Baltic J. Modern Computing, Vol. 0 (2012), No. 0, pp. 00–00

Comparative analysis of Robustness of Bayes Neural Nets and Monte Carlo Dropout methods

Rinalds Daniels Pikše[1], Evalds Urtans[1]

Riga Technical University, Department of Artificial Intelligence and Systems Engineering, Riga, Latvia

rinalds.pikse@gmail.lv, evalds.urtans@rtu.lv

Abstract. This is a survey paper that compares three Beyesian methods - Bayes by Backprop, Variational Inference, and Monte Carlo Dropout. These methods measure the uncertainty and robustness of the data. The research aims to identify the strengths and weaknesses of each method in order to guide practitioners in selecting the most suitable method for various applications.

The paper reviews the theoretical foundations of each algorithm and describes the experimental setup, which involves mixing labels for the mushroom classification data set from 0 to 50% of the training data. The performance of the method is evaluated on the basis of its uncertainty estimates, accuracy, and robustness.

In noise-free conditions, the Monte Carlo Dropout algorithm delivered 100% accuracy, while Bayes by Backprop also performed impressively at 99.7%. However, with 40% data noise, Bayes by Backprop proved the most robust, its accuracy decreasing to 92.93%, compared to Monte Carlo Dropout's more significant drop to 90.67%. Variational inference showed the highest susceptibility to noise, dropping to 76.6%. At the same time, Variational Inference and Monte Carlo Dropout were shown to be a potential estimator of the noiseiness of the data set, since its standard deviation correlated with noise $r = 0.95$.

Keywords: Machine Learning, Bayesian Neural Networks, Monte Carlo Dropout, Bayesian Inference, Bayes by Backprop, MC-D, VI, BBB

1 Introduction

Bayesian neural networks represent a specific type of neural networks that express a measure of uncertainty about their predictions, providing not only point predictions, but also a probability distribution over possible outcomes. They work by applying principles from Bayesian statistics to estimate the uncertainty in the network's weights, hence providing a mathematical framework for representing and calculating uncertainties.

The concept of Bayesian neural networks has seen significant evolution since its inception in the 1990s. Renowned methods have been proposed over the years such as

Variational Inference (VI), introduced by Jordan et al. (1999), which provides a method for approximating complex probability distributions. Subsequently, the Monte Carlo Dropout (MC-D) technique was proposed, which used dropout layers in a network as a means of implementing Bayesian inference Gal and Ghahramani (2015). Another significant contribution was the development of Bayes by backpropagation (BBB) by Blundell et al. (2015), a method that applied Bayesian inference to the backpropagation algorithm, one of the core algorithms for training neural networks. Further advances have been continuously researched (Jospin et al. (2020)), expanding the scope and applicability of Bayesian neural networks.

The goal of this research is to develop and apply a novel approach to comparatively assess these existing Bayesian neural network algorithms, focusing specifically on methods such as Variational Inference, Monte Carlo Dropout, and Bayes by Backpropagation. The methodology involves implementing these methods in the Python programming language and then selecting tabular data for empirical analysis.

2 Related work

Bayesian methods strive to employ probability to quantify uncertainty in conclusions based on data analysis. Probability claims made by Bayesian methods are contingent on observed data, differentiating them from other techniques (Gelman et al. (2013)). Bayesian inference, a statistical method grounded in Bayes' theorem, aims to determine the probability of a possible event, considering both prior information and newly acquired observational data.

Ai et al. (2021) describes how Bayesian neural networks (BNNs), grounded in Bayesian inference, utilize an initial probability distribution and aim to approximate the true probability distribution. Two main approximating strategies have been proposed: sampling and optimization methods. Sampling methods repetitively execute an algorithm with variable results for the input parameters, forming a distribution, whereas optimization methods implement a distribution into the function parameters, training them to obtain the final distribution. Optimization techniques include the Bayesian variational inference method, which shapes and refines value distribution (Blei et al. (2016)), and Bayes by Backpropagation, which applies the backpropagation algorithm to Bayesian inference (Jospin et al. (2020)). Monte Carlo Dropout, a sampling method, repetitively generates variable results, thus creating a value distribution. This article provides a comprehensive analysis of these methodologies.

2.1 Variational Inference

In probabilistic models, inference refers to the process of estimating latent variables given observed data and the model's parameters. Latent variables, often referred to as hidden variables, are indirectly obtained from observable variables and influence the final result. The goal of variational inference is to create an approximate final posterior probability from observed and latent variables (Blei et al. (2016)).

Variational inference is a technique in Bayesian statistics in which an approximate distribution (the variational distribution) is used to represent the posterior distribution

of the model parameters. The parameters of this variational distribution are optimized so that the distance between the variational distribution and the true posterior distribution is minimized, often using measures such as the Kullback-Leibler (KL) divergence, which measures the difference between two probability distributions based on Shannon information theory (Jospin et al. (2020)).

However, minimizing KL divergence is not easily applicable, as machine learning algorithms do not always have access to the true value distribution. Hence, using Jensen's inequality, it is possible to derive a new optimizable measure called ELBO (Evidence Lower BOund) (Blei et al. (2016)). Although minimizing the KL divergence is equivalent to maximizing the ELBO, the advantage of the latter approach is that it does not require knowledge of the true value distribution for optimization (Jospin et al. (2020)).

Several methods can be used to maximize the ELBO value. One possible approach is the gradient descent method, which adjusts the weight parameters of the variational distribution to maximize the ELBO value. Stochastic Variational Inference (SVI) uses batches of samples that form probability distributions, which are iteratively improved using stochastic optimization (Hoffman et al. (2012)). This approach can be scalable, since ELBO can be calculated for each batch of samples, in each training iteration (Jospin et al. (2020)). ADVI is another method that seeks to maximize the ELBO value by transforming the inference problem into a uniform space and then solving the variational optimization problem. As described (Kucukelbir et al. (2015)), there are also so-called black-box variational inference methods (Ranganath et al. (2013)), which allow maximizing ELBO, but often rely on stochastic optimization (Kucukelbir et al. (2015)), hence they can be considered a subgroup of SVI algorithms.

Other methods also exist for Bayesian Variational Inference, which are briefly described in this section. Structured Mean Field Variational Inference (SMFVI) is a method that maintains dependencies between variables (Hoffman and Blei (2014)). This method is practical when variable relationships are known. Nonparametric models perform variational inference by creating a more complex value distribution that can contain several modes (Nguyen and Bonilla (2013)), which is illustrated in Figure 1.2. This figure provides an example of the multimodal problem, where a single-mode probability distribution would not fully characterize the true probability.

2.2 Monte Carlo dropout

Monte Carlo Dropout (MC-D) is a method involving the stochastic deactivation of a proportion of neurons in a network. MC-D is widely adopted in traditional neural networks to avoid overfitting by reducing the influence of each individual neuron on the entire network. After model training, dropout is deactivated (Srivastava et al. (2014)).

This technique, where neurons are deactivated, can also be used in a different context - to calculate the uncertainty measure of a neural network. Gal and Ghahramani (Gal and Ghahramani (2015)) demonstrate that, by not deactivating dropout after training, this method can be considered an approximation of Bayesian methods and yield a distribution of values equivalent to the Variational Inference algorithm. Monte Carlo methods, through random neuron deactivation (dropout), allow the creation of a dynamic model that provides different results each time a specific data point is analyzed.

This occurs because deactivated neurons are removed from calculations, thereby introducing variability in the model's output upon repeated evaluations.

MC-D is easily incorporated into existing artificial neural networks without the need for model retraining (Jospin et al. (2020)), making it a popular and practical choice for injecting Bayesian principles into established deep learning models. As a dropout-based approach, MC-D shares core functionalities with traditional artificial neural networks, including a layered neuron structure with adjustable weight and bias factors, the propagation of input data to deeper network layers, and iterative optimization of weight values via error backpropagation and optimization algorithms. Despite this, the uniqueness of MC-D within the Bayesian framework arises from its ability to reflect uncertainty via multiple model evaluations, each one influenced by a distinct dropout-induced network configuration.

2.3 Bayes by Backprop

Bayes by Backpropagation (BBB), an algorithm introduced in the research work "Weight Uncertainty in Neural Networks" (Blundell et al. (2015)), incorporates principles of variational inference in neural networks. Essentially, BBB performs an evaluation of neural network weights by employing distributions instead of fixed values. The goal of this approach is to introduce uncertainty into weight values, assessing the confidence in the model's prediction accuracy. Training is executed by maximizing the Variational Lower Bound (ELBO) cost function, using both the error function and Kullback-Leibler divergence for optimization. The authors of BBB argue that, based on their studies, this algorithm's classifying performance is comparable to the Monte Carlo dropout algorithm.

The BBB algorithm uses the reparametrization trick method to assess and optimize the weight distribution parameters in the neural network. Variable reparametrization has long been a technique used in the statistical literature, but only recently has it found applications in gradient-based machine learning (Kingma and Welling (2013)). This method proves useful when learning parameters that define the distribution of values, such as the distribution of weights in a neural network.

Within the Bayes by Backpropagation (BBB) algorithm, the weight distribution of the neural network is represented as a normal distribution with adjustable parameters μ and σ. During backpropagation, neural networks employ gradient descent, which necessitates the computation of gradients of the network error function relative to the weights of the network. These gradients guide the adjustment of the network's weights to minimize the error.

In the context of BBB, it becomes important to discern not only how the error fluctuates with respect to the weights of the network but also in relation to the parameters of the weight distribution. This presents a challenge as computing gradients for stochastic values can be computationally complex and potentially lead to high variance due to inherent randomness. The reparameterization trick solves this issue - it reformulates the problem in a way that enables the calculation of gradients with respect to a deterministic weight distribution instead of a stochastic one, thus simplifying the gradient computation process.

Instead of attempting to calculate gradients in relation to the parameters of the stochastic weight distribution, the problem can be transformed to calculate gradients in relation to a deterministic weight distribution. This is accomplished by transforming samples from a standard normal distribution using our weight distribution parameters. This method is implemented by first obtaining a sample ϵ from a standard normal distribution $N(0, I)$, which is then transformed using our weight distribution parameters μ and σ and can be represented as

$$g_{\varphi}(\epsilon, x) = \mu + \sigma * \epsilon \tag{1}$$

In this equation, $g_{\varphi}(\epsilon, x)$ is the vector value function, a sample of our weight distribution, and as this transformation function is deterministic and differentiable, the calculation of gradients is straightforward using the standard backpropagation method. BBB could also be referred to as a practical implementation of the Stochastic Variational Inference (SVI) algorithm with the reparametrization trick (Jospin et al. (2020)). A visual representation of the reparametrization trick method can be seen in Figure 1.

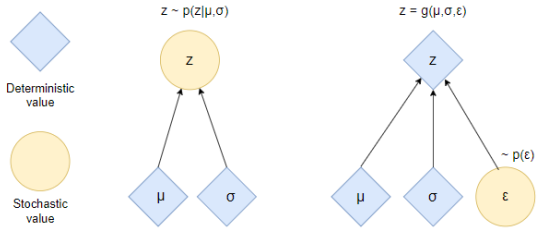

Fig. 1: The illustration reflects the method of the reparametrization trick - the left side shows that the output value is stochastic, while the right side shows that the stochastic parameter ϵ is parameterized to obtain a deterministic value z.

3 Methodology

3.1 Formal definition of the task

In this study, our objective is to conduct experiments under different levels of data noise conditions. To achieve this, we employ a technique called 'label noise injection', which involves the intentional modification of the values in categorical data within the training dataset. This method can be applied to both the input features and the classes - in this case, the classification of mushrooms as edible or inedible.

We conduct a series of experiments in which noise is injected into features, classes, or both. A specific program is designed for these experiments. This program randomly alters a specified percentage of the values of the training data. The process unfolds as follows: for a pre-defined percentage of records, the indices are randomly selected. If we are injecting noise into classes, which only contain values of 0 or 1, the program randomly flips the label of the selected percentage of records.

On the other hand, if we are injecting noise into features, the program replaces the original values for the selected indices with another valid value for that specific feature.

We conducted a series of six experiments, varying the noise injection rate from 0% (no injection) to 50%, with an increment of 10% for each successive experiment.

3.2 Dataset

This study uses the mushroom classification dataset, which comprises 8124 records and 23 distinct features related to various species of mushrooms. The dataset includes 22 descriptive features and one classification column, indicating whether the mushroom is poisonous (p) or edible (e). Each feature includes several categories, with the exception of the 'veil-type' feature, which is universally denoted as 'p' across all mushroom records. Other features fall into 2 to 12 distinct categories. As the data processing library used does not efficiently handle letter values representing categorical features, these values are transformed into numerical equivalents, where each unique number represents a different category. Lastly, the data set is divided into training and testing subsets through a random selection process, ensuring a representative distribution of data in both groups.

3.3 Applied approaches

The goal of our experiments is to compare the accuracy and standard deviation of the algorithms described under increasing levels of noise in the training data. During the experimental investigation we have used the following implementations of the algorithms described in the following subsections.

3.3.1 Variational Inference A commonly used Python library for implementing Variational Inference is PyMC3 (Salvatier et al. (2015)), which provides a variety of tools for probabilistic programming and Bayesian analysis.

After preparing our dataset, a Bayesian model is first defined using the pm.Model() function of PyMC3. This function allows us to construct a Bayesian model that includes all required prior probability distributions and associated likelihood functions. Within this model, we assume normal distributions, frequently symbolized by alpha and beta, as our initial prior distributions.

The mean, μ, is estimated as a linear combination of the features in the dataset. This function considers the initial probability distribution values and the features extracted from the dataset. Following this, the likelihood function is defined, typically employing the Bernoulli distribution, given the binary nature of many prediction tasks.

The Variational Inference approximation is then performed using the pm.fit() function provided by the PyMC3 library. This step is crucial to the VI algorithm. It aims to find the optimal parameters that maximize the Evidence Lower BOund (ELBO), simultaneously working to minimize the Kullback-Leibler (KL) divergence.

Once the variational approximation concludes, we draw samples from the approximated posterior distribution. These samples allow us to generate predictions on an independent test dataset. We assess the model's performance by comparing these predictions

with the actual outcomes from the test set. Metrics such as accuracy and the standard deviation of results provide insight into the model's performance and variability.

3.3.2 Monte Carlo dropout In implementing the Monte Carlo dropout (MC-D) algorithm, we utilize PyTorch's torch.nn library (Paszke et al. (2019)). The algorithm's execution happens in two distinct phases: training and testing.

First, we define the model's linear layers using the torch.nn.Linear class. Data is then propagated through these network layers, resulting in a tensor for each record which contains two values representing each of the final classes.

The performance of the algorithm is then evaluated using a cross-entropy loss calculation. Additionally, we introduce dropout layers after the ReLU activation function applied to the first and second layers. These dropout layers, set with a 70% dropout rate, aid in reducing overfitting and in enhancing the model's generalization capabilities by introducing an element of randomness.

The testing phase involves running the algorithm multiple times - 100 times in our case - on each group of input data. This process allows us to obtain a measure of confidence or variance for the results. We then tally the number of records correctly categorized by the model, adding this information to a list for graphical representation. Upon reaching a specified number of cycles, the results from the final testing cycle are collected.

For our neural network architecture, we establish three linear layers: input, hidden, and output, comprising 200, 300, and 2 neurons respectively. The output layer size corresponds with the number of classes in the dataset - poisonous or edible. We set the learning rate to 0.001 and execute the algorithm for 30 epochs. The data split used for our experiment is 80% for training and 20% for testing.

3.3.3 Bayes by backprop Bayes by backprop algorithm can be implemented using the PyTorch library, specifically using the torchbnn library (Lee et al. (2022)) for its Bayesian-specific functionalities. The neural network model is defined using the "Module" class from PyTorch, but the linear transformation layer is chosen from the "BayesLinear" function available in the torchbnn library. The activation function used is the ReLU function. An essential difference from other methods like the Monte Carlo Dropout (MC-D) is that for each layer, the input and output data size variables are defined along with the μ and σ values, which determine the distribution to define the weights of the model.

The BBB algorithm's training phase includes calculating the KL divergence value in addition to the cross-entropy error. Both these quantities are combined and used for optimizing the model to improve the total result. After the training phase, a testing phase is performed similarly to the MC-D algorithm, including error calculation, counting the correctly categorized records, and creating an error matrix.

Hyperparameter tuning is performed to determine the optimal configurations for the neural network. This tuning establishes the number of neurons for each layer (typically 200 and 2 for the input and output layers, respectively), the μ and σ values for all layers (usually both 0 and 0.1), and the weights for the KL divergence error and cross-entropy error (0.3 and 0.7, respectively). The training rate is generally set to 0.001, with 30

training cycles planned. The dataset is split into 80% for training and 20% for testing, with a batch size of 64, and 100 samples used for variance calculations.

4 Results

Results show that as the noise in the dataset increases, the accuracy of all algorithms decreases as seen in Figure 2. This observation is consistent for both clean and noisy data across all three types of noise additions. These results are expected because as the noise in the training data increases, the algorithm may start to emphasize random correlations instead of the true dependencies in the data. These findings highlight the importance of considering data noise when improving algorithm performance.

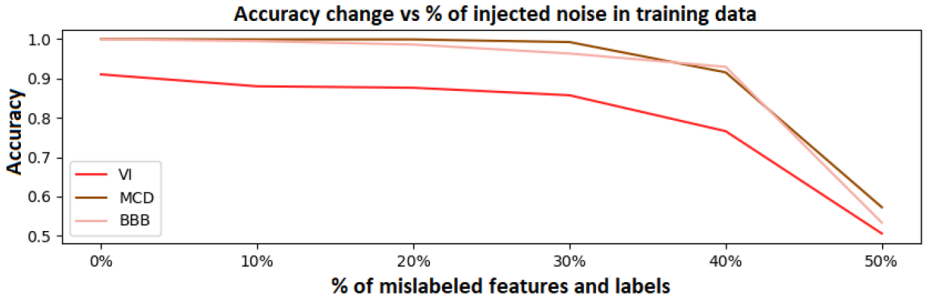

Fig. 2: Change in accuracy corresponding to the amount of added noise for each of the three algorithms.

Looking at the standard deviation values presented in Figure 3, there are noticeable differences between all algorithms. Variational inference (VI) exhibits the largest standard deviation, approximately 10 times greater than Bayes' Bayes by Backprop (BBB) and noticeably larger than Monte Carlo Dropout (MC-D). This higher standard deviation for VI corresponds to its lower accuracy, suggesting that the algorithm's frequent errors tend to yield values far from the average. It is also noticeable that for all algorithms, the standard deviation increases as the number of mixed labels grows, indicating that the algorithms are sensitive to the amount of noise. The largest changes in standard deviation with increasing noise are observed for the VI and MC-D algorithms. VI and MC-D seem to be the best candidates for esitmating noise in the dataset using standard deviation.

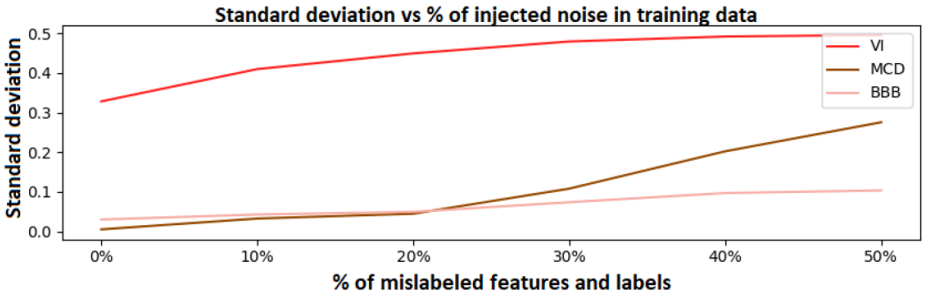

Fig. 3: Change in standard deviation in response to the quantity of incorporated noise for all three investigated algorithms.

Under conditions with 50% added noise, the highest accuracy and the lowest standard deviation are observed when noise is introduced exclusively to the features. However, scenarios where noise is added either solely to classes or to both classes and features yield similar accuracy results. This observation suggests differential impacts of class and feature noise on algorithm performance. It appears that output data noise may more strongly affect performance. A possible explanation is that, as the applied dataset contains 23 different features, the algorithms are relatively more resistant to introducing noise in the features, leveraging the information contained among the unmixed features. As seen in tables below, in most experiments MC-D showed the highest robustness with highest accuracy at different noise levels in both input features and output class labels. The negative correlation between accuracy and noise level is listed as $-r$ while the correlation between standard deviation and noise level is listed as r_{std}. From the results, it can be concluded that MC-D and BBB are more robust than VI, as these methods have significantly lower correlation between accuracies and noise levels. On the other hand, VI and MC-D captures better noise in standard deviation as these methods have a higher correlation between noise.

Table 1: Accuracy and standard deviation of each method for different levels of noise in input features and class labels.

Method	0%	10%	20%	30%	40%	50%	$-r$	r_{std}
MC-D	**100** $\pm$ 0.00	**99.94** $\pm$ 0.03	**99.79** $\pm$ 0.05	**98.18** $\pm$ 0.10	90.67 $\pm$ 0.20	**59.49** $\pm$ 0.27	**0.77**	**0.97**
VI	90.76 $\pm$ 0.33	88.49 $\pm$ 0.41	87.62 $\pm$ 0.45	85.71 $\pm$ 0.48	76.60 $\pm$ 0.49	51.17 $\pm$ 0.50	0.84	0.93
BBB	99.70 $\pm$ 0.05	98.35 $\pm$ 0.06	98.05 $\pm$ 0.06	94.60 $\pm$ 0.07	**92.93** $\pm$ 0.11	55.29 $\pm$ 0.11	**0.77**	0.92

Table 2: Accuracy and standard deviation of each method for different levels of noise in input features only.

Method	0%	10%	20%	30%	40%	50%	$-r$	r_{std}
MC-D	**100** ± 0.01	**99.94** ± 0.02	**99.88** ± 0.02	**99.88** ± 0.03	**99.34** ± 0.05	**95.77** ± 0.07	**0.73**	0.95
VI	91.63 ± 0.33	88.49 ± 0.35	87.81 ± 0.36	86.88 ± 0.37	86.08 ± 0.37	84.36 ± 0.38	0.96	**0.96**
BBB	99.70 ± 0.05	99.04 ± 0.05	98.32 ± 0.05	95.68 ± 0.07	94.46 ± 0.06	92.33 ± 0.06	0.97	0.65

Table 3: Accuracy and standard deviation of each method for different levels of noise in class labels only.

Method	0%	10%	20%	30%	40%	50%	$-r$	r_{std}
MC-D	**100** ± 0.01	**100** ± 0.03	**99.94** ± 0.04	**98.71** ± 0.08	**94.72** ± 0.10	50.45 ± 0.09	0.72	**0.95**
VI	90.76 ± 0.33	91.87 ± 0.39	89.78 ± 0.44	90.95 ± 0.48	83.93 ± 0.49	53.39 ± 0.50	0.75	**0.95**
BBB	99.82 ± 0.05	99.70 ± 0.05	99.70 ± 0.06	98.38 ± 0.09	96.91 ± 0.12	**57.20** ± 0.09	**0.70**	0.84

5 Discussion

This study offers insights into the robustness of Bayesian algorithms, such as MC-Dropout (MCD), Bayes by Backprop (BB), and Variational Inference (VI), against data noise. Examination of their performance under varying degrees of data cleanliness underscores the necessity of assessing these algorithms across diverse datasets.

We also identify promising areas for further research. Enriching the comparison with additional Bayesian methods could refine our understanding, while studying Black Box Variational Inference or Structured Mean-Field Variational Inference methods might offer further insight into the relationships between VI, MCD, and BBB algorithms.

Our findings highlight the significant impact of noise on accuracy and standard deviation. Future research should consider other potentially influencing factors, such as the complexity of the classification task and the size of the training dataset. Although BBB and MCD showed comparable performance across test conditions, it would be worthwhile exploring circumstances under which one might outperform the other, thus informing algorithm selection in specific contexts.

The study has uncovered intriguing evidence that class mixing influences algorithm performance more than feature mixing. This observation could inform the strategic allocation of resources for data cleaning and help in balancing accuracy, dispersion, and execution time. We also suggest investigating the role of sample size in result dispersion and the resilience of different machine learning algorithms to data noise from various sources.

Lastly, an intriguing possibility emerging from our study is that algorithms could infer the level of noise in a dataset, presenting a potential avenue for cost savings, particularly in contexts where data cleaning expenses are significant.

6 Conclusions

This study sheds light on the sensitivity of Bayesian and Monte Carlo Dropout (MC-D) algorithms to data noise. Evidently, the MC-D and Variational Inference (VI) algorithms, in comparison to Backpropagation by Bayesian (BBB) methods, demonstrate an amplified response to noise, which could reflect a more precise uncertainty modeling.

In terms of binary classification tasks under various noise levels, both MC-D and BBB prove robust.

Performance differences become more pronounced under low-noise conditions, with both BBB and MC-D achieving high accuracy. In particular, MC-D maintains 100% accuracy even after deliberate noise introduction. In contrast, VI exhibits lower accuracy and, even with hyperparameter tuning, fails to match the performance of MC-D and BBB.

Significant differences also extend to dispersion values. With the increase of training data noise, VI and MC-D's dispersion is markedly affected, whereas BBB, despite showing changes, exhibits minor shifts, which raises questions about its ability to model confidence effectively. The dispersion that increases with noise allows for the estimation of noise in the dataset which is a valuable feature in real-life datasets that are not always clean.

The study finds that up to 40% mixed training data, the algorithms under study can deliver high accuracy and stable dispersion. However, stability deteriorates and accuracy dips to approximately 50% when classes are mixed 50%. Interestingly, all three algorithms exhibit greater stability in feature-mixing scenarios than in class label-only or combined class label and input feature mixing scenarios.

Highlighting the differential impact of noise, we observe that class-level noise in training data exerts a greater effect than feature-level noise. However, even with mixed 50% of all features, the algorithms maintain high accuracy and relatively low standard deviation. This stability is not mirrored when training data classes are mixed, emphasizing the importance of output noise over input noise in the training data.

References

Ai, Q., Liu, S., He, L., Xu, Z. (2021). Stein variational gradient descent with multiple kernels, *Cognitive Computation* **15**, 672–682.

Blei, D. M., Kucukelbir, A., McAuliffe, J. D. (2016). Variational inference: A review for statisticians, *Journal of the American Statistical Association* **112**, 859 – 877.

Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D. (2015). Weight uncertainty in neural networks, *ArXiv* **abs/1505.05424**.

Gal, Y., Ghahramani, Z. (2015). Dropout as a bayesian approximation: Representing model uncertainty in deep learning, *ArXiv* **abs/1506.02142**.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., Rubin, D. B. (2013). *Bayesian Data Analysis*, CRC press.

Hoffman, M. D., Blei, D. M. (2014). Structured stochastic variational inference, *arXiv: Learning* .

Hoffman, M. D., Blei, D. M., Wang, C., Paisley, J. W. (2012). Stochastic variational inference, *ArXiv* **abs/1206.7051**.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., Saul, L. K. (1999). An introduction to variational methods for graphical models, *Machine Learning* **37**, 183–233.

Jospin, L. V., Buntine, W. L., Boussaid, F., Laga, H., Bennamoun (2020). Hands-on bayesian neural networks—a tutorial for deep learning users, *IEEE Computational Intelligence Magazine* **17**, 29–48.

Kingma, D. P., Welling, M. (2013). Auto-encoding variational bayes, *CoRR* **abs/1312.6114**.

Kucukelbir, A., Ranganath, R., Gelman, A., Blei, D. M. (2015). Automatic variational inference in stan, *NIPS*.

Lee, S., Kim, H., Lee, J. (2022). Graddiv: Adversarial robustness of randomized neural networks via gradient diversity regularization, *IEEE Transactions on Pattern Analysis and Machine Intelligence* .

Nguyen, T. V., Bonilla, E. V. (2013). Efficient variational inference for gaussian process regression networks, *International Conference on Artificial Intelligence and Statistics*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library, *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., pp. 8024–8035.
`http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-\`
`high-performance-deep-learning-library.pdf`

Ranganath, R., Gerrish, S., Blei, D. M. (2013). Black box variational inference, *International Conference on Artificial Intelligence and Statistics*.

Salvatier, J., Wiecki, T. V., Fonnesbeck, C. J. (2015). Probabilistic programming in python using pymc, *arXiv: Computation* .

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research* **15**.