Baltic J. Modern Computing, Vol. 13 (2025), No. 1, pp. 290–303
https://doi.org/10.22364/bjmc.2025.13.1.15

# Comparison of Bayesian Neural Network Methods under Noisy Conditions

Rinalds Daniels PIKŠE, Evalds URTANS

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

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

ORCID 0009-0005-4070-762X, ORCID 0000-0001-9813-0548

**Abstract.** This study presents an analysis of the effects of noise on the performance of Bayesian neural networks, comparing Monte Carlo Dropout (MC-D), Bayes by Backpropagation (BBB), and Variational Inference (VI) algorithms. 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 performance of the methods are evaluated using uncertainty estimates, accuracy, and robustness. This study gives insights into which method works best in noisy data and how well they can estimate the noise in the data. The results show that Bayes backpropagation is the most robust method in noisy conditions, while Monte Carlo Dropout is the most accurate in noise-free conditions. Variational Inference is the most sensitive to noise, but can be used as an estimator of the noisiness of the data set

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

## 1 Introduction

Bayesian Neural Networks (BNNs) represent a significant advancement in neural network architectures, integrating Bayesian statistics to quantify predictive uncertainty. Unlike traditional networks, BNNs provide both point predictions and a probability distribution of outcomes, offering a more detailed assessment of model confidence. The ongoing development of BNN methodologies has substantially broadened their scope and utility, allowing more accurate estimation of weight uncertainties within the network (Jospin et al. (2020)).

This research specifically examines the most widely used methods of Variational Inference (VI), Monte Carlo Dropout (MC-D), and Bayes by Backpropagation (BBB).

These methods stand out for their substantial contributions and comprehensive theoretical underpinnings in uncertainty management. VI (Jordan et al. (1999)) enables the scalable approximation of complex posterior distributions. MC-D (Gal and Ghahramani (2016)) innovatively uses network dropout for Bayesian inference, offering an efficient alternative to traditional methods. Meanwhile, the BBB (Blundell et al. (2015)) seamlessly integrates Bayesian inference with the backpropagation process, providing a direct measure of weight uncertainty. The selection of VI, MC-D, and BBB reflects their pivotal roles in the evolution of BNN research and their adaptability to diverse application scenarios.

The aim of this study is to evaluate the performance of these three BNN methodologies in the presence of noise, a common challenge in real-world data environments. Noise, defined as random or irrelevant data that disrupt the learning process, can significantly impact the accuracy and reliability of the model. By introducing noise to the mushroom dataset, we simulate real-world data imperfections, enabling a comprehensive assessment of each BNN method's response to varying noise levels. This analysis seeks to identify the most effective BNN approach for managing uncertainty in noisy data, thereby informing practitioners' selection of appropriate methodologies for their specific applications. The purpose of this paper also is to find the methods that could be used to estimate the noise of the data set. The results of this study will provide valuable information on the relative strengths and limitations of VI, MC-D, and BBB, improving our understanding of the practical applicability of BNN in managing uncertainty in different data environments.

## 2 Related work

Recent developments in Bayesian neural networks (BNN) have introduced a variety of methods to integrate uncertainty quantification into deep learning. Jospin et al. (2020) provide a comprehensive classification of BNNs and a workflow for the design and implementation of BNN, focusing on the transition from conventional deep learning to Bayesian methods.

Specific research has also analyzed the performance of the methods used in this investigation. The Bayes backpropagation algorithm (BBB), introduced by Blundell et al. (2015), shows a performance comparable to the Monte Carlo dropout. This work underscores the effectiveness of BNNs in uncertainty estimation in different data scenarios. Olivier et al. (2021) investigate variational Bayesian inference (VI), highlighting the limitations of VI, and proposing enhancements for more accurate uncertainty quantification. Their work suggests the necessity of refining VI through alternative metrics and model averaging.

Other research has focused on investigating the relationship between uncertainty and "noisy" training data. Pawlowski et al. (2018) introduced "Bayes by Hypernet" (BbH), using hypernetworks as implicit distributions to model weight uncertainty. Their findings indicate that varying noise levels markedly affect predictive uncertainty, underscoring the importance of considering noise in BNN models. Furthermore, Shridhar et al. (2019) emphasizes the distinction between aleatoric and epistemic uncertainties for targeted model improvements, highlighting that aleatoric uncertainty is merely a

measure of noisy data. This distinction helps identify whether data quality or model limitations contribute to uncertainty, guiding strategies for improvement.

Building on these studies, our research compares Variational Inference (VI), Bayes by Backpropagation (BBB), and Monte Carlo Dropout (MC-D) against the mushroom dataset with controlled noise variations. By evaluating how each method responds to noise and quantifies uncertainty, this analysis aims to clarify their applicability and effectiveness in noisy conditions, contributing to the optimal selection of BNN methodologies for uncertainty management.

### 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 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. (2013a)). 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 black-box variational inference methods (Ranganath et al. (2014)), which allow maximizing ELBO, but often rely on stochastic optimization (Kucukelbir et al. (2015)), hence they can be considered a subgroup of SVI algorithms.

Structured Mean-Field Variational Inference (SMFVI) is a method that maintains dependencies between variables (Hoffman et al. (2013b)). This method is practical when variable relationships are known. Non-parametric models perform variational inference by creating a more complex value distribution that can contain several modes

(Nguyen and Bonilla (2013)), as illustrated in Figure 1. This figure provides an example of the multimodal problem, where a single-mode probability distribution would not fully characterize the true probability.

It is important to note that VI methods are not trained using backpropogation as other methods used in this research.

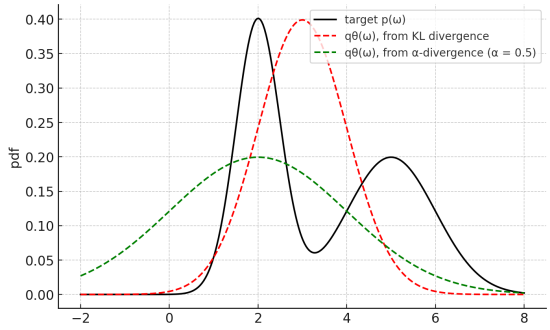


Fig. 1: Visualization of the multimodal problem, where a single-mode probability distribution would not fully characterize the true probability using KL divergence or alpha divergence.

### 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 artifical 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 (2016)) 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 the calculations, thus introducing variability in the model output after 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 influenced by a distinct dropout-induced network configuration.

MC-D consists of Dropout function that is used in training phase and testing phase. Normally, Dropout would be used only in training phase. In the training phase, dropout is used to prevent overfitting by randomly deactivating neurons. In the testing phase, dropout is used to approximate the posterior distribution of the model. This is done by running the model multiple times with different dropout configurations and aggregating the results. This method is used to estimate the uncertainty of the model. Dropout function is placed after the activation function in the neural network. The dropout rate is set to 0.5, which means that 50% of the neurons are deactivated.

### 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 using 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 $\mu$ and $\sigma$. During backpropagation, neural networks employ gradient descent, which requires 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.

This is achieved by initial sampling $\epsilon$ from a standard normal distribution, $N(0, I)$. This sample is subsequently transformed through the parameters of the weight distribution, $\mu$ (mean) and $\sigma$ (standard deviation), to produce:

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

In this equation, $g_{\varphi}(\epsilon, x)$ represents a deterministic function that models a sample from the weight distribution, facilitating the direct application of backpropagation for the computation of gradients. This deterministic transformation preserves the stochastic properties essential for Bayesian inference while streamlining the optimization process.

By adopting the reparametrization trick, BBB effectively transforms the stochastic variables into a format amenable to gradient-based optimization, combining the stochastic nature of Bayesian inference with the efficiency of deterministic gradient descent. Because of this, BBB can be comparable to 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 2. To model probabilistic neural networks, the reparameterization trick is used to transform stochastic variables into deterministic ones. Deterministic values are necessary for backpropagation, which is used to optimize the weights of the neural network.

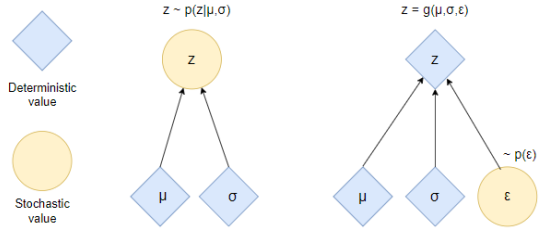


Fig. 2: Visualization of the reparametrization trick. In the middle of the image fully stochastic representation which is not derivable. In the right side of the image it is achieved by transforming the stochastic variable $\epsilon$ with the parameters $\mu$ and $\sigma$ which are derivable.

## 3 Methodology

### 3.1 Formal definition of the task

In our experiments, we simulate real-world data noise by introducing it to both features and classes within our training data set. We use a custom program that randomly modifies a predetermined portion of the data. Specifically, for class noise, it flips the binary labels of a randomly selected subset of the data. For feature noise, it substitutes the original feature values with other legitimate values for a random set of records. This approach aims to mimic the variability and imperfections often found in real-world data, potentially impacting model predictions.

To evaluate our model's performance after adding noise, we focus on two metrics: accuracy and its standard deviation. Accuracy provides a quick measure of how often the model predicts correctly, often used in classification tasks. The standard deviation

of accuracy shows the model's consistency across tests, which is important for assessing stability in real-world scenarios. Together, they offer a concise evaluation of the effectiveness and reliability of a model.

We carried out six experiments, incrementally adjusting the noise rate from 0% to 50% in 10% steps.

### 3.2 Dataset

In this study, we employ the widely recognized mushroom classification dataset, sourced from the UCI Machine Learning Repository (Dua and Graff (2019)), consisting of 8,124 instances and 23 features that detail various species of mushrooms. This data set is particularly chosen for its balanced mix of 22 descriptive features and a binary classification column, poisonous (p) or edible (e), making it an ideal candidate for evaluating model performance in binary classification tasks. The inclusion of a diverse range of features, from 2 to 12 distinct categories each, introduces a level of complexity that mimics real-world data challenges, enhancing the relevance of our experiments. We converted categorical values to numerical codes to accommodate our data processing library's limitations, ensuring that the dataset's structure is preserved. The data set was randomly divided into training and testing subsets to ensure a representative sample. While using a single dataset might seem limiting, the mushroom dataset's balance between complexity and manageability uniquely positions it to explore the effects of noise injection on model performance. Other studies have not used this dataset, so our results will provide a fresh perspective on the performance of Bayesian neural networks in a binary classification task. In fact, other studies have not attempted to compare the performance of these three methods on the same dataset using methods that introduce noise to the data.

### 3.3 Applied approaches

The goal of our experiments is to compare the accuracy and standard deviation of the algorithms described with 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 prior required 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, $\mu$, is estimated as a linear combination of the characteristics of the data set. 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.

In Bayesian models employing VI, the prediction of mean is represented as a linear projection of the input features. For an input vector shown in Equation (2) the model typically defines the mean prediction as Equation (3) where $w$ in Equation (4) is the weight vector (coefficients) associated with each feature and $b$ is the bias term.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \tag{2}$$

$$\mu(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{d} w_i x_i + b \tag{3}$$

$$\mathbf{w} = [w_1, w_2, \ldots, w_d]^T \tag{4}$$

In VI each weight $w_i$ is treated as a random variable with an approximate posterior distribution. Generally, we assume a Gaussian variational posterior as shown in Equation (5) where $mu_i$ is the mean (expected value) of the weight $w_i$, and $\sigma_i^2$ is the variance of the weight $w_i$.

$$q(w_i) = \mathcal{N}(w_i; \mu_i, \sigma_i^2) \tag{5}$$

Thus, when making predictions, one often uses the means of these weight distributions, resulting in the mean prediction being computed as shown in Equation (6).

$$\mu(\mathbf{x}) = \sum_{i=1}^{d} \mu_i \, x_i + b. \tag{6}$$

In this expression, the coefficients of the linear combination are the parameters $\mu_i$, which are learned during the variational optimization process (typically by maximizing the Evidence Lower BOund, or ELBO).

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 is complete, we draw samples from the approximated posterior distribution. These samples allow us to generate predictions on an independent test data set. We evaluated the performance of the model by comparing these predictions with the actual outcomes of the test set. Metrics such as accuracy and the standard deviation of results provide insight into the performance and variability of the model.

**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 execution of the algorithm occurs in two distinct phases: training and testing.

First, we define the model's linear layers using the torch.nn.Linear class. The data is then propagated through these network layers, resulting in a tensor for each record that 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. By setting the dropout layers to a 70% rate, we mitigate overfitting and increase the model's ability to generalize across unseen data through the introduction of randomness.

During the testing phase, the Monte Carlo Dropout (MC-D) algorithm is executed 100 times on the same set of same input data for each test instance. This repeated execution strategy is designed to emulate the stochastic nature of dropout, allowing us to approximate the posterior distribution of model predictions. By aggregating the results of these 100 runs, we generate a distribution of predictions for each data point, from which we derive a measure of confidence or variance in the model results. For each run, we record the accuracy, specifically, the number of records correctly identified by the model, and compile these accuracy figures for graphical analysis. The aggregation of results after the predetermined 100 cycles provides a comprehensive overview of the model's performance consistency and its predictive confidence across the entire dataset.

The artificial neural network for the Monte Carlo Dropout (MC-D) algorithm comprises three layers: an input layer with 200 neurons, despite the dataset having 23 features, to allow for a richer feature representation through learned combinations; a hidden layer with 300 neurons to adequately learn data complexities without overfitting; and an output layer with 2 neurons corresponding to classification task. The neural network architecture has been chosen on the basis of preliminary testing to ensure optimal performance and learning efficiency. The number of layers is the same for all three methods to ensure a fair comparison.

We chose a learning rate of 0.001 and 30 epochs for training based on preliminary tests that showed an optimal balance between learning efficiency and avoiding overfitting. The 80%/20% training/test data split follows common practice for adequate learning and validation.

**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 such as Monte Carlo Dropout (MC-D) is that, for each layer, the input and output data size variables are defined along with the values of $\mu$ and $\sigma$, which determine the distribution to define the weights of the model.

The BBB algorithm training phase includes calculating the KL divergence value in addition to the cross-entropy error. Both of these quantities are combined and used to

optimize 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 adjustment establishes the number of neurons for each layer (typically 200 and 2 for the input and output layers, respectively), the $\mu$ and $\sigma$ values for all layers (usually 0 and 0.1), and the weights for the KL divergence error and the cross-entropy error (0.3 and 0.7, respectively). The learning rate is set at 0.001 for 30 training epochs. The data set is divided 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 data set increases, the accuracy of all algorithms decreases, as seen in Figure 3. This observation is consistent with clean and noisy data for 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.

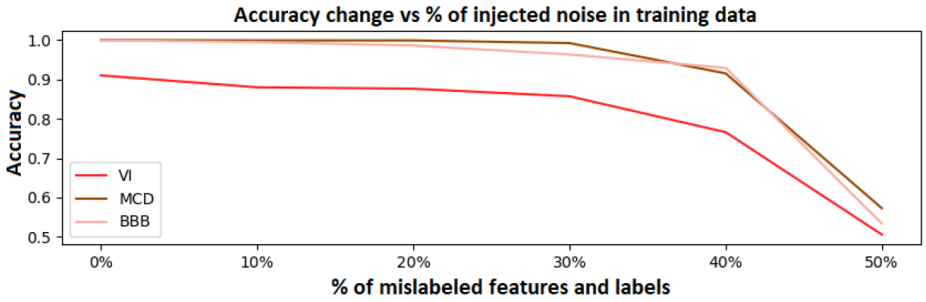


Fig. 3: Change in accuracy corresponding to the amount of added noise for each of the three algorithms.

Looking at the standard deviation values presented in Figure 4, there are noticeable differences between all algorithms. Variational inference (VI) exhibits the largest standard deviation, approximately 10 times greater than 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.

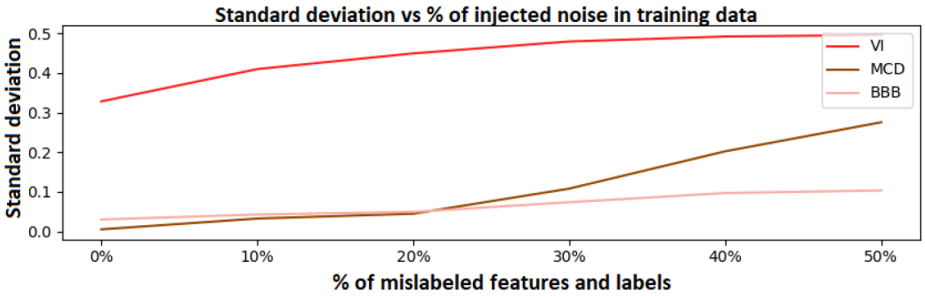


Fig. 4: Change in standard deviation in response to the quantity of incorporated noise for all three investigated algorithms.

Under conditions with 50% added noise, the highest accuracy and the lowest standard deviation are observed when noise is introduced exclusively to the features (Table 2). However, scenarios where noise is added solely to classes (Table 3) or to both classes and features (Table 1) 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, since the applied data set contains 23 different features, the algorithms are relatively more resistant to introducing noise into the features, using the information contained among the unmixed features. As seen in the 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 precision 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 a 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** $\pm$ 0.01 | **99.94** $\pm$ 0.02 | **99.88** $\pm$ 0.02 | **99.88** $\pm$ 0.03 | **99.34** $\pm$ 0.05 | **95.77** $\pm$ 0.07 | **0.73** | 0.95 |
| VI | 91.63 $\pm$ 0.33 | 88.49 $\pm$ 0.35 | 87.81 $\pm$ 0.36 | 86.88 $\pm$ 0.37 | 86.08 $\pm$ 0.37 | 84.36 $\pm$ 0.38 | 0.96 | **0.96** |
| BBB | 99.70 $\pm$ 0.05 | 99.04 $\pm$ 0.05 | 98.32 $\pm$ 0.05 | 95.68 $\pm$ 0.07 | 94.46 $\pm$ 0.06 | 92.33 $\pm$ 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** $\pm$ 0.01 | **100** $\pm$ 0.03 | **99.94** $\pm$ 0.04 | **98.71** $\pm$ 0.08 | **94.72** $\pm$ 0.10 | 50.45 $\pm$ 0.09 | 0.72 | **0.95** |
| VI | 90.76 $\pm$ 0.33 | 91.87 $\pm$ 0.39 | 89.78 $\pm$ 0.44 | 90.95 $\pm$ 0.48 | 83.93 $\pm$ 0.49 | 53.39 $\pm$ 0.50 | 0.75 | **0.95** |
| BBB | 99.82 $\pm$ 0.05 | 99.70 $\pm$ 0.05 | 99.70 $\pm$ 0.06 | 98.38 $\pm$ 0.09 | 96.91 $\pm$ 0.12 | **57.20** $\pm$ 0.09 | **0.70** | 0.84 |

## 5 Discussion

This study offers insights into the robustness of Bayesian algorithms, such as MC-Dropout (MC-D), Bayes by Backprop (BBB), 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, MC-D, 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 MC-D 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.

VI exhibits inferior performance under noisy conditions, primarily due to its estimation of the true posterior being extremely sensitive to data noise. The integration of hierarchical variational methods with annealed objectives may mitigate these issues, enhancing robustness against noise.

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 to be robust methods.

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-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

Blei, D. M., Kucukelbir, A., McAuliffe, J. D. (2016). Variational inference: A review for statisticians, *Journal of the American Statistical Association* **112**, 859 – 877. `https://arxiv.org/abs/1601.00670`

Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D. (2015). Weight uncertainty in neural networks, *ArXiv* **abs/1505.05424**. `http://proceedings.mlr.press/v37/blundell15.pdf`

Dua, D., Graff, C. (2019). Mushroom data set, `https://archive.ics.uci.edu/ml/datasets/mushroom`. UCI Machine Learning Repository.

Gal, Y., Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning., *in* Balcan, M.-F., Weinberger, K. Q. (eds), *ICML*, Vol. 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 1050–1059. `http://dblp.uni-trier.de/db/conf/icml/icml2016.html#GalG16`

Hoffman, M. D., Blei, D. M., Wang, C., Paisley, J. (2013a). Stochastic variational inference, *Journal of Machine Learning Research* **14**, 1303–1347. `http://www.jmlr.org/papers/v14/hoffman13a.html`

Hoffman, M. D., Blei, D. M., Wang, C., Paisley, J. (2013b). Stochastic variational inference, *Journal of Machine Learning Research* **14**, 1303–1347.
http://www.jmlr.org/papers/v14/hoffman13a.html

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.
http://arxiv.org/abs/2007.06823

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

Kucukelbir, A., Ranganath, R., Gelman, A., Blei, D. M. (2015). Automatic variational inference in stan, *NIPS*, pp. 568–576.
http://dblp.uni-trier.de/db/conf/nips/nips2015.html#KucukelbirRGB15

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* .
https://ieeexplore.ieee.org/document/9761760

Nguyen, T. V., Bonilla, E. V. (2013). Efficient variational inference for gaussian process regression networks., *AISTATS*, Vol. 31 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 472–480.
http://dblp.uni-trier.de/db/conf/aistats/aistats2013.html#NguyenB13

Olivier, A., Shields, M. D., Graham-Brady, L. (2021). Bayesian neural networks for uncertainty quantification in data-driven materials modeling, *Computer Methods in Applied Mechanics and Engineering* **386**, 114079.

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, *NeurIPS*, Curran Associates, Inc., pp. 8024–8035.
http://dblp.uni-trier.de/db/conf/nips/nips2019.html#PaszkeGMLBCKLGA19

Pawlowski, N., Brock, A., Lee, M. C. H., Rajchl, M., Glocker, B. (2018). Implicit weight uncertainty in neural networks.
https://arxiv.org/abs/1711.01297

Ranganath, R., Gerrish, S., Blei, D. M. (2014). Black box variational inference.

Salvatier, J., Wiecki, T. V., Fonnesbeck, C. J. (2015). Probabilistic programming in python using pymc, *arXiv: Learning* .
https://arxiv.org/abs/1507.08050

Shridhar, K., Laumann, F., Liwicki, M. (2019). A comprehensive guide to bayesian convolutional neural network with variational inference, *ArXiv* **abs/1901.02731**.
http://arxiv.org/abs/1901.02731

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**, 1929–1958.
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Received September 7, 2023 , revised February 21, 2025, accepted March 3, 2025