REPRESENTING INFERENTIAL UNCERTAINTY IN DEEP NEURAL NETWORKS THROUGH SAMPLING

Patrick McClure & Nikolaus Kriegeskorte
MRC Cognition and Brain Sciences Unit
University of Cambridge
Cambridge, UK
{patrick.mcclure,nikolaus.kriegeskorte}@mrc-cbu.cam.ac.uk

ABSTRACT

As deep neural networks (DNNs) are applied to increasingly challenging problems, they will need to be able to represent their own uncertainty. Modelling uncertainty is one of the key features of Bayesian methods. Bayesian DNNs that use dropout-based variational distributions and scale to complex tasks have recently been proposed. We evaluate Bayesian DNNs trained with Bernoulli or Gaussian multiplicative masking of either the units (dropout) or the weights (dropconnect). We compare these Bayesian DNNs ability to represent their uncertainty about their outputs through sampling during inference. We tested the calibration of these Bayesian fully connected and convolutional DNNs on two visual inference tasks (MNIST and CIFAR-10). By adding different levels of Gaussian noise to the test images in z-score space, we assessed how these DNNs represented their uncertainty about regions of input space not covered by the training set. These Bayesian DNNs represented their own uncertainty more accurately than traditional DNNs with a softmax output. We find that sampling of weights, whether Gaussian or Bernoulli, led to more accurate representation of uncertainty compared to sampling of units. However, sampling units using either Gaussian or Bernoulli dropout led to increased convolutional neural network (CNN) classification accuracy. Based on these findings we use both Bernoulli dropout and Gaussian dropconnect concurrently, which approximates the use of a spike-and-slab variational distribution. We find that networks with spike-and-slab sampling combine the advantages of the other methods: they classify with high accuracy and robustly represent the uncertainty of their classifications for all tested architectures.

1 INTRODUCTION

Deep neural networks (DNNs), particularly convolutional neural networks (CNN), have recently been used to solve complex perceptual and decision tasks (Krizhevsky et al., 2012; Mnih et al., 2015; Silver et al., 2016). However, these networks fail to model the uncertainty of their predictions or actions. Although many networks deterministically map an input to a probabilistic prediction, they do not model the uncertainty of that mapping. In contrast, Bayesian neural networks (NNs) attempt to learn a distribution over their parameters thereby offering uncertainty estimates for their outputs (MacKay, 1992; Neal, 2012). However, these methods do not scale well due to the difficulty in computing the posterior of a network's parameters.

One type of method for sampling from the posteriors of these networks is Hamiltonian Monte Carlo (HMC) (Neal, 2012). These techniques use the gradient information calculated using backpropagation to perform Markov chain Monte Carlo (MCMC) sampling by randomly walking through parameter space. proposed stochastic gadient Langevien dynamcis (SGLD)

Approximate methods, in particular variational inference, have been used to make Bayesian NNs more tractable (Hinton & Van Camp, 1993; Barber & Bishop, 1998; Graves, 2011; Blundell et al., 2015). Due in large part to the fact that these methods substantially increase the number of parameters in a network, they have not been applied to large DNNs, such as CNNs. Gal & Ghahramani

(2016) and Kingma et al. (2015) bypassed this issue by developing Bayesian CNNs using dropout (Srivastava et al., 2014). Dropout is a widely used regularization technique where units are dropped out of a network with a probability p during training and the output of all unit are multiplied by p during inference. A similar technique is dropconnect (Wan et al., 2013), which drops network connections instead of units. Gal & Ghahramani (2015) detailed how dropping units was equivalent to sampling weights from a Bernoulli-based variational distribution and that in order to make a DNN with dropout Bayesian, sampling should be used during both training and inference. Monte-Carlo (MC) sampling at inference allows a DNN to efficiently model a distribution over its outputs. One limitation of the Bayesian dropout method is that it does not model the uncertatiniy of each network parameter. The uncertainty of a DNN can then be calculated using this probability distribution. In addition to Bernoulli and Gaussian distributions, there has also been work done using spike-an-slab distributions (Louizos, 2015), a combination of the two, which has been shown to increase the quality of linear regression (Ishwaran & Rao, 2005). Interestingly, dropout and dropconnect can be seen as approximations to spike-and-slab distributions for units and weights, respectively (Louizos, 2015; Gal, 2016; Li et al., 2016). Dropout- and dropconnect-based variational DNNs are dependent on the dropout probability, which is often used as a hyperparameter. However, work has been done on automatically learning the dropout probability during training for dropconnect (Louizos, 2015) using spike-and-slab distributions and Gaussian dropout (Kingma et al., 2015).

In this paper, we investigate how using MC sampling to model uncertainty affects a network's probabilistic predictions. Specifically, we test if using MC sampling improves the calibration of the probabilistic predictions made by Bayesian DNNs with softmax output layers. We used variational distributions based on dropout and dropconnect with either Bernoulli or Gaussian sampling during both training and inference. Additonally, we propose a formulation of a spike-and-slab variational distribution based on Bernoulli dropout and Gaussian dropconnect. We find that the spike-and-slab networks robustly represented their uncertainty like Bayesian dropconnect networks and have the increased CNN classification accuracy of Bayesian dropout networks. Each of these variational distributions scale extremely well and make the results of this work applicable to a large range of state-of-the-art DNNs.

2 METHODS

2.1 BAYESIAN NEURAL NETWORKS

Artificial neural networks (NNs) can be trained using Bayesian learning by finding the maximum a posteriori (MAP) weights given the training data (D_{train}) and a prior over the weight matrix W ($p(W)$):

$$\max_W p(W|D_{train}) = \max_W p(D_{train}|W)p(W) \tag{1}$$

This is usually done by minimizing the mean squared error (MSE) or cross entropy error for either regression or classification, respectively, while using L2 regularization, which corresponds to a Gaussian prior over the weights. At inference, the probability of the test data (D_{test}) is then calculated using only the maximum likelihood estimate (MLE) of the weights (W^*):

$$p(D_{test}|W^*) \tag{2}$$

However, ideally the full posterior distribution over the weights would be learned instead of just the MLE:

$$p(W|D_{train}) = \frac{p(D_{train}|W)p(W)}{p(D_{train})} \tag{3}$$

This can be intractable due to both the difficulty in calculating $p(D_{train})$ and in calculating the joint distribution of a large number of parameters. Instead, $p(W|D_{train})$ can be approximated using a variational distribution $q(W)$. This distribution is constructed to allow for easy generation of samples. Using variational inference, $q(W)$ is learned by minimizing:

$$-\int \log\ p(D_{train}|W)q(W)dW + KL(q(W)||p(W)) \tag{4}$$

Monte-Carlo (MC) sampling can then be used to estimate the probability of test data using $q(W)$:

$$p(D_{test}) \approx \frac{1}{n}\sum_{i}^{n} p(D_{test}|\hat{W}^i)\ \ where\ \ \hat{W}^i \sim q(W) \tag{5}$$

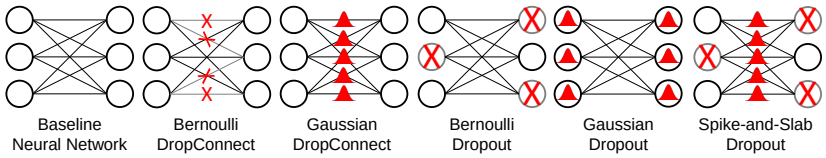

Figure 1: A visualization of the different variational distributions on a simple neural network.

2.2 Variational Distributions

The number and continuous nature of the parameters in DNNs makes sampling from the entire distribution of possible weight matrices computationally challenging. However, variational distributions can make sampling easier. In deep learning, the most common sampling method is dropout with Bernoulli variables. However, dropconnect, which independently samples a Bernoulli for each weight, and Gaussian weights have also been used. A visualization of the different methods is shown in Figure 1. All of these methods can be formulated as variational distributions where weights are sampled by element-wise multiplying the variational parameters V, the $n \times n$ connection matrix with an element for each connection between the n units in the network, by a mask $\hat{M}$, which is sampled from some probability distribution. Mathematically, this can be written as:

$$\hat{W} = V \circ \hat{M}\ \ where\ \ \hat{M} \sim p(M) \tag{6}$$

From this perspective, the difference between dropout and dropconnect, as well as Bernoulli and Gaussian methods, is simply the probability distribution used to generate the mask sample, $\hat{M}$ (Figure 2).

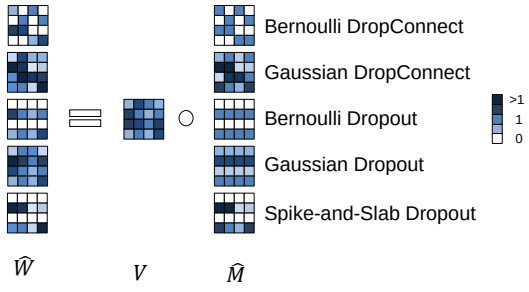

Figure 2: An illustration of sampling network weights using the different variational distributions.

2.2.1 Bernoulli DropConnect & Dropout

Bernoulli distributions are simple distributions which return 1 with probability p and 0 with probability $(1-p)$. In Bernoulli dropconnect, each element of the mask is sampled independently, so $\hat{m}_{i,j} \sim Bernoulli(p)$. This sets $\hat{w}_{i,j}$ to $v_{i,j}$ with probability p and 0 with a probability $(1-p)$.

In dropout, however, the weights are not sampled independently. Instead, one Bernoulli variable is sampled for each row of the weight matrix, so $\hat{m}_{i,*} \sim Bernoulli(p)$.

2.2.2 GAUSSIAN DROPCONNECT & DROPOUT

In Gaussian dropconnect and dropout, the elements of the mask are sampled from normal distributions. This corresponds to sampling $\hat{w}_{i,j}$ from a Gaussian distribution centered at variational parameter $v_{i,j}$. Srivastava et al. (2014) proposed using Gaussian distribution with a mean of 1 and a variance of $\sigma_{dc}^2 = (1-p)/p$, which matches the mean and variance of dropout when training time scaling is used. In Gaussian dropconnect, each element of the mask is sampled independently, which results in $\hat{m}_{i,j} \sim \mathcal{N}(1, \sigma_{dc}^2)$. In Gaussian dropout, each element in a row has the same random variable, so $\hat{m}_{i,*} \sim \mathcal{N}(1, \sigma_{dc}^2)$.

2.2.3 SPIKE-AND-SLAB DROPOUT

A spike-and-slab distribution is the normalized linear combination of a "spike" of probability mass at zero and a "slab" consisting of a Gaussian distribution. This spike-and-slab returns a 0 with probability p_{spike} or a random sample from a Gaussian distribution $\mathcal{N}(\mu_{slab}, \sigma_{slab}^2)$. We propose concurrently using Bernoulli dropout and Gaussian dropconnect to approximate the use of a spike-and-slab variational distribution by optimizing a lower-bound of the objective function (See Appendix A). In this formulation, $m_{i,j} \sim b_{i,*}\mathcal{N}(1, \sigma_{dc}^2)$, where $b_{i,*} \sim Bern(p_{do})$ for each mask row and $\sigma_{dc}^2 = p_{dc}/(1-p_{dc})$. As for Bernoulli dropout, each row of the mask M, $m_{i,*}$, is multiplied by 0 with probability $(1-p_{do})$, otherwise each element in that row is multiplied by a value independently sampled from a Gaussian distribution as in Gaussian dropconnect. During non-sampling inference, spike-and-slab dropout uses the mean weight values and, per Bernoulli dropout, multiplies unit outputs by p_{do}. This differs from the work done by Louizos (2015) and Gal (2016) in that they used additive Gaussian noise and learn separate means and variances for each weight. In contrast, we define the variance as a function of the learned weight mean $v_{i,j}$. Tying the variance of a weight to its magnitude makes it only beneficial to learn large weights if they are robust to variance (Wang & Manning, 2013). Although we treat p_{do} and p_{dc} as a hyperparameters thereby reducing the space of variational distributions we optimize over, similar methods could potentially learn these during training (Louizos, 2015; Kingma et al., 2015; Gal, 2016).

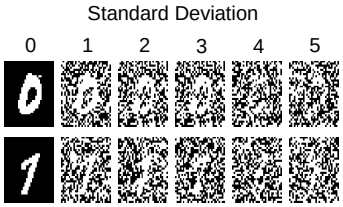

Figure 3: Examples of MNIST images with added Gaussian noise with varying standard deviations.

3 RESULTS

In this paper, we investigate how using MC sampling affects a DNN's ability to represent the uncertainty of it's probabalistic predictions. To test this, we trained several networks differing only in whether no sampling was performed (baseline DNN and DNN with L2-regularization), sampling was only performed during training (dropout and dropconnect), or sampling was performed both during training and inference (MC dropout and MC dropconnect). We used the MNIST and CIFAR-10 datasets to train networks that sampled from different variational distribution using the above methods.

For these DNNs, we compared the test classification error, the uncertainty of the softmax output, and the calibration of the softmax output for each type of sampling and variational distribution. The test classification error shows how well the probability distribution learned by each DNN models

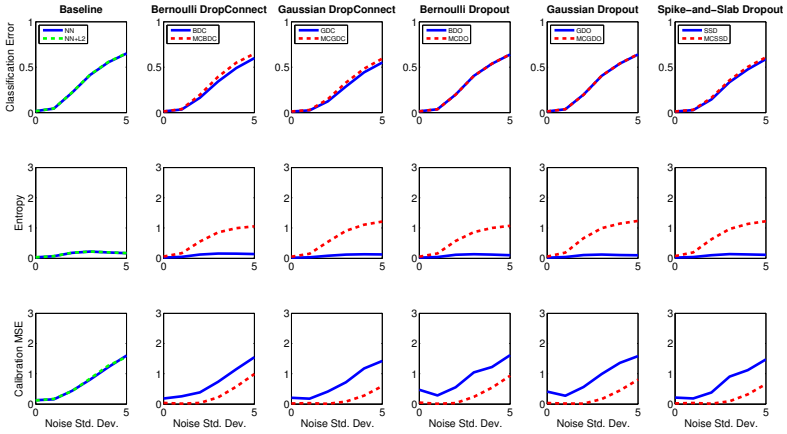

Figure 4: The MNIST test classification error, entropy, and calibration of the predictions of the fully connected networks: NN, NN+L2, Bernoulli DropConnect (BDC) with and without Monte-Carlo (MC) sampling, Gaussian DropConnect (GDC) with and without MC sampling, Bernoulli Dropout (BDO) with and without MC sampling, Gaussian Dropout with and without MC sampling, and spike-and-slab Dropout (SSD) with and without MC sampling.

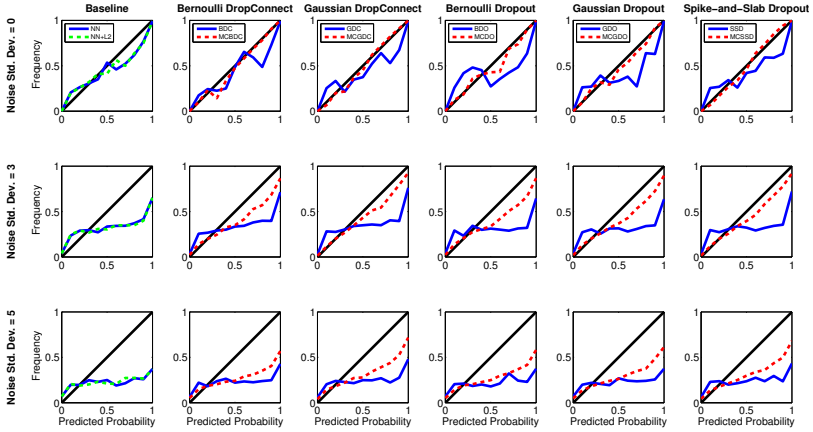

Figure 5: The calibration curves for the MNIST test set with and without Gaussian noise of the softmax outputs of the fully connected networks: NN, NN+L2, Bernoulli DropConnect (BDC) with and without Monte-Carlo (MC) sampling, Gaussian DropConnect (GDC) with and without MC sampling, Bernoulli Dropout (BDO) with and without MC sampling, Gaussian Dropout with and without MC sampling, and spike-and-slab Dropout (SSD) with and without MC sampling.

the data. The uncertainty shows how the probability distribution learned by each DNN is distributed across classes. A low entropy means that the probability mass is primarily located at a few labels and a high entropy means that the probability mass is distributed across many labels. The calibration shows how well the probability distribution learned by the DNN models it's own uncertainty. We evaluated how calibrated a prediction was by the following procedure: (1) We binned test set predictions by predicted probability. (2) We calculated the percentage of predictions in each predicted-probability bin that correctly predicted a target label. Perfect calibration means that targets predicted with probability z are correct in z times 100% of the cases. We therefore (3) calculated the mean squared calibration error (i.e. the mean across bins of the squared deviations between the bin-mean predicted probability and the proportion of correct predictions in that bin). We evaluated

Table 1: MNIST test error for the trained fully connected neural networks with and without Monte-Carlo (MC) sampling using 100 samples.

Method	Mean Error (%)	Std. Dev.
NN	1.68	-
NN+L2	1.64	-
Bernoulli DropConnect	1.33	-
MC Bernoulli DropConnect	1.30	0.04
Gaussian DropConnect	1.24	-
MC Gaussian DropConnect	1.27	0.03
Bernoulli Dropout	1.45	-
MC Bernoulli Dropout	1.42	0.03
Gaussian Dropout	1.36	-
MC Gaussian Dropout	1.37	0.03
Spike-and-Slab Dropout	1.23	–
MC Spike-and-Slab Dropout	1.23	0.03

these three measures for the trained networks on the MNIST and CIFAR test set with noise sampled from Gaussian distributions with varying standard deviations (Figure 3). This tested how well modelled each network's uncertainty was for the test sets and the regions of input space not seen in the training set. For dropout and dropconnect, p was set to 0.5, which corresponds to the best value for regularizing a linear layer Baldi & Sadowski (2013). However in practice, different values for p have been used Srivastava et al. (2014). We found that 0.5 was a robust choice for different networks, measures and sampling methods we used. The one exception were the dropconnect parameter used for spike-and-slab distributions where 0.5 made learning difficult due to the variance during training. Through validation, we found that using larger values spike-and-slab probabilities (0.75 for the fully connected and 0.9 for the convolutional) allowed the networks to fit to the training data better while still maintaining good generalization.

3.1 MNIST

We trained two groups of DNNs, one with a fully connected (FC) architecture and one with a convolutional architecture, on digit classification using the MNIST dataset (LeCun et al., 1998). This set contains 60,000 training images and 10,000 testing images. No data augmentation was used.

3.1.1 Fully Connected Neural Networks

First, we trained DNNs with two FC hidden layers, each with 800 units and ReLU non-linearities. For the L2-regularized network, an L2-coefficient of 1e-5 was used for all weights. For the dropout methods, unit sampling was performed after each FC layer. For the dropconnect methods, every weight was sampled. The classification errors of the FC networks on the MNIST test set are shown in Table 1. Sampling during learning significantly increased accuracy in comparison to the baseline NNs, with the dropconnect-based networks being the most accurate. MC sampling at inference did not significantly increase accuracy. We found that Gaussian dropconnect and spike-and-slab dropout had the best accuracy.

The classification error, uncertainty, and calibration of the learned probability distributions of each FC network for varying levels of noise are shown in Figure 4. While not improving accuracy, MC sampling led to networks that better represent their own uncertainty. As the noise in the test set was increased, the uncertainty of the networks with MC sampling highly increased, especially when compared to networks with no sampling at inference. This resulted in better calibrated FC networks for all levels of noise.

The calibration curves show that sampling only during training, especially when using only dropout, led to overconfidence through placing too much probability mass on the most predicted label (Figure 5). In particular, sampling only during training resulted in under-confidence for low predicted probabilities and over-confidence for high predicted probabilities. By distributing probability mass over several labels, the DNNs that sampled at inference better represented the uncertainty of their predictions.

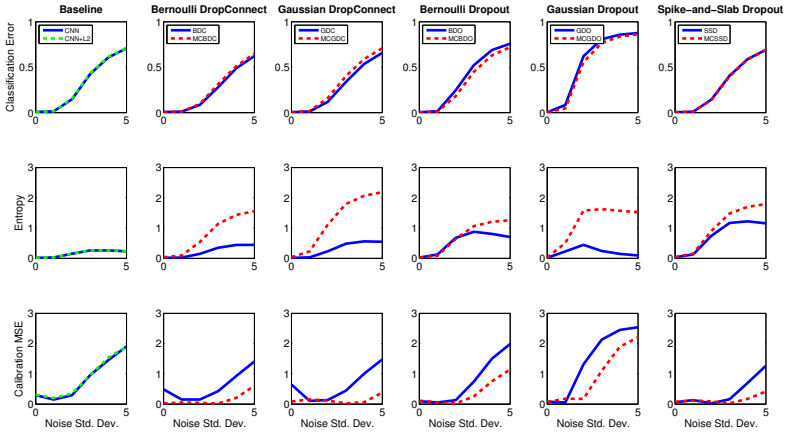

Figure 6: The MNIST test classification error, entropy, and calibration of the predictions of the convolutional networks: CNN, CNN+L2, Bernoulli DropConnect (BDC) with and without Monte-Carlo (MC) sampling, Gaussian DropConnect (GDC) with and without MC sampling, Bernoulli Dropout (BDO) with and without MC sampling, Gaussian Dropout with and without MC sampling, and spike-and-slab Dropout (SSD) with and without MC sampling.

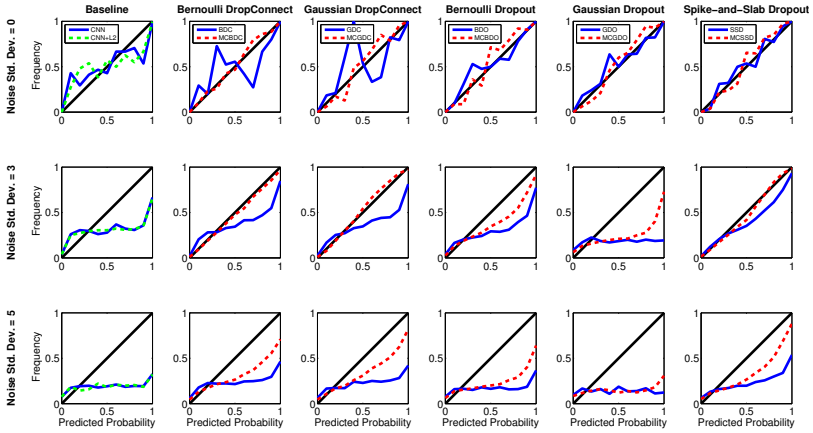

Figure 7: The calibration curves for the MNIST test set with and without Gaussian noise of the softmax outputs of the convolutional networks: CNN, CNN+L2, Bernoulli DropConnect (BDC) with and without Monte-Carlo (MC) sampling, Gaussian DropConnect (GDC) with and without MC sampling, Bernoulli Dropout (BDO) with and without MC sampling, Gaussian Dropout with and without MC sampling, and spike-and-slab Dropout (SSD) with and without MC sampling.

3.1.2 Convolutional Neural Networks

We also trained CNNs on MNIST. Every network had two convolutional layers and a fully-connected layer (See Appendix B for details). For the L2-regularized network, an L2-coefficient of 1e-5 was used for all weights. For Bernoulli and Gaussian dropout, dropout was performed after each convolutional layer and after the FC layer. For Bernoulli and Gaussian dropconnect, every weight was sampled. The classification error of the CNNs on the MNIST test set is shown in Table 2. Sampling during training significantly increased the accuracy for the all of the networks, but especially for the Gaussian dropout network. However, unlike for the FC networks, the dropout-based methods were more accurate than the dropconnect-based methods. Unlike for the FC networks, spike-and-slab had

Table 2: MNIST test error for the trained convolutional neural networks (CNNs) with and without Monte-Carlo (MC) sampling using 100 samples.

Method	Mean Error (%)	Error Std. Dev.
CNN	0.70	-
CNN+L2	0.70	-
Bernoulli DropConnect	0.59	-
MC Bernoulli DropConnect	0.59	0.02
Gaussian DropConnect	0.49	-
MC Gaussian DropConnect	0.49	0.01
Bernoulli Dropout	0.45	-
MC Bernoulli Dropout	0.46	0.01
Gaussian Dropout	0.38	-
MC Gaussian Dropout	0.37	0.01
Spike-and-Slab Dropout	0.43	–
MC Spike-and-Slab Dropout	0.44	0.01

accuracies more similar to Bernoulli dropout, which classified more accurately than Gaussian dropconnect. MC sampling during inference did not significantly increase the accuracy of the networks.

The classification error, uncertainty, and calibration of the learned probability distributions of each network for varying levels of noise are shown in Figure 6. As with the FC networks, MC sampling at inference greatly increased the CNNs' ability to estimate their own uncertainty, particularly for inputs that are different from the training set. MC sampling led to increased entropy as inputs became more noisy, which resulted in better calibration. In particular, this was true of both the Bernoulli and Gaussian dropconnect networks, which very accurately represented their uncertainty even for highly noisy inputs. The spike-and-slab CNN had similar robust calibration. The calibration curves show that not using MC sampling at inference led networks that were under-confident when making low probability predictions and over-confident when making high probability predictions (Figure 7).

3.2 CIFAR-10

We trained large CNNs on natural image classification using the CIFAR-10 dataset, which contains 50,000 training images and 10,000 testing images (Krizhevsky & Hinton, 2009). The CNNs had 13 convolutional layer followed by a fully connected layer (See Appendix B for details). For L2-regularization, an L2-coefficient of 5e-4 was used for all weights. For the dropout networks, was used after each convolutional layer, but before the non-linearities. For the dropconnect networks, all weights were sampled. During training, random horizontal flipping was used. The classification error of the CNNs on the CIFAR-10 test set is shown in Table 3. For each variational distribution, MC sampling significantly increased test accuracy. Also, the that used dropout, including spike-and-slab, had significantly higher accuracies than the networks that only used dropconnect.

The classification error, uncertainty, and calibration of the learned probability distributions of each network for varying levels of noise are shown in Figure 8. One of the major differences between the CIFAR-10 and the MNIST results was that using the layer-wise expectation for dropout did not produce good models, regardless of what variational distribution was used. Instead, the standard test time dropout methods led to relatively inaccurate networks with very high output entropy even when

Table 3: CIFAR-10 test error for the trained convolutional neural networks (CNNs) with and without Monte-Carlo (MC) sampling using 100 samples.

Method	Mean Error (%)	Error Std. Dev.
CNN	19.63	-
CNN+L2	19.44	-
Bernoulli DropConnect	17.64	-
MC Bernoulli DropConnect	17.29	0.05
Gaussian DropConnect	16.00	-
MC Gaussian DropConnect	15.63	0.04
Bernoulli Dropout	37.47	-
MC Bernoulli Dropout	10.19	0.06
Gaussian Dropout	24.10	-
MC Gaussian Dropout	9.29	0.10
Spike-and-Slab Dropout	18.05	–
MC Spike-and-Slab Dropout	10.44	0.03

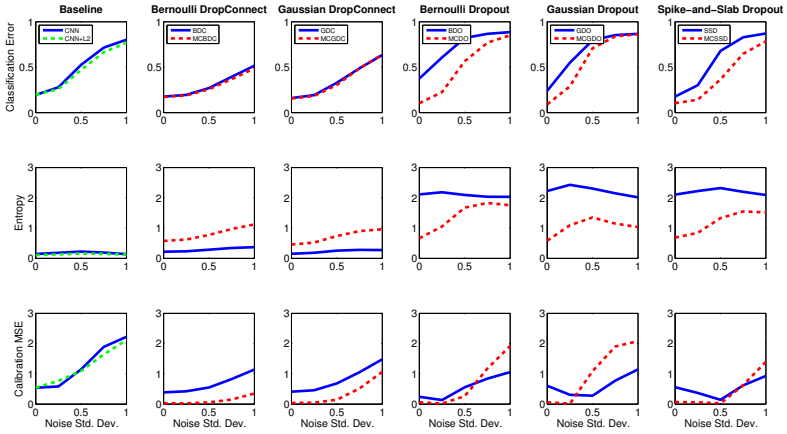

Figure 8: The CIFAR-10 test classification error, entropy, and calibration of the predictions of the convolutional neural networks: CNN, CNN+L2, Bernoulli DropConnect (BDC) with and without Monte-Carlo (MC) sampling, Gaussian DropConnect (GDC) with and without MC sampling, Bernoulli Dropout (BDO) with and without MC sampling, Gaussian Dropout with and without MC sampling, and spike-and-slab Dropout (SSD) with and without MC sampling.

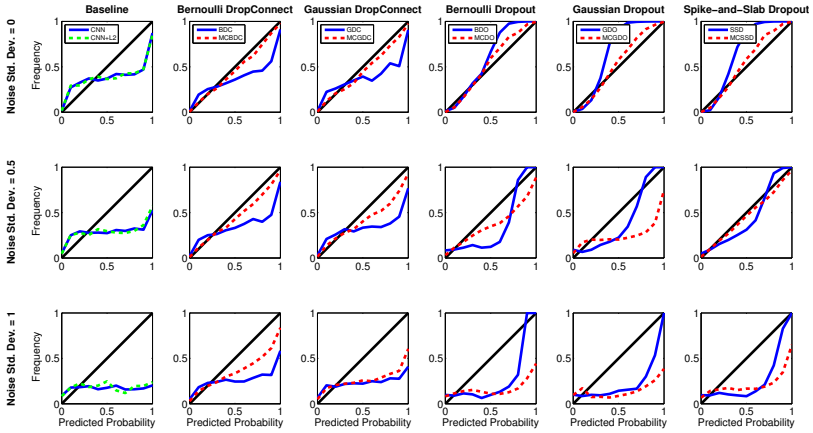

Figure 9: The calibration curves for the CIFAR-10 test set with and without Gaussian noise of the softmax outputs of the convolutional neural networks: CNN, CNN+L2, Bernoulli DropConnect (BDC) with and without Monte-Carlo (MC) sampling, Gaussian DropConnect (GDC) with and without MC sampling, Bernoulli Dropout (BDO) with and without MC sampling, Gaussian Dropout with and without MC sampling, and spike-and-slab Dropout (SSD) with and without MC sampling.

no input noise was used. This agrees with the results reported by Gal & Ghahramani (2015)), who also found that using dropout at every layer can reduce accuracy if MC sampling is not used. However, these results differ from those of Srivastava et al. (2014). In our experience, deeper networks with higher regularization (e.g. Bernoulli dropout probabilities closer to 0.5) result in traditional dropout inference performing significantly worse than MC dropout. As for the MNIST networks, MC sampling at inference overall greatly increased the CIFAR-10 trained CNNs' ability to estimate their own uncertainty when no or little noise was added to the test images.

The classification accuracies and the ability to model uncertainty of the networks with dropconnect sampling were far more robust to noise than the networks with only dropout. However, the MC dropconnect networks are significantly less accurate than the MC dropout networks for the CIFAR-

10 test set when no noise was added. Networks that used traditional dropout inference instead of sampling were consistently uncertain, regardless of the noise. These networks have worse calibration than the MC dropout networks at low levels of noise but better calibration than the MC dropout networks at low levels of noise because they always had high uncertainty. For CIFAR-10, not using MC sampling resulted in networks that were generally over-confident when making predictions (Figure 9). However, this was not true for the non-sampling dropout networks when no input noise was used. In that case, the networks were highly under-confident.

4 DISCUSSION

In this paper, we investigated the ability of MC sampling to improve a DNN's representation of its own uncertainty. We did this by training Bayesian DNNs with either multiplicative masking of the weights (dropconnect) or units (dropout) using Bernoulli, Gaussian, or spike-and-slab sampling. Based on the results, we draw the following main conclusions:

1. Sampling during both learning and inference improved a network's ability to represent its own uncertainty

MC sampling at inference improved the calibration of a network's predictions. Overall, this improvement was particularly large for inputs from outside the training set, which traditional models classified with high confidence despite not being trained on similar inputs.

2. Sampling weights independently led to networks that best represented their own uncertainty

For all the network architectures and datasets tested, using dropconnect sampling at training and inference resulted in the best calibrated networks overall. This was true regardless of whether dropconnect sampling led to the most accurate network. This is in contrast to CNNs with Gaussian dropout sampling, which were significantly the most accurate and also the worst calibrated of the networks with sampling both during training an inference

3. Sampling weights independently led to the most accurate FC networks, but sampling units led to the most accurate CNNs

For the FC networks, using dropconnect, particularly with Gaussian sampling, resulted in the most accurate networks. However, using dropout led to the most accurate CNNs. A potential cause of this is the large correlation in the information contained by adjacent elements in an image, which are often covered by the same convolutional kernel. This could mean that sampling the weights of a kernel does not provide as much regularization as the dropout methods.

4. Sampling using both Bernoulli dropout and Gaussian dropconnect led to accurate and well calibrated networks

Using spike-and-slab dropout, which combines Bernoulli dropout and Gaussian dropconnect, resulted in networks that performed well for all architectures. Spike-and-slab networks had accuracies similar to the Bernoulli dropout or Gaussian dropconnect depending on which performed better for a given architecture and task, Gaussian dropconnect for FC networks and Bernoulli dropout for CNNs. Spike-and-slab networks also were robustly well calibrated similar to all of the other dropconnect methods.

These scalable methods for improving a network's representation of its own uncertainty are widely applicable, since most DNNs already use dropout and getting uncertainty estimates only requires using MC sampling at inference. We plan to further investigate the use of different variational distributions. We also plan to evaluate the use of dropout and dropconnect sampling on large recurrent neural networks. Our results suggest that sampling at inference allows DNNs to efficiently represent their own uncertainty, an essential part of real-world perception and decision making.

ACKNOWLEDGMENTS

We would like to thank Yarin Gal and Sergii Strelchuk for their helpful discussions regarding the manuscript. This research was funded by the Cambridge Commonwealth, European & International Trust, the UK Medical Research Council (Program MC-A060-5PR20), and a European Research Council Starting Grant (ERC-2010-StG 261352).

REFERENCES

Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pp. 2814–2822, 2013.

David Barber and Christopher M Bishop. Ensemble learning in bayesian neural networks. *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES*, 168:215–238, 1998.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1613–1622, 2015.

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015.

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. In *4th International Conference on Learning Representations (ICLR) workshop track*, 2016.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pp. 2348–2356, 2011.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13. ACM, 1993.

Hemant Ishwaran and J Sunil Rao. Spike and slab variable selection: frequentist and bayesian strategies. *Annals of Statistics*, pp. 730–773, 2005.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Chunyuan Li, Andrew Stevens, Changyou Chen, Yunchen Pu, Zhe Gan, and Lawrence Carin. Learning weight uncertainty with stochastic gradient mcmc for shape classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Christos Louizos. Smart regularization of deep architectures. Master's thesis, University of Amsterdam, 2015.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066, 2013.

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 118–126, 2013.

A DERIVATION OF APPROXIMATE SPIKE-AND-SLAB DROPOUT

For Bayesian inference:

$$p(D_{test}|D_{train}) = \int p(D_{test}|W)p(W|D_{train})dW \tag{A.1}$$

Using variational inference:

$$p(D_{test}) = \int p(D_{test}|W)q(W)dW \tag{A.2}$$

Where the variational distribution $q(W)$ is learned by maximizing the log-evidence lower bound:

$$log(p(D_{train})) \geq \int \log(p(D_{train}|W))q(W)dW - KL(q(W)||p(W)) \tag{A.3}$$

For spike-and-slab dropout, as when using Bernoulli dropout, $W = B \circ V$ where $b_{i,*} \sim Bern(p_{do})$, so if we assume independence between the random variables B and V:

$$\begin{aligned} log(p(D_{train})) \geq & \sum_B \int_V \log(p(D_{train}|B,V))q(B)q(V)dVdB \\ & - KL(q(B)||p(B)) - KL(q(V)||p(V)) \end{aligned} \tag{A.4}$$

For a spike-and-slab distribution, each element of V is independently sampled from a Gaussian distribution, $\mathcal{N}(\mu_{v_{i,j}}, \sigma^2_{v_{i,j}})$. As in Gaussian dropconnect, $\sigma^2_{v_{i,j}} = \alpha\mu^2_{v_{i,j}}$. V can be sampled using the "reparameterization trick":

$$v_{i,j} = N(\mu_{v_{i,j}}, \alpha\mu^2_{v_{i,j}}) = g(\mu_{v_{i,j}}, \epsilon_{i,j}) = \mu_{v_{i,j}} + \sqrt{\alpha}\mu_{v_{i,j}}\epsilon_{i,j} \tag{A.5}$$

where $\epsilon \sim N(0,1)$, $\alpha = (1-p_{dc})/p_{dc}$, and p_{dc} is the dropconnect keep probability.

This leads to:

$$\begin{aligned} log(p(D_{train})) \geq & \sum_B \int_\epsilon \log(p(D_{train}|B,V)q(\epsilon)q(B)d\epsilon dB \\ & - KL(q(B)||p(B)) - KL(q(V)||p(V)) \end{aligned} \tag{A.6}$$

This results in the following minimization objective function:

$$\mathcal{L}_{\mu_V} := -\sum_B \int_\epsilon \log(p(D_{train}|B,V))q(\epsilon)q(B)d\epsilon dB + KL(q(B)||p(B)) + KL(q(V)||p(V) \tag{A.7}$$

Using $Bern(p_{do})$ as a prior for B leads to a constant KLD of zero. Using a prior of $\mathcal{N}(0, \sigma_p^2)$ for each element of V leads to the following:

$$KL(q(v_{i,j})||p(v_{i,j})) = \frac{(\mu_{v_{i,j}} - 0)^2}{2\sigma_p^2} + \log\frac{\sigma_p}{\sigma_{v_{i,j}}} + \frac{\sigma^2_{v_{i,j}}}{2\sigma_p^2} - \frac{1}{2} \tag{A.8}$$

By using L2 regularization, we are optimizing a lower-bound of the KLD between $q(V)$ and $\mathcal{N}(0, \lambda^{-1})$ by only matching the first moment (i.e. the mean):

$$\mathcal{L}_{\mu_V} \geq \widetilde{\mathcal{L}}_{\mu_V} := -\sum_B \int_\epsilon \log(p(D_{train}|B,V))q(\epsilon)q(B)d\epsilon dB + \frac{\lambda}{2}\mu_V\mu_V^\intercal \tag{A.9}$$

where μ_V is a vector containing each $\mu_{v_{i,j}}$ and ϵ is a vector containing each $\epsilon_{i,j}$.

Approximating using Monte Carlo integration for learning (Eq. A.10) and inference (Eq. A.11):

$$\widetilde{\mathcal{L}}_{\mu_V} :\approx -\frac{1}{n}\sum_{(B,\epsilon)} \log(p(D_{train}|B,V)) + \frac{\lambda}{2}\mu_V\mu_V^\intercal \tag{A.10}$$

$$p(D_{test}) \approx \frac{1}{n}\sum_{(B,\epsilon)} p(D_{test}|B,V) \tag{A.11}$$

where $b_{i,*} \sim Bern(p_{do})$ and $\epsilon \sim N(0,1)$.

B CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES

B.1 MNIST

Table B.1: The convolutional neural network (CNN) architecture used for MNIST.

Layer	**Kernel Size**	**# Features**	**Stride**	**Non-linearity**
Conv-1	5x5	32	1	ReLU
MaxPool-1	2x2	32	2	Max
Conv-2	5x5	64	1	ReLU
MaxPool-2	2x2	64	2	Max
FC	1500	500	-	ReLU
Linear	500	10	-	-

B.2 CIFAR-10

Table B.2: The convolutional neural network (CNN) architecture used for CIFAR-10.

Layer	**Kernel Size**	**# Features**	**Stride**	**Non-linearity**
Conv-1	3x3	64	1	ReLU
Conv-2	3x3	64	1	ReLU
MaxPool-1	2x2	64	2	Max
Conv-3	3x3	128	1	ReLU
Conv-4	3x3	128	1	ReLU
MaxPool-2	2x2	128	2	Max
Conv-5	3x3	256	1	ReLU
Conv-6	3x3	256	1	ReLU
Conv-7	3x3	256	1	ReLU
MaxPool-3	2x2	256	2	Max
Conv-8	3x3	512	1	ReLU
Conv-9	3x3	512	1	ReLU
Conv-10	3x3	512	1	ReLU
MaxPool-4	2x2	512	2	Max
Conv-11	3x3	512	1	ReLU
Conv-12	3x3	512	1	ReLU
Conv-13	3x3	512	1	ReLU
MaxPool-5	2x2	512	2	Max
FC	512	512	-	ReLU
Linear	512	10	-	-