"Gaussian Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning"

by

Avirup Das (MSc Data Science, Chennai Mathematical Institute)
Abhisek Chatterjee (MSc Statistics, University Of Calcutta)

Understanding Bayesian Neural Networks and model uncertainty using Dropout.

Instructor:
Dr Sushma Kumari

Post-Doctoral Fellow, Chennai Mathematical Institute
ksushma@cmi.ac.in

January, 2022

Gaussian Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning

Avirup Das
MSc Data Science
Chennai Mathematical Institute
avirupdas@cmi.ac.in

Abhisek Chatterjee
BRIDGEi2i Analytics Solutions
MSc Statistics, University Of Calcutta
abhisekchatterjee108@gmail.com

Abstract

Deep Learning has gained a lot of attention in various fields since the development of technology and availability of High-Computation resources. However these tools fail to capture model uncertainty and Bayesian networks come to rescue by providing a mathematical framework for the aforesaid challenge. These networks come with a prohibitive computational costs and there have studies to prove that Neural Networks with Dropouts give us a way to capture model uncertainty without sacrificing the computational complexity or test accuracy. However, repeated sampling of a random subset of input features slows down the training. Our proposal mitigates these issues by using Gaussian approximation instead of Monte Carlo, for Dropouts. This approximation is justified by the Central Limit Theorem (CLT) and provides a significant speedup and more stability apart from capturing model uncertainty. This paper is on our understanding of Bayesian Neural Networks, Modelling Uncertainty in Deep Learning, Gaussian Dropouts and the implementation of our proposal on different data-sets.

1 Introduction

1.1 Brief Overview

Deep Learning is a tool that is being used my researchers from various fields such as Physics, Material Science, Biology, Astronomy [(Anjos et al., 2015), (Baldi et al., 2014), (Bergmann et al., 2013)] just to name a few and these tools have found an extensive use in all these fields, however modelling uncertainty is of crucial importance (Ghahramani, 2015). With more and more shift towards the use of Bayesian techniques for capturing uncertainty, these techniques have found a way into Deep Learning techniques as well [(Herzog and Ostwald, 2013), (Marks and Trafimow, 2015), (Nuzzo, 2014)].

Standard deep learning tools for regression and classification are not designed to capture model uncertainty. There is a common erroneous interpretation of the predictive probabilities given out by a softmax layer, as model confidence. Whereas in fact, a model can be uncertain about the prediction even with a high softmax output (fig. 1). Passing a point estimate of a function (solid line in fig. 1a) through a softmax (solid line in fig. 1b) results in extrapolations which are essentially unjustified high confidence for points that are far from the training data (for eg: x^* would be classified as class 1 almost certainly). However, if the distribution (shaded area in fig. 1a) is passed through a softmax (shaded area in fig. 1b), it reflects a better idea of classification uncertainty associated with the points that are far from the training data.

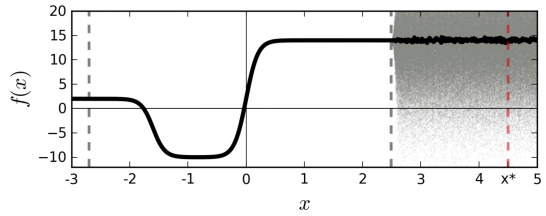

Figure 1a: Arbitrary function $f(x)$ as a function of data x (softmax input)

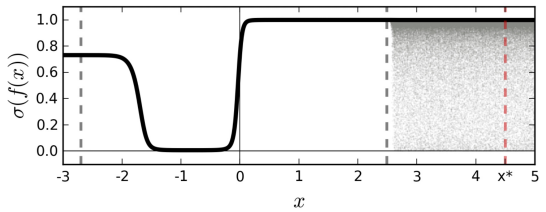

Figure 1b: $\sigma\left(f(x)\right)$ as a function of data x (softmax output)

Model uncertainty is important for deep learning practitioners as well. If we get the model confidence then methods can be devised on treatment of uncertain inputs and special cases in a more explicit fashion. For example, let us consider a language model which can generate transcripts from audio files. Now if the model has a low confidence for a particular audio file then the generated transcript can be sent to a team of manual transcribers for review and correction.

A Bayesian Neural Network (BNN) is defined slightly differently across the literature but a common definition is that a BNN is a stochastic artificial neural network (ANN) trained using Bayesian Inference. **Stochastic neural networks** are a type of ANNs built by using stochastic

components in the network either using a stochastic activation (fig. 2b) or by using stochastic weights (fig. 2c) in order to simulate multiple possible models θ with their associated probability distribution $p(\theta)$. Thus they are considered as a special case of **ensemble learning** (Zhou, 2012), the main motivation of which comes from the observation that aggregating the predictions of a large number of independent predictors (or classifiers) which are average performing, can provide better predictions than a single strong predictor.

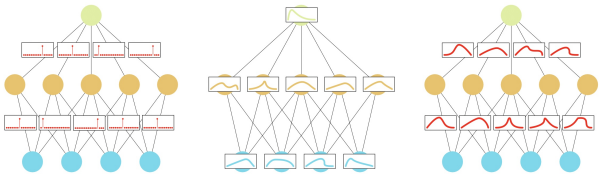

Figure 2: (a) Point estimate NN, (b) Stochastic network with distribution over activations, (c) Stochastic network with distribution over weights

A stochastic neural network compares the predictions of multiple sampled model parameterizations θ. If the different models agree then the uncertainty is low, and if they disagree then the uncertainty is high. This can be summarised as:

$$\theta \sim p(\theta)$$
$$y = \Phi_\theta(x) + \epsilon \tag{1}$$

where ϵ represent random noise to account for the fact that the function Φ is only an approximation. Let us denote the training set by D, the training inputs by D_x and the training labels by D_y. By Bayes theorem, and enforcing independence between the model parameters and the inputs, the Bayesian posterior can be written as,

$$p(\theta|D) = \frac{p(D_y|Dx,\theta)\ p(\theta}{\int_\theta p(D_y|D_x,\theta^*)\ p(\theta^*)\ d\theta^*} \propto p(D_y|D_x,\theta)\ p(\theta) \tag{2}$$

While using BNNs we are more interested in the marginal probability distribution $p(y|x,D)$ which essentially quantifies the model's uncertainty on its prediction. The advantages of using BNN's are as follows:

1. Bayesian methods provide a mathematical framework to quantify uncertainty in deep learning since they have better calibration than classical neural networks, so their uncertainty is more consistent with the observed errors.

2. BNN allows us to distinguish between **epistemic uncertainty** $p(\theta|D)$ and the **aleatoric uncertainty** $p(y|x,\theta)$ [(Der Kiureghian and Ditlevsen, 2009), (Depeweg et al., 2017)]. This makes these networks capable of learning from a comparatively small dataset without the risk of over-fitting. During prediction, out-of-training distribution points will have a high epistemic uncertainty.

3. According to the *No Free-Lunch Theorem* for machine learning (Wolpert, 1996), one can say that any supervised learning algorithm includes some implicit prior. Bayesian methods make these priors explicit which in practice act as soft constraints, like regularization, or via data transformations like data augmentation.

However, these models come with a prohibitive computation cost. To represent uncertainty, the number of parameters is doubled in these networks for the same network size. Also, they require more time to converge and do not improve much on existing techniques. The authors of (Gal and Ghahramani, 2015) gave a complete theoretical treatment of the link between Gaussian processes and dropout, and developed the tools to represent uncertainty in deep learning. But with the effectiveness of dropout in preventive feature co-adaptation or averaging over many neural network with shared weights, sampling or training multiple models slows down the training significantly. Moreover, with a dropout rate of p, the proportion of data still not seen after n passes is p^n (eg, in order to see 95% of the data with a dropout rate of 0.5, there has to be 5 passes of the data). Now if the data is not highly redundant and if we make the data only partially observable at random, then the task gets more complicated and subsequently training efficiency will reduce. The authors of (Wang and Manning, 2013) proposed a Gaussian approximation for the task, which mitigates the problems and drawbacks of sampling.

1.2 Objective

In this paper we propose the use of Gaussian Dropouts for Bayesian approximation and obtain the model uncertainty. We will try to provide a theoretical justification for our proposal, but most importantly we assess various network architectures on regression and classification tasks, using datasets like MNIST (use the LeNet network with tanh and softmax non-linearities) and CO_2 dataset (use MLP networks with similar non-linearities) and report the input-output scatter for the different dropout types. We would also report the training and inference time for these models using both normal dropout and Gaussian dropouts. We will show a considerable improvement in predictive log-likelihood and RMSE compared to existing state-of-the-art methods along with a considerable drop in the time required for training and inference.

2 Dropout as a Bayesian Approximation

The authors of (Gal and Ghahramani, 2015) have shown already that a neural network with arbitrary depth and non-linearities, with dropout applied before each deep layer, is equivalent to the probabilistic deep Gaussian Process (Damianou and Lawrence, 2013) marginalised over its covariance function. It has also been stressed that no simplifying assumptions were made on the use of dropout in the literature, and that the results that were

derived are applicable to any architecture that makes use of dropout.

Let $\hat{y}$ be the output of a NN model with L layers and a loss function $E(,)$ such as the softmax loss or the Euclidean loss (square loss). We denote by W_i the NN's weight matrices of dimensions $K_i \times K_{i-1}$, and by b_i the bias vectors of dimensions K_i for each layer $i = 1, \ldots, L$. We denote by y_i the observed output corresponding to input x_i for $1 \leq i \leq N$ data points, and the input and output sets as **X,Y**. During NN optimisation a regularisation term is often added. We often use L_2 regularisation weighted by some weight decay λ, resulting in a minimisation objective (often referred to as cost),

$$\mathcal{L}_{\text{dropout}} = \frac{1}{N} \sum_{i=1}^{N} E(y_i, \hat{y}_i) + \lambda \sum_{i=1}^{L} \left(\|W_i\|_2^2 + \|b_i\|_2^2 \right) \quad (3)$$

With dropout, we sample binary variables for every input point and for every network unit in each layer (apart from the last one). Each binary variable takes value 1 with probability p_i for layer i. A unit is dropped (i.e. its value is set to zero) for a given input if its corresponding binary variable takes value 0. We use the same values in the backward pass propagating the derivatives to the parameters. In comparison to the non-probabilistic NN, the deep Gaussian process is a powerful tool in statistics that allows us to model distributions over functions. Assume we are given a covariance function of the form

$$K(x, y) = \int p(w)p(b)\sigma\left(w^T x + b\right)\sigma\left(w^T y + b\right)\ dw\ db \quad (4)$$

with some element-wise non-linearity $\sigma()$ and distributions $p(w), p(b)$. The authors of (Gal and Ghahramani, 2015) have proved that a deep Gaussian process with L layers and covariance function $K(x, y)$ can be approximated by placing a variational distribution over each component of a spectral decomposition of the GPs' covariance functions. This spectral decomposition maps each layer of the deep GP to a layer of explicitly represented hidden units.

Let W_i be a (now random) matrix of dimensions $K_i \times K_{i-1}$ for each layer i, and write $\omega = \{W_i\}_{i=1}^{L}$. A priori, we let each row of W_i distribute according to the $p(w)$ above. In addition, assume vectors mi of dimensions K_i for each GP layer. The predictive probability of the deep GP model (integrated w.r.t. the finite rank covariance function parameters ω) given some precision parameter $\tau > 0$ can be parameterised as

$$p(y|x, X, Y) = \int p(y|x, \omega)p(\omega|X, Y)d\omega \quad (5)$$

$$p(y|x, \omega) = \mathcal{N}(y; \hat{y}(x, \omega); \tau^{-1} I_D)$$

$$\hat{y}\left(x, \omega = \{W_1, \ldots, W_L\}\right) = \sqrt{\frac{1}{K_L}} W_L \sigma\left(\ldots \sqrt{\frac{1}{K_1}} W_2 \sigma\left(W_1 x + m_1\right) \ldots\right)$$

The posterior distribution $p(\omega|X, Y)$ in equation (5) is intractable. We use $q(\omega)$, a distribution over matrices whose columns are randomly set to zero, to approximate the intractable posterior. We define $q(\omega)$ as:

$$W_i = M_i.\text{diag}\left(\left[z_{i,j}\right]_{j=1}^{K_i}\right)$$
$$z_{i,j} \sim \text{Bernoulli}(p_i)\ \forall\ i = 1, \ldots, L,\ j = 1, \ldots, K$$

given some probabilities pi and matrices M_i as variational parameters. The binary variable $z_{i,j} = 0$ corresponds then to unit j in layer $i - 1$ being dropped out as an input to layer i. The variational distribution $q(\omega)$ is highly multi-modal, inducing strong joint correlations over the rows of the matrices W_i (which correspond to the frequencies in the sparse spectrum GP approximation).

We minimise the KL divergence between the approximate posterior $q(\omega)$ above and the posterior of the full deep GP, $p(\omega|X, Y)$. This KL is our minimisation objective

$$-\int q(\omega)\ log\ p\left(Y|X, \omega\right) d\omega + KL\left(q\left(\omega\right)||p(\omega)\right) \quad (6)$$

We rewrite the first term as the following sum

$$-\sum_{n=1}^{N} \int q(\omega)\ log\ p\left(y_n|x_n, \omega\right)\ d\omega$$

and approximate each term in the sum by Monte Carlo integration with a single sample $\hat{\omega}_n \sim q(\omega)$ to get an unbiased estimate- $log\ p(y_n|x_n, \hat{\omega}_n)$. We further approximate the second term in eq. (6) and obtain $\sum_{i=1}^{L} \left(\frac{p_i l^2}{2} \|M_i\|_2^2 + \frac{l^2}{2} \|m_i\|_2^2 \right)$ with prior length-scale l. Given model precision τ we scale the result by the constant $1/\tau N$ to obtain the objective:

$$\mathcal{L}_{\text{GP-MC}} \propto \frac{1}{N} \sum_{n=1}^{N} \frac{-log\ p(y_n|x_n, \hat{\omega}_n)}{\tau} + \sum_{i=1}^{L} \left(\frac{p_i l^2}{2\tau N} \|M_i\|_2^2 + \frac{l^2}{2\tau N} \|m_i\|_2^2 \right) \quad (7)$$

Setting $E\left(y_n, \hat{y}\left(x_n, \hat{\omega}_n\right)\right) = -\log\ p\left(y_n|x_n, \hat{\omega}_n\right)/\tau$ we recover equation (3) for an appropriate setting of the precise hyper-parameter τ and length-scale l. The sampled $\hat{\omega}_n$ result in realisations from the Bernoulli distribution $z_{i,j}^n$ equivalent to the binary variables in the dropout case.

3 Gaussian Dropout

3.1 Objective function

Let us illustrate the idea with logistic regression (LR) given the training vector x and label $y \in \{0, 1\}$. To train an LR with dropout on data with dimension m, we first sample $z_i \sim \text{Bernoulli}(p_i)$ for $i = 1, \ldots, m$. Here p_i

is the probability of not dropping out the input x_i. After sampling $z = \{z_i\}\,,\ \forall\ i = 1, \ldots, m$ we can compute the stochastic gradient descent (sgd) update as follows:

$$\Delta w = \left(y - \sigma\left(w^T D_z x\right)\right) D_z x$$

where $D_z = \text{diag}(z) \in \mathbb{R}^{m \times m}$ and $\sigma = 1/\left(1 + e^{-x}\right)$ is the logistic function. The update rule, applied over the training data for multiple passes, can be seen as a Monte Carlo approximation to the following gradient:

$$\Delta \bar{w} = E_{z; z_i \sim \text{Bernoulli}(p_i)}\left[\left(y - \sigma\left(w^T D_z x\right)\right) D_z x\right] \quad (8)$$

The objective function with the above gradient is the expected conditional log-likelihood of the label given the data with dropped out dimensions indicated by z, for $y \sim \text{Bernoulli}\left(\sigma\left(w^T D_z x\right)\right)$. This is the implied objective function for dropout training:

$$\begin{aligned} &L\left(w\right) = E_z\left[log\left(p\left(y | D_z x; w\right)\right)\right] \\ &= E_z[y\ log\left(\sigma(w^T D_z x)\right) + (1-y)\ log\left(1 - \sigma(w^T D_z x)\right)] \end{aligned} \quad (9)$$

Note that we still have a convex optimisation problem since we are just taking an expectation, provided that the negative log-likelihood is convex. Evaluating the expectation in (8) naively by summing over all possible z, has a complexity of $O(2^m m)$. Rather than directly computing the expectation with respect to z, we propose a variable transformation that allows us to approximately compute the expectation with respect to a simple random variable $Y \in \mathbb{R}$, instead of $z \in \{0, 1\}^m$. In the next subsection, we describe an efficient $O(m)$ approximation that is accurate for machine learning applications where $w_i x_i$ usually come from a unimodal or bounded distribution.

3.2 Gaussian approximation

We make the observation that evaluating the objective function $L(w)$ involves taking the expectation with respect to the variable $Y(z) = w^T D_z x = \sum_i^m w_i x_i z_i$, a weighted sum of Bernoulli random variables. For most machine learning problems, $\{w_i\}$ typically forms a unimodal distribution centered at 0, x_i is either unimodal or in a fixed interval. In this case, Y can be well approximated by a normal distribution even for relatively low dimensional data with $m = 10$. More technically, the Lyapunov condition is generally satisfied for a weighted sum of Bernoulli random variables of the form Y that are weighted by real data (Lehmann, 2004).

Then, Lyapunov's central limit theorem states that $Y(z)$ tends to a Gaussian distribution as $m \to \infty$ (figure 3). The authors of (Wang and Manning, 2013) have empirically verified that the approximation is good for typical datasets of moderate dimensions, except when a couple of dimensions dominate all others. Finally, let S be the approximating Gaussian ($Y \xrightarrow{d} S$)

$$S = E_z\left[Y\left(z\right)\right] + \sqrt{Var\left[Y(z)\right]}\,\epsilon = \mu_S + \sigma_S \epsilon \quad (10)$$

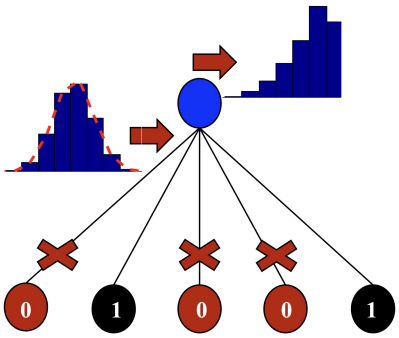

Figure 3: *Illustration of the fast dropout idea:* The numbers at the bottom are the dropout indicator variables $z_1, \ldots, z_5$. As z is repeatedly sampled, the resulting inputs to the top unit are close to being normally distributed.

where $\epsilon \sim \mathcal{N}(0, 1),\ E_z\left[Y(z)\right] = \sum_{i=1}^m p_i w_i x_i$ and $Var\left[Y(z)\right] = \sum_{i=1}^m p_i\left(1 - p_i\right)\left(w_i x_i\right)^2$

3.3 Fast gradient computation by sampling from Gaussian

Given good convergence, we note that drawing samples of the approximating Gaussian S of $Y(z)$, a constant time operation, is much cheaper than drawing samples of $Y(z)$ directly, which takes $O(m)$. This effect is very significant for high dimensional datasets. So we can already approximate the objective function (9) m-times faster by sampling from S instead of $Y(z)$. Approximating the gradient introduces a complication when using samples from the Gaussian. The gradient (8) involves not only $Y(z) \xrightarrow{d} S$, but also $D_z x$ directly:

$$\nabla L(w) = E_z\left[\left(y - \sigma(Y(z))\right) D_z x\right] \quad (11)$$

Let $f(Y(z)) = y - \sigma(Y(z))$ and let $g(z) = D_z x$. Naively approximating $E_z[f(Y(z))g(z)]$ by either $E_S[f(S)]E_z[g(z)]$, or worse, by $f(E_s[S])E_z[g(z)]$ works poorly in terms of both approximation error and final performance. Note that $g(z)$ is a linear function and therefore $E_z[g(z)] = g(E_z[z]) = \text{diag}(p)x$. A good way to approximate (11) is by analytically taking the expectation with respect to z_i and then using a linear approximation to the conditional expectation. More precisely, consider dimension i of the gradient:

$$\begin{aligned} \frac{\partial L(w)}{\partial w_i} &= E_z[f(Y(z)) x_i z_i] \\ &= \sum_{z_i \in \{0,1\}} p(z_i) z_i x_i E_{z_{-i}|z_i}[f(Y(z))] \\ &= p(z_i = 1) x_i E_{z_{-i}|z_i = 1}[f(Y(z))] \end{aligned}$$

$$\frac{\partial L(w)}{\partial w_i} \approx p_i x_i \Big(E_{S\sim N(\mu_s,\sigma_S^2)}[f(S)] + \Delta\mu_i \frac{\partial E_{S\sim N(\mu_s,\sigma_S^2)}[f(S)]}{\partial \mu}\Big|_{\mu=\mu_S} + \Delta\sigma_i^2 \frac{\partial E_{S\sim N(\mu_s,\sigma_S^2)}[f(S)]}{\partial \sigma^2}\Big|_{\sigma^2=\sigma_S^2} \Big)$$

Thus we get,

$$\frac{\partial L(w)}{\partial w_i} = p_i x_i(\alpha(\mu_S,\sigma_S^2)+\Delta\mu_i\beta(\mu_S,\sigma_S^2)+\Delta\sigma_i^2\gamma(\mu_S,\sigma_S^2)) \tag{12}$$

where z_{-i} is the collection of all other $z's$ except z_i, μ_S, σ_S is defined in (10), $\Delta\mu_i = (1-p_i)x_i w_i, \Delta\sigma_i^2 = -p_i(1-p_i)x_i^2 w_i^2$ are the changes in μ_S, σ_S^2 due to conditioning on z_i. Note that the partial derivatives as well as $E_{S\sim N(\mu_s,\sigma_S^2)}[f(S)]$ only need to be computed once per training case, since they are independent of i. α, β, γ can be computed by drawing K samples from S, taking time $O(K)$ (Note: K samples from $Y(z)$ took $O(mK)$ time). So concretely we have,

$$\alpha(\mu,\sigma^2) = y - E_{S\sim N(\mu_s,\sigma_S^2)}\left[\frac{1}{1+e^{-\mu-\sigma_S S}}\right]$$
$$\beta(\mu,\sigma^2) = \frac{\partial\alpha(\mu,\sigma^2)}{\partial\mu}, \quad \gamma(\mu,\sigma^2) = \frac{\partial\alpha(\mu,\sigma^2)}{\partial\sigma^2}$$

The above quantities can be computed by differentiating inside the expectation.

4 Methodology

Concretely, instead of using a Bernoulli mask for dropout to obtain a Bayesian approximation for predictions, we propose the use of a mask whose elements are essentially Gaussian random variables (backed by the law of Large Numbers). Using Gaussian masks does not change anything concerning the relevance of using normal Dropout against overfitting due to co-adaptation and/or the predictive capacity of the neurons in the network, or even obtaining the model uncertainty. But it changes everything in terms of execution time required for the training phase compared to the previous methods. Logically, by omitting at each iteration neurons with a dropout, those omitted on an iteration are not updated during the backpropagation. These neurons do not exist. So the training phase is slowed down. On the other hand, by using a Gaussian Dropout method, all the neurons are exposed at each iteration and for each training sample. This avoids the slowdown while providing the ensembling capability of the model during inference. Figure 4 illustrates the idea for a 4-neuron layer with a dropout rate of 0.5.

4.1 Implementation

We run our experiments using *Tensorflow 2.6* using which we implemented a custom Dropout layer which obtains keeps the dropout masks turned on during both training and inference using the *Keras backend* (contrary to a normal dropout which is applied only during training). We also implemented a custom layer for the *Gaussian dropout* by obtaining the standard deviation as $\sigma = \sqrt{\frac{r}{1-r}}$ where r is the dropout-rate. A random sample from a p-variate Gaussian distribution is then obtained with mean 1 and standard deviation σ, where p is the shape of the input to this custom Gaussian dropout. The activity of this layer during training and inference phase is again enabled using the Keras backend.

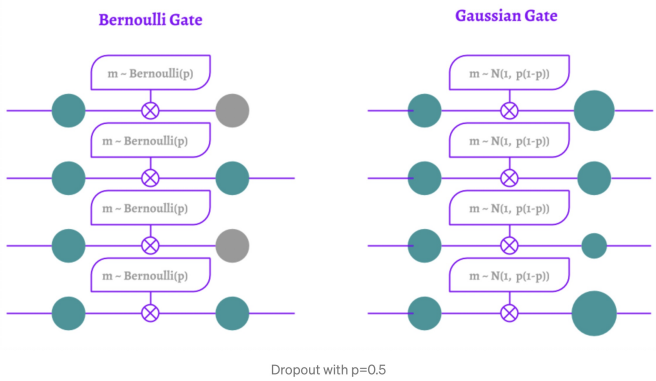

Figure 4: *Illustration of the Gaussian Dropout:* The dropout rate is 0.5, the green nodes represent the active neurons and the grey ones represent those blocked during the dropout

4.2 Experiments

We next perform an extensive assessment of the properties of the uncertainty estimates obtained from dropout NNs and Convolutional networks on the tasks of regression and classification. We compare the uncertainty obtained from different model architectures and non-linearities, on tasks of extrapolation, and show that model uncertainty is important for classification tasks using MNIST (Gal and Ghahramani, 2015) have shown considerable improvement in predic- tive log-likelihood and RMSE compared to existing state-of-the-art methods. We show the improved performance of the *Gaussian Dropouts* by reporting the training time.

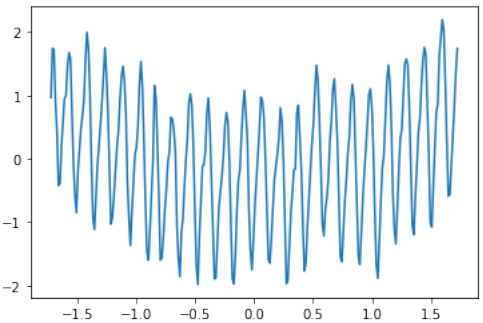

Figure 5: A simple trend of the CO2 emmissions

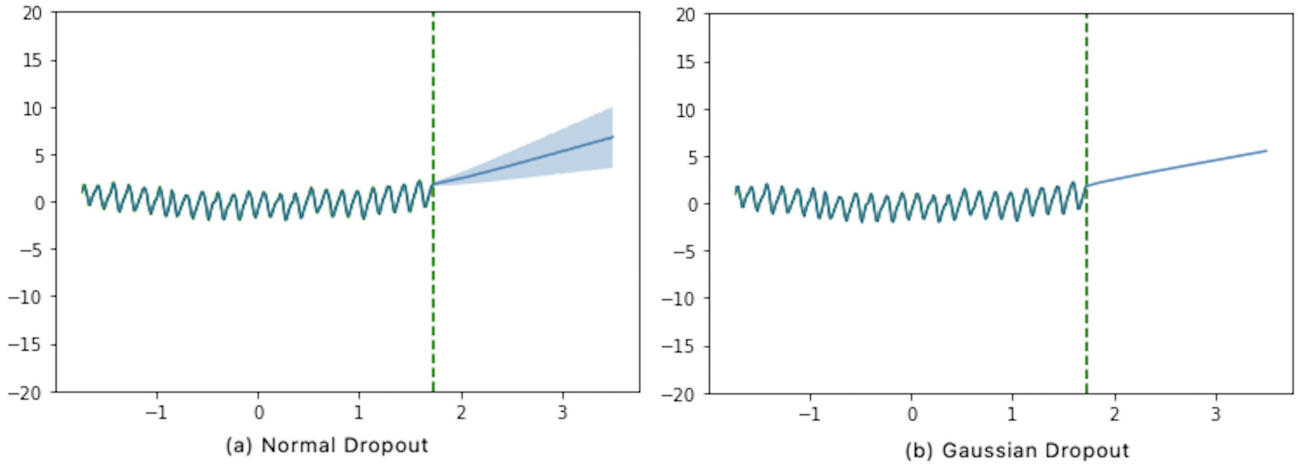

Figure 6: *Predictive mean and uncertainties on the Mauna Loa CO_2 concentration dataset, for different dropouts* The left of the dashed green line is the training data and to the right is the testing data for extrapolation. The blue line is the predictive mean for each input point.

4.2.1 Regression Experiment on CO2 Dataset

We use a subset of the atmospheric CO2 concentrations dataset derived from in situ air samples collected at Mauna Loa Observatory, Hawaii (Keeling et al., 2010) (referred to as *CO2*). A simple trend regarding the data is shown in Figure 5. We standardize and pre-process the data appropriately and trained the model for the dataset using NNs with several hidden layers with two dropout layers and 1024 hidden units. ReLU activation was used in each network (for hidden layers), and dropout probabilities of 0.1 and a learning rate of 0.0001 were found to be suitable for dataset. We also used an *Adam Optimizer* with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for training the aforesaid model.

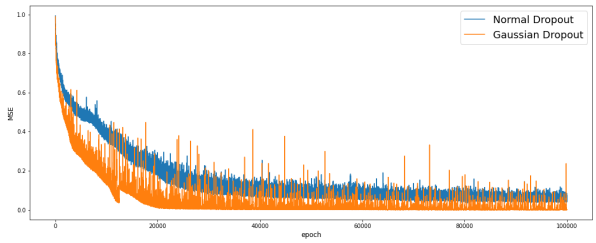

Figure 7: Training History for model trained with Normal and Gaussian Dropout

The model was individually trained with both Normal and Gaussian Dropout for 10^5 epochs. The model was trained on 70% of the dataset and was tested on the entire dataset. Figure 6 shows the results for both types of dropouts with ReLU activations, the shade of blue represents model uncertainty. For the Normal dropout case, we see that the uncertainty is increasing for points which are away from the seen data. During inference time 5000 forward iterations were used, and for Gaussian Dropout we don't see any shade of blue in the predictions implying that the model uncertainty was decreased by a large extent. The difference between the training times for these dropouts were significantly high. The wall-time for training the Normal Dropout for 10^5 epochs was *1hr 16mins 41s* whereas for the same training size and epochs, the Gaussian Dropout took *1hr 10mins 23s*, which may not be a significant difference if one considers absolute figures, but the difference can be seen once the rate of convergence to the global minima of the risk-function is considered. From Figure 7 (training history for both networks) it is evident that the network with the Gaussian Dropout converged faster and to a lower minima (MSE: 0.0005 - 0.001) in comparison to the network with the normal dropout (MSE: 0.04 - 0.06)

4.2.2 Classification Experiment on MNIST Dataset

To assess model classification confidence in a realistic example we test a convolutional neural network trained on the full MNIST dataset (LeCun et al., 2010). We chose the LeNet architecture (two convolutional blocks: each having a convolutional layer and a max-pooling layer; followed by a fully-connected layer with Dropout) (Lecun et al., 1998) with three dropout layers applied after each of the two convolutional layers and one before the last fully connected layer. We trained the model (with a dropout rate of 0.5) for 20 epochs with a learning rate of 0.001 using an *Adam Optimizer* with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

We evaluated the trained model on a continuously rotated image of the digit 1 (shown on the X axis of Figure 8). We scatter 5000 stochastic forward passes of the softmax input (the output from the last fully connected layer), as well as that of the softmax output for the classes 1, 5 and 7. We first trained the model using Standard dropout. For the 12 images, the model predicts classes [1 1 1 1 3 2 1 7 7 7 7 7]. The plots show the softmax input value and softmax output value for the 3

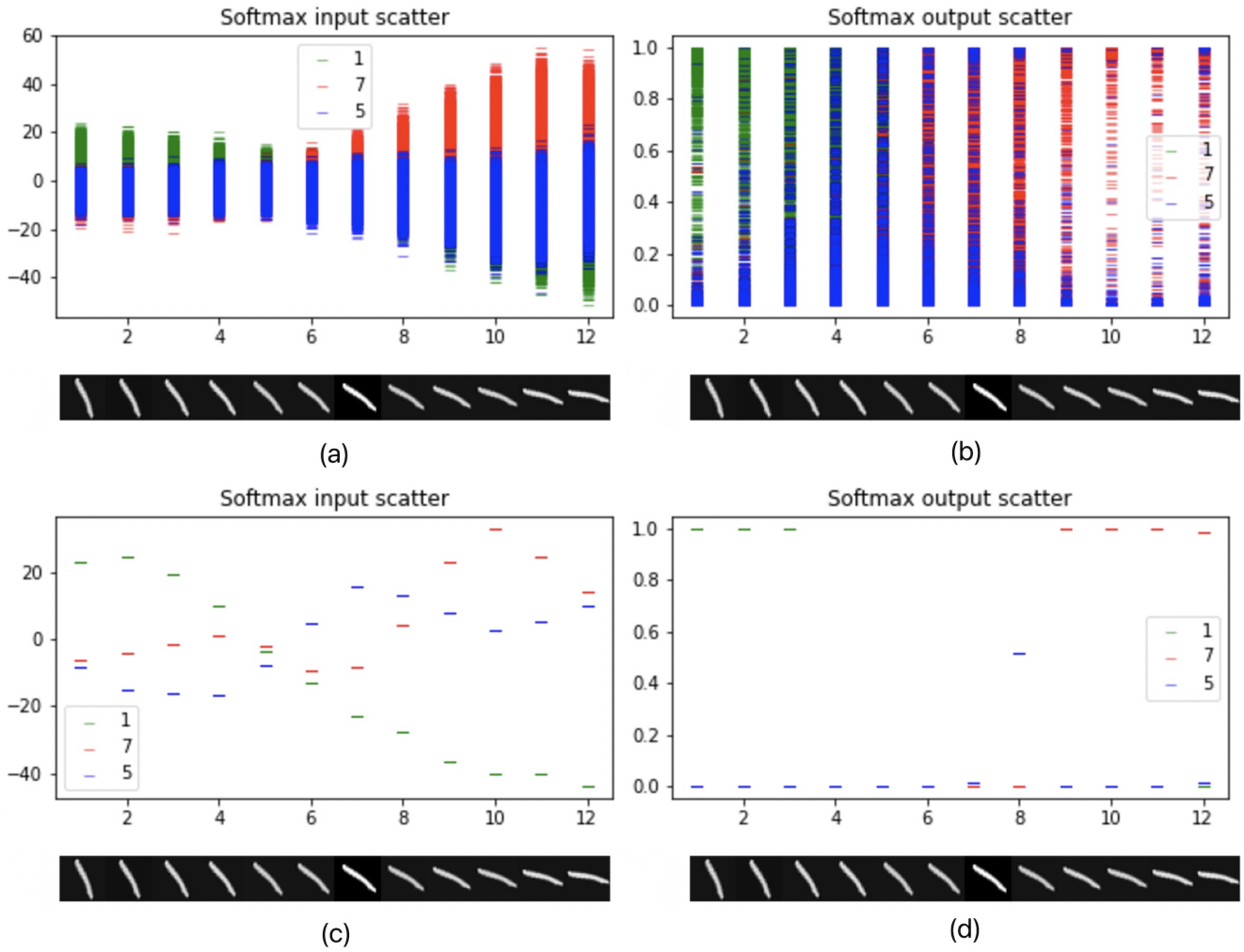

Figure 8: A scatter of 5000 forward passes of the softmax input and output for LeNet with Standard dropout (a and b) and with Gaussian dropout (c and d) . On the X axis is a rotated image of the digit 1. The inputs are classified as digit 1 for images towards the left and 7 towards the right, the model uncertainty is very high. (best viewed in colour).

digits 1, 5 and 7.

Although, none of the pictures were classified as 5, we keep the softmax values of 5 in the plot following the experiment in (Gal and Ghahramani, 2015). When the softmax input for a class is larger than that of all other classes (class 1 for the first 4 images, class 3 for the next image, class 2 for the sixth image, class 1 for the seventh image and 7 for the rest in Figure 8 (a and b)), the model predicts the corresponding class. Looking at the softmax output values, if the uncertainty envelope or the colour scatter of a class is dominating the bar in the higher end (for example the left most image or the right most image) then the input is classified with high confidence. On the other hand, if the uncertainty envelope intersects that of other classes and one single colour scatter does not dominate the entire bar (such as in the case of the images in the middle), then even though the softmax output can be arbitrarily high (as far as 1 if the mean is far from the means of the other classes), the softmax output uncertainty can be as large as the entire space. This signifies the model's uncertainty in its softmax output value i.e. in the prediction. In this scenario it would not be reasonable to return the classes for the middle images when their uncertainty is so high. One would expect the model to ask an external annotator for a label for this input. Model uncertainty in such cases can be quantified by looking at the entropy or variation ratios of the model prediction.

With the same model architecture and sequence of layers, we train another model to classify the same set of 12 rotated images of the digit 1, this time using Gaussian dropout. For the 12 images, the model predicts classes [1 1 1 4 4 4 4 5 7 7 7 7]. We again scatter 5000 stochastic forward passes of the softmax input, as well as of the softmax output for the classes 1, 5 and 7. The plots in Figure 8 (c and d) show the softmax input value and softmax output value for the 3 digits 1, 5 and 7. Looking at the softmax output values, we see that the colour scatters are very small and the softmax output values of the classified class is close to 1 for most of the images

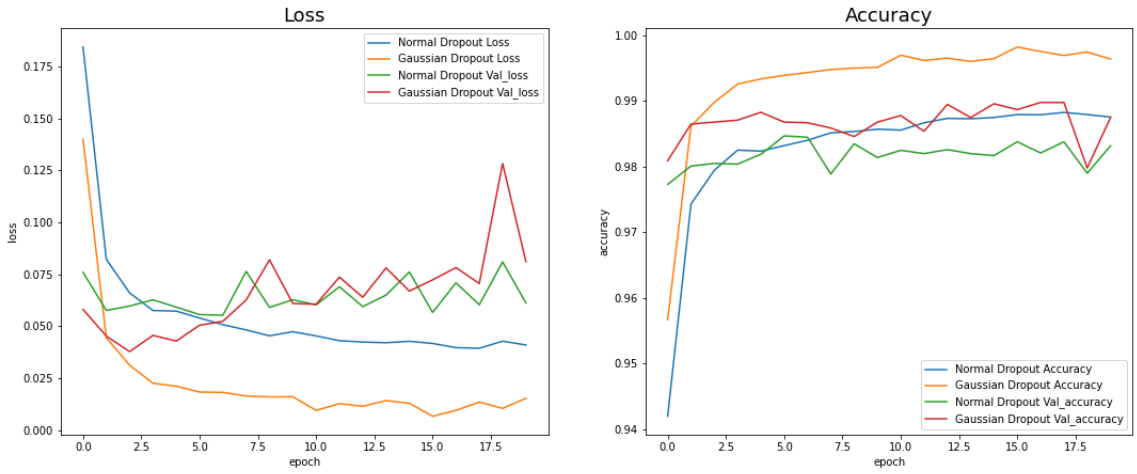

Figure 9: Training history of the LeNet classifier trained on MNIST data using Normal and Gaussian Dropout. The left image shows the loss (Categorical Crossentropy) and the right image shows the accuracy. The significance of the different colors have been indicated in the legends of the individual plots.

towards the left and the right. For the 8th image, the softmax output value for the class 5 is around 0.5, although this is the highest softmax value among all the classes for that input. We can thus say that the predictive confidence for this model with Gaussian dropout is much more than the model with Normal dropout. So, using Gaussian dropout, we could increase predictive confidence and decrease uncertainty to a large extent.

Here again we see a faster training in case of Gaussian dropouts. The wall-time observed for training the LeNet Classifier with Normal dropout and Gaussian dropout are *4min 13s* and *3min 30s* respectively. From Figure 9 we conclude again that the convergence to global minima for the risk function is faster for Gaussian dropouts while preserving the ability to estimate uncertainty. The result is the observation that the classifier with Gaussian dropout converges at around 3 epochs (after which it starts to overfit) as opposed to the classifier with Normal dropout which converges at around 20 epochs.

4.2.3 Classification Experiment on Monkey Dataset

This is a full-blown experiment where we assess the performance of a realistically complex model for classifying monkeys from a variety of images (Dataset, 2017). The broad objectives of the experiment are as follows:

- We train the model for classifying the images into the first 7 classes (namely *Mantled Howler, Patlas Monkey, Bald Uakari, Japanese Macaque, Pygmy Marmoset, White-headed Capuchin* and *Silvery Marmoset*)
- Create two instances for the model, one with Gaussian Dropout and the other with Normal Dropout.
- Check model performance and uncertainty for both dropout-variants on Test-data (191 images belonging to 7 classes the model was trained on, these images were not part of the training set)
- Check model performance and uncertainty for both dropout-variants on Out-of-Distribution Data (81 images belonging to 3 classes, the models are not trained to classify images into these classes)

The model architecture used is given below:

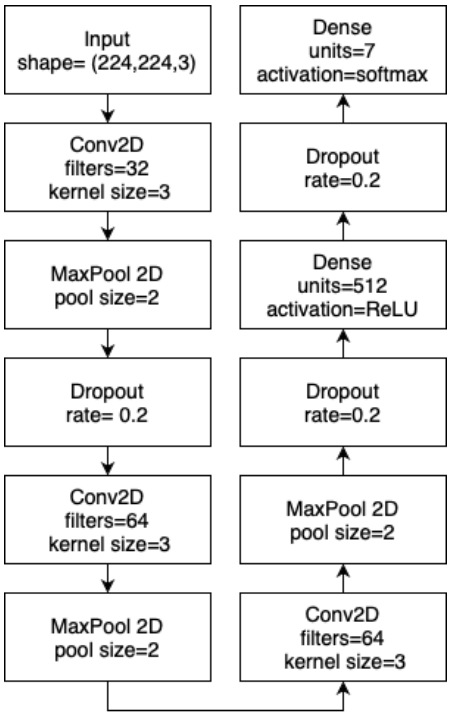

Figure 10: Model architecture for monkey classifier

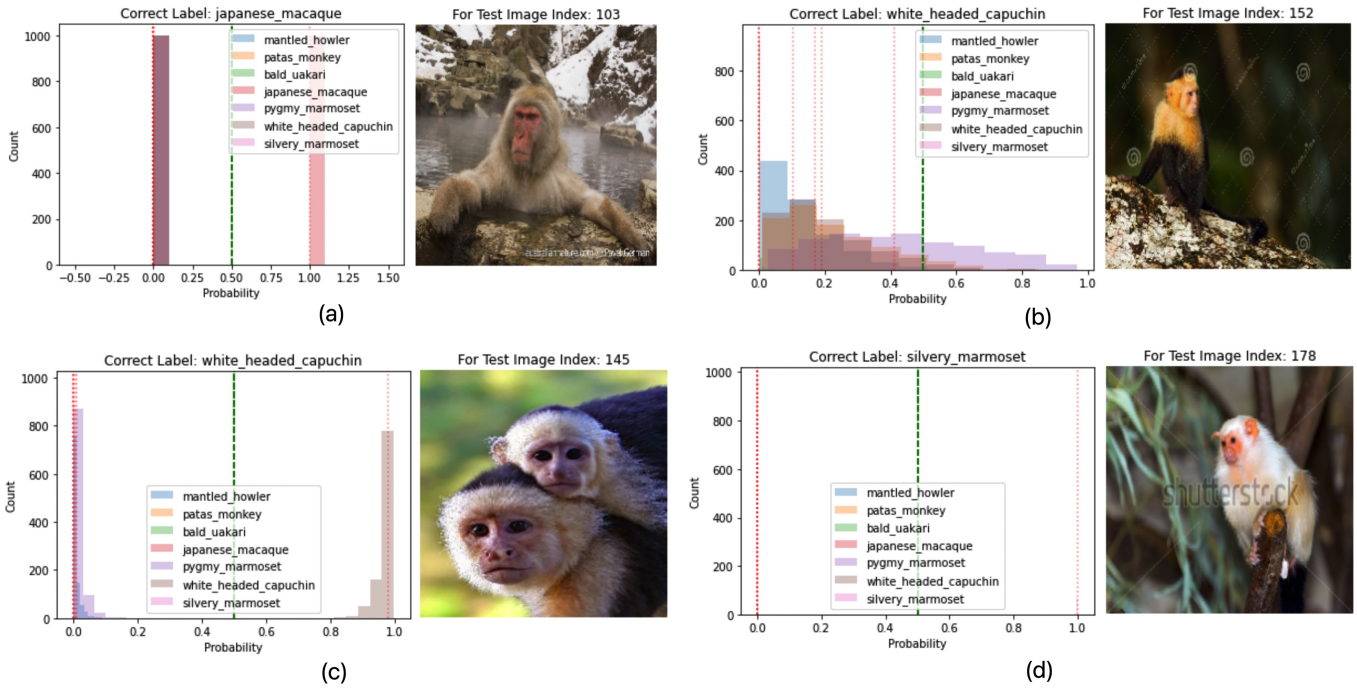

Figure 11: Different levels of uncertainties: (a) Moderately certain predictions, (b) Uncertain Predictions, (c) Predictions with high certainty , (d) Prediction with very high certainty

The implementation of the above classifier has been done in such a way that 2 dropout-rates are required for creating the model instance, one for all Dropout-layers in the convolution blocks and the other for the fully-connected block. We have used Bayesian Optimization (BO) to determine the dropout-rates and other specifications of the model (filters and kernel size of convolutional layers, pooling size of max-pooling layers, etc.) and the dropout rates came out to be 0.2 for both convolutional and fully-connected blocks. It is to be noted that the Monkey Dataset has images with different sizes, so we had to design our data pipeline such that all images are scaled to a size of $(224, 224, 3)$ (a higher resolution was chosen to obtain proper results for model uncertainty). The data-pipeline also augments rotated, width-shifted, height-shifted, horizontally flipped (occasionally) and zoomed images to the training-set so that the model does not overfit and generalises well on the testing data-set.

Dropout Type	**Training Accuracy Threshold**	**Validation Accuracy Threshold**
Normal	0.94	0.83
Gaussian	0.95	0.86

Table 1: Threshold for accuracies used in the custom callback

Both the model instances (with Normal and Gaussian Dropout) were trained on the augmented-set for 200 epochs using an *Adam optimizer* with a learning rate of 0.001. This helped us to get familiar with the training dynamics, so we designed a custom callback which stops the training once a particular threshold of training and validation accuracy are crossed. The thresholds taken for the two model-instances are as given in Table 1.To achieve the thresholds, the model with Normal Dropout took 90 epochs, whereas the model with Gaussian Dropout took 112 epochs but the training time was significantly lower for the latter (exact training time reported in Table 2). The plot of training history is shown in Figure 12.

For each of the model instances and for each objective we plotted the probabilities for each class. Figure 11 shows the different levels of uncertainties observed from the experiment. The model with Normal dropout performed very poorly in capturing uncertainties (both for in-distribution and out-of-distribution data). For out-of-distribution data, the probabilities of certainty shouldn't be very high considering that the classes to which the images belong (namely *Common Squirrel monkey, Black-headed night monkey* and *Nilgiri Langur*) are completely unseen by the model. Normal Dropout gives abnormally high confident predictions for *Nilgiri Langurs* while for other classes, the predictions are moderately certain. Whereas, the model with Gaussian dropout shows moderate to highly uncertain predictions for all classes of the out-of-distribution samples. Ideally we would want the model to show high uncertainty for all out-of-distribution samples, but considering the performance of the model with Normal dropout, Gaussian dropout shows a significant improvement. It is also worthy to note that the Normal-dropout model is highly uncertain for a considerable number of in-distribution samples and overfits the

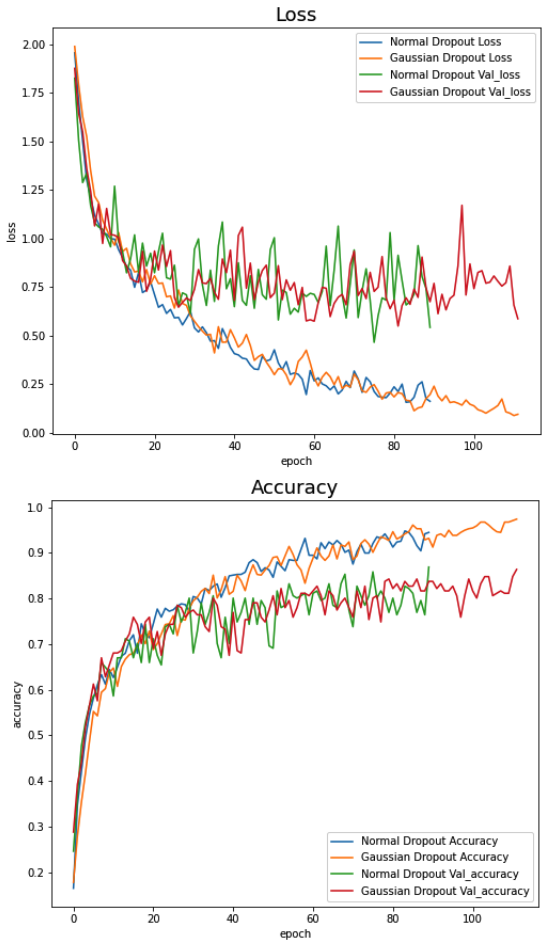

Figure 12: Training history of the model trained on Monkey-Data. The image on the top shows the loss (Categorical Crossentropy) and the bottom image shows the accuracy.

training data very easily in comparison to the Gaussian-dropout model, which shows moderate to very high confidence for the predictions of in-distribution samples at the same time generalising well on test-data.

4.3 Results

Predictive log-likelihood captures how well a model fits the data, with larger values indicating better model fit. Uncertainty quality can be determined from this quantity as well. Following our Bayesian interpretation of dropout we need to define a prior length-scale, and find an optimal model precision parameter τ which will allow us to evaluate the predictive log-likelihood. We have used Bayesian Optimisation (BO) over validation log-likelihood to find optimal τ , and set the prior length-scale to 0.427114830213 for most datasets based on the range of the data. Note that this is a standard dropout NN, where the prior length-scale l and model precision τ are simply used to define the model's weight decay. The Bayesian Optimisation runs with 40 iterations following the original setup, but after finding the optimal parameter values we used some more iterations, as dropout takes longer to converge. Even though the model doesn't converge within 40 epochs (for CO_2 and Monkey Dataset), it gives the Bayesian Optimisation a good indication of whether a parameter is good or not.

Metric	CO2	MNIST	Monkey
Log-likelihood (Normal Dropout)	-10.2631326	-9.861753	-8.257929
Log-likelihood (Gaussian Dropout)	-10.2889364	-9.8619375	-8.258726
RMSE (Normal Dropout)	0.1616447	0.0462226	0.1659820
RMSE (Gaussian Dropout)	0.0328808	0.0356801	0.1768756
Training Time (Normal Dropout)	4601s	253s	3905s
Training Time (Gaussian Dropout)	4223s	210s	3723s

Table 2: Summary of results obtained for various data-sets.

Table 2 summarizes the predictive log-likelihood and test-RMSE for each of the experiments. Gaussian Dropouts clearly show better results than the conventional Dropout considering that the former takes lesser time to converge and out-performs conventional Dropout for unseen data (both in-distribution and out-of-distribution). All codes and detailed results can be found at our Github Repository.

5 Conclusions and Future Research

The probabilistic interpretation of dropout had already been proposed by (Gal and Ghahramani, 2015), (Wang and Manning, 2013) which allows us to model uncertainty out of existing deep learning models. While this approach is faster (in terms of both training and inference time) than stochastic neural networks, we provide an approach which is even faster and tends to handle Out-of-Distribution better than the pre-existing method.

Bernoulli dropout is only one example of a regularisation technique corresponding to an approximate variational distribution which results in uncertainty estimates. Other variants of dropout follow our interpretation as well and correspond to alternative approximating distributions. These would result in different uncertainty estimates, trading-off uncertainty quality with computational complexity. In particular we would follow up with Spatial Dropouts (Tompson et al., 2015). This version of Dropout performs the same function as a conventional Dropout, however, it drops entire 2D feature maps instead of individual elements. If adjacent pixels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, Spatial Dropout will help promote independence between feature maps and should be used instead.

References

[Anjos et al.2015] Ofélia Anjos, Carla Iglesias, Fatima Peres, Javier Martínez, Angela Garcia, and Javier Taboada. 2015. Neural networks applied to discriminate botanical origin of honeys. *Food Chemistry*, 175:128–136, 05.

[Baldi et al.2014] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 07.

[Bergmann et al.2013] Soeren Bergmann, Sören Stelzer, and Steffen Straßburger. 2013. On the use of artificial neural networks in simulation-based manufacturing control. *Journal of Simulation*, 8, 04.

[Damianou and Lawrence2013] Andreas Damianou and N.D. Lawrence. 2013. Deep gaussian processes. 01.

[Dataset2017] Kaggle Dataset. 2017. Monkey image dataset. `https://www.kaggle.com/slothkong/10-monkey-species`.

[Depeweg et al.2017] Stefan Depeweg, José Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. 2017. Decomposition of uncertainty for active learning and reliable reinforcement learning in stochastic systems. 10.

[Der Kiureghian and Ditlevsen2009] Armen Der Kiureghian and Ove Ditlevsen. 2009. Aleatory or epistemic? does it matter? *Structural Safety*, 31:105–112, 03.

[Gal and Ghahramani2015] Yarin Gal and Zoubin Ghahramani. 2015. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *Proceedings of The 33rd International Conference on Machine Learning*, 06.

[Ghahramani2015] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature*, 521:452–9, 05.

[Herzog and Ostwald2013] Stefan Herzog and Dirk Ostwald. 2013. Sometimes bayesian statistics are better. *Nature*, 494:35, 02.

[Keeling et al.2010] Charles Keeling, Robert Bacastow, Arnold Bainbridge, Carl Jr, Peter Guenther, Lee Waterman, and John Chin. 2010. Atmospheric carbon dioxide variations at mauna loa observatory, hawaii. *Tellus*, 28:538 – 551, 03.

[Lecun et al.1998] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12.

[LeCun et al.2010] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

[Lehmann2004] E.L. Lehmann. 2004. *Elements of Large-Sample Theory*. Springer Texts in Statistics. Springer New York.

[Marks and Trafimow2015] Michael Marks and David Trafimow. 2015. Editorial. *Basic and Applied Social Psychology*, 37:1–2, 02.

[Nuzzo2014] Regina Nuzzo. 2014. Statistical errors. *Nature*, 130:150–152, 02.

[Tompson et al.2015] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann Lecun, and Christoph Bregler. 2015. Efficient object localization using convolutional networks. pages 648–656, 06.

[Wang and Manning2013] Sida Wang and C.D. Manning. 2013. Fast dropout training. *30th International Conference on Machine Learning, ICML 2013*, pages 777–785, 01.

[Wolpert1996] David Wolpert. 1996. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8, 03.

[Zhou2012] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms*. Chapman Hall/CRC, 1st edition.