from course Machine Learning: Advanced Probabilistic Methods). We approximate the intractable $p(\mathbf{W}|\mathbf{X}, \mathbf{Y})$ with a simpler distribution, $q_\theta(\mathbf{W})$, by minimizing the $\mathrm{KL}(q_\theta(\mathbf{W})||p(\mathbf{W}|\mathbf{X}, \mathbf{Y}))$. This is equivalent to minimizing the negative ELBO

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \int q_\theta(\mathbf{W}) \log p(\mathbf{y}^{(i)}|g^{\mathbf{w}}(\mathbf{x}^{(i)}))\mathrm{d}\mathbf{W} + \mathrm{KL}(q_\theta(\mathbf{W})||p(\mathbf{W}))). \tag{1}$$

which still looks very difficult to solve. We will introduce two algorithms that related to dropout, which can optimize Eq.1 easily.

2.2.1 Algorithm 1: MC Dropout

The first algorithm that we used is called *MC dropout*, which was first proposed by Gal and Ghahramani [2015]. Gal and Ghahramani [2015] shows that implementing variance inference with a specific class of distributions $q_\theta(\mathbf{W})$ in Eq.1 is equivalent to implement dropout training that we use very often in neural networks. He shows that dropout training with L_2 regularization:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{y}^{(n)} - f(\mathbf{x}^{(n)}, \mathbf{W}, \mathbf{z})))^2 + \lambda \sum_{i,j}^{L}(||w_{i,j}||^2) \tag{2}$$

is equivalent to optimize Eq.1 with $q_{\mathbf{M}}(\mathbf{W}) = \prod_{l=1}^{L} \prod_{k=1}^{K_l} \mathbf{m}_{l,k}\text{Bern}(1-p_l)$, where p_l is the pre-specified dropout probability for layer l, and $\mathbf{m}_{l,k}$ is a vector of outgoing weights from node k in layer l, and $p(\mathbf{W}) \sim N(0, \mathbf{I})$.

Based on this equivalence, the easiest way to obtain prediction uncertainty is training a neural network with dropout, but with one small difference: In a classical neural network, during training, we turn on dropout, but in prediction stage, we **turn off** dropout. But In Bayesian neural network setting: we turn on dropout during training, and when prediction, we still **turn on** dropout. Then we for each input $\mathbf{x}^n$ we forward pass several times and calculate the sample mean and sample variance (uncertainty estimation).

2.2.2 Algorithm 2: Variational Dropout

We first introduce Gaussian dropout. We multiply a Gaussian noise $\xi \sim N(1, \alpha_l)$ to all the nodes in layer l instead of multiply a Bernoulli mask $z \sim \text{Bernoulli}(1-p_l)$. Some works [Srivastava et al., 2014] show that they are equivalent by setting $\alpha_l = \frac{1-p_l}{p_l}$. In *Variational Dropout*, we use Gaussian dropout instead of Bernoulli dropout, and we want to learn the variance α_l from data directly.

Now we only consider the **first term** in Eq.1. We optimize the first term directly by using SGVB (or reparametrization trick). First, we sample $\mathbf{W}^{(i)} \sim q_\alpha(\mathbf{W})$, and then the first term in Eq.1 becomes:

$$\mathcal{L}_1(\mathbf{W}) = -\frac{N}{M} \sum_{i=1}^{M} \log p(\mathbf{y}^{(i)} | g^{\mathbf{w}^{(i)}}(\mathbf{x}^{(i)})), \mathbf{W}^{(i)} \sim q_\alpha(\mathbf{W}) \tag{3}$$

where $q_\alpha(\mathbf{W})$ is a Gaussian distribution with mean equals to neural network parameter W and variance equals to α_l. Eq.3 is an unbiased differentiable minibatch-based Monte Carlo estimator, and can be optimized directly.

The second term in Eq.1 can be approximated by following equation:

$$\mathcal{L}_2(\alpha) \approx 0.5 \log(\alpha) + 1.1615\alpha - 1.50\alpha^2 + 0.586\alpha^3 \tag{4}$$

We optimize Eq.4 and Eq.3 together to obtain the most suitable dropout variance from data.

After we learn dropout variance, we apply the same strategy that we used in MC dropout: For each input $\mathbf{x}^n$ we forward pass several times with learned dropout and calculate the sample mean and sample variance (uncertainty estimation).

3 Experiments and Results

In this section, we will offer more detail information about our models. One thing should be noticed is that we adopt a relatively simple deep learning model for MC dropout and Variational dropout model, as our main points in this project are to capture the uncertainty of the model and its calibration rather than the fancy model. In the following subsections, we will show the detail.

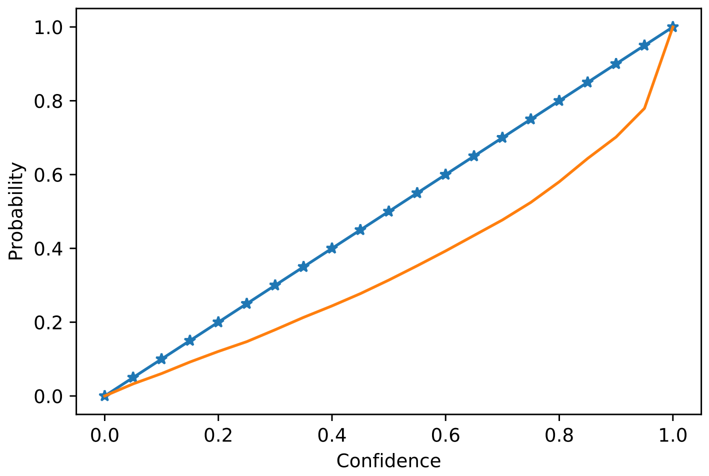
3.1 Experiment 1: MC Dropout

For MC Dropout, we train four layers, full-connected model. In our model, we first add a Dropout layer with dropout probability. Then pass through a Linear layer with 100 hidden neurons and also follows a *ReLU()* function. After these, we add the second and third layers in the same way, but adopt a dropout layer with 0.15 and 0.25 dropout probability. Then we adopt a Dropout layer with the dropout probability 0.3 before the output layer. In this model, the dropout probability is hand-tuned to get good prediction results and calibrated uncertainty. We can notice that with the increase of the layer, the dropout probability is also increased. This makes sense as the model try to keep more information about the data at the first beginning, and then drop information in deep layers to avoid the overfitting. In this model, we directly use the MSE loss as our criterion, and adopt Adam optimizer.For the hyper-parameter, we set the learning rate as 0.001, the weight regularization as $\frac{(1e-4)^2}{\#data}$ and the dropout probability regularization as $\frac{2.0}{\#data}$. And we also set the number of the epoch as 1000, the batch size as 50.

To obtain the uncertainty of the output. In the test section, different from the normal test procedure which will turn off the dropout mechanism, the dropout is still opened. We test the trained model in 20 times, therefore, 100 sets of outputs are obtained. Then we calculate the sample mean and sample variance as the approximation of the output distribution. And the sample mean is the prediction result, and the sample variance represents the uncertainty.

In order to evaluate the calibration of this uncertainty, we calculate the confidence interval of each test points under a series of different confidence, and then calculate the probability of the true value that is in this confidence interval. For a well calibrate uncertainty, this probability should close to the corresponding confidence. For example, under 95% confidence, we anticipate there exist 95% test labels that are in the interval.

And the resulting plot is shown as follow. The horizontal axis is the confidence, and the vertical axis is the probability that the probability of the true values exist in the corresponding confidence interval.

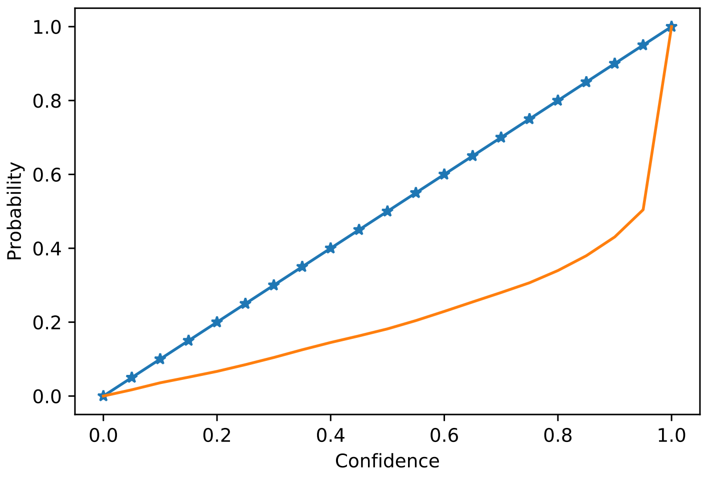


3.2 Experiment 2: Variational Dropout

In the Variational Dropout, we almost follow the same procedure to capture the uncertainty and evaluate the calibration of its uncertainty, which also first train a model, then sample from its result to obtain the sample mean and the sample variance. After obtaining the variance, we adopt exactly the same method and plot to evaluate its calibration. The difference between the two methods is that in the Variational Dropout, instead of tuning the dropout probability by hand, the dropout probability can be learned from data. As an extension of the original paper, which set a dropout probability for each layer, we set an individual dropout probability for each node, which will offer more flexibility for the model.

In the experiment of the Variational Dropout, instead of using the standard packaged dropout layer in the PyTorch, we first define the variational dropout layer, which has been introduced in the methodology part. Then, we adopt the same model structure, and using our self-defined variational layer to replace the standard dropout layer of the PyTorch. Notice that the layer and the number of the hidden node of each layer is defined in parameter *layers*. For the hyper-parameter, the learning rate is 0.001, the tolerance and patience for early stop are 0.01 and 20. And we also the batch size is 50, the number of the epoch is 1000.

In the prediction period, we sample 100 times from the result to calculate its mean and sample variance, which is the same with the MC dropout method, and also obtain the same plot to represent the calibration of the uncertainty.



To compare the performance of two methods, except directly comparing the calibration plot of two uncertainty, we also introduce another criterion – the integral area between our results curve and the desired curve. According to our results, the integral area of the variational dropout is 0.280 and the integral area of the MC dropout is 0.147. Also, from two plot, we find that both variational dropout and MC dropout tend to underestimate the uncertainty.

4 Conclusion

In this project, we introduced two methods to capture the uncertainty of the prediction – the MC Dropout and the Variational Dropout. The MC Dropout is easy to implement and can capture the uncertainty in an efficient way. However, in the MC Dropout, the dropout probability is set for each layer and serves as a hyper-parameter that needed to be tuned by hand or using the grid-search method, which will be a waste of computational resources. In order to tune this hyper-parameter more efficient, we introduced the Variational Dropout, which can find the optimal dropout probability using the gradient method. Besides, in order to make the model more flexible, we extended the model used in the original paper. Instead of setting a dropout probability for each layer, we set the dropout probability for each node.

After obtaining the uncertainty successfully, we also evaluated the calibration of the uncertainty by calculating the probability that the true label values are inside the confidence interval. We also adopted the integral area between the probability curve we obtained from experiments and the desired curve to describe the overall uncertainty of the prediction.

From the calibration plot, we find that both MC Dropout and Variational Dropout are underestimating the prediction uncertainty. According to the original paper, the calibration of these two methods should near each other in most cases, however, we find that the Variational Dropout gets a relatively poor calibration compared with MC Dropout. This is because the drop probability of the MC Dropout is well tuned by hand. And for Variational Dropout, this dropout probability is learned from data, the lack of data and the complexity of the model will impact the prediction uncertainty. Also, variational Dropout adopts an approximated loss function, which can introduce poor prediction performance. And this method also assumes the prior of the dropout probability follows the Gaussian distribution, which is improper. Furthermore, there exist other methods can solve this problem, such as Concrete Dropout, which can also learn the dropout probability from data, but assumes a more proper prior of the dropout probability that follows Bernoulli distribution.

5 Code

The code is available in this link: `https://github.com/Yizhao111/Deep_Learning`. If there exists any problems to access this link, please feel free to contact `yi.zhao@aalto.fi`

References

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. arxiv, 2015.

Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3): 448–472, 1992.

R. McAllister, Y. Gal, A. Kendall, M. Van Der Wilk, A. Shah, R. Cipolla, and A. V. Weller. Concrete problems for autonomous vehicle safety: advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017.

R. Michelmore, M. Kwiatkowska, and Y. Gal. Evaluating uncertainty quantification in end-to-end autonomous driving control. *arXiv preprint arXiv:1811.06817*, 2018.

R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.

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.