Uncertainty estimation in Neural Networks (Group-4)

Shreyas Kowshik [*]
Roll No. : 17MA20039

Siddhant Agarwal[*]
Roll No.: 17CS30035

Dewang Modi[*]
Roll No.: 17CS30012

Abstract

Uncertainties in a neural network is a measure of how certain the model is in its prediction. These can be of two types : Epistemic or model uncertainty and aleatoric or uncertainty in the predictions. While aleatoric uncertainty is easy to capture, it is difficult to capture epistemic uncertainty using conventional deep learning methods and thus requires Bayesian probability theory. There are several advantages of estimating epistemic uncertainty in sensitive real world settings like robotics and healthcare. In this work, we shall discuss several algorithms that obtain such uncertainty estimates, giving a detailed comparison between these. Moreover, we will highlight the advantages of Bayesian Deep Learning algorithms over conventional deep learning in a variety of tasks and analyse their performance. We conclude by highlighting two real world settings where such estimates have been used to improve the deep learning system.

1 Introduction

Deep Learning has grown tremendously over the past 10 years, leading to all sorts of applications in domains of computer-vision, natural language processing and robotics. However, standard neural network training involves minimizing the empirical risk and obtaining point estimates of the parameters involved. A single setting of parameters is not optimal in several respects : 1). It does not take into account overfitting as is common in MLE and MAP estimates in probabilistic machine learning. 2). The network can have unreasonably confident predictions on unseen data, even though in reality the network does not know what the output is. In other words, the variance of the predictive distribution is very high.

None of the above problems can be tackled using a standard framework. Bayesian Deep Learning is a field that brings together insights from Bayesian Statistics and combines them with deep learning. In theory, using techniques from Bayesian Inference, one can obtain calibrated uncertainty estimates of neural network predictions by modelling the posterior over weights of a neural network. This addresses problem (1) given above by doing a Bayesian Posterior Averaging of the predictions and problem (2) by introducing the uncertainty of the prediction. This has paramount importance in real world applications. For instance, decisions made by neural networks in domains like Autonomous-Driving, Healthcare and Finance should always have an estimate of how confident the network is in its decisions. Moreover, such networks can also be used in reinforcement learning to aid sample-efficient exploration.

Unfortunately, the posterior over a NN's weights is intractable for even the simplest of practical applications. Hence one has to resort to approximate inference techniques built on sampling or variational inference.

In this paper, we survey multiple papers that have addressed the above problems, have built uncertainty estimates for a model's predictions. We show their theoretical motivation and experimental results on standard datasets like MNIST. We conclude finally by looking at approaches that have used uncertainty estimates in real world settings of autonomous driving and reinforcement learning.

[*]All have contributed equally

2 Related Work

To prevent overfitting in neural networks, several regularization schemes have been developed. Some of them are dropout [Srivastava et al., 2014], weight decay, early stopping, drop connect [Wan et al., 2013]. Use of Bayesian methods can also be used to overcome overfitting and capturing uncertainty [MacKay, 1992]. Approximation to posterior can be obtained by using MCMC methods to get Monte Carlo approximations [Neal, 1996]. However, these methods use full dataset, which brings the problem of scalability to large datasets. SGLD [Welling and Teh, 2011b] is based on Langevin dynamics and performs updates using only minibatches of data [Neal et al., 2011]. Some higher-order versions of SGLD with momentum have also become popular, such as stochastic gradient Hamiltionian Monte Carlo (SGHMC) [Chen et al., 2014] and stochastic gradient Nose-Hoover Thermostats (SGNHT) [Ding et al., 2014]. Variational inference [Hinton and van Camp, 1993] is an alternative to MCMC. For neural networks, stochastic variational inference[Graves, 2011] has been somewhat successful but limited by high variance in the gradients. In [Graves, 2011], the authors introduced a variational lower bound estimator for recurrent neural networks with good results. In [Kingma and Welling, 2013], the aim was to obtain more practical estimators using reparameterization trick which resulted in unbiased and efficient stochastic gradient-based variational inference. Their works were oriented towards latent-variable inference.[Blundell et al., 2015b] reported results on inference of global model parameters and also included an application to reinforcement learning. Probabilistic backpropagation[Hernández-Lobato and Adams, 2015] is used to infer marginal posterior probabilities, however they are unsuitable because of intractibilities in deep neural networks.

3 Method

The goal of any classification or regression task on a dataset D is to learn the parameters or the weights for the conditional probability $p(y|x, w)$. Bayesian inference aims to compute the posterior distribution over the weights $p(w|D)$. To compute the distribution of an unknown label $\hat{y}$ for an unknown input $\hat{x}$, it takes the expectation under this posterior distribution as $P(\hat{y}|\hat{x}) = \mathbb{E}_{p(w|D)}[P(\hat{y}|\hat{x}, w)]$. Each possible weight makes a contribution to the final prediction and their contributions are weighed according to the posterior distribution. In other words, this expectation is similar to taking an ensemble over infinite neural networks. Computing the posterior distribution $p(w|D)$, given by $p(w|D) = \frac{p(D|w)p(w)}{p(D)}$ unfortunately is intractable so good approximation techniques are used to perform bayesian inference on neural networks.

3.1 Variation Inference based approaches

Works like [Blundell et al., 2015a], [Kingma et al., 2015] have suggested the use of variational approximation for bayesian inference on NNs. This involves learning a distribution $q(w|\phi)$, which is a known distribution parameterized using ϕ, that is close enough to the posterior distribution $p(w|D)$. KL-Divergence is used as measure of distance between the two distributions. Naturally, the objective is then to minimize $D_{KL}[q(w|\phi)||p(w|D)]$. The objective function $L(\phi)$ becomes,

$$\begin{aligned} L(\phi) &= D_{KL}[q(w|\phi)||p(w|D)] \\ \Rightarrow L(\phi) &= \int q(w|\phi)\text{log}\,\frac{q(w|D)}{p(w)p(D|w)}dw \\ \Rightarrow L(\phi) &= D_{KL}[q(w|\phi)||p(w)] - \mathbb{E}_{q(w|\phi)}[\log p(D|w)] \end{aligned}$$

The first part of the objective is dependent of prior distribution $p(w)$ and second part is the likelihood or dependent on data.

3.1.1 Unbiased Monte-Carlo Gradients

Let ϵ be a random variable sampled from a distribution $q(\epsilon)$. The weights w are computed by transforming this random noise using the deterministic function $t(\phi, \epsilon)$ where ϕ is the variational posterior parameter. The gradient of a expected value function $f(w, \phi)$ under $q(w|\phi)$ is given by,

$$\frac{\delta}{\delta\phi}\mathbb{E}_{q(w|\phi)}[f(w,\phi)] = \mathbb{E}_{\shortparallel(\epsilon)}[\frac{\delta f(w,\phi)}{\delta w}\frac{\delta w}{\delta\phi} + \frac{\delta f(w,\phi)}{\delta\phi}] \tag{1}$$

Using the equation 1 and Monte-Carlo samples, the objective, $f(\phi) = \log q(w|\phi) - \log p(w)p(D|w)$ is optimized. This gives rise to a backpropogation like algorithm called *Bayes by Backprop* [Blundell et al., 2015a] which uses unbiased gradients for the objective to learn a distribution over weights of a neural network. The objective is approximated to $f(\phi) = \sum_i^N \log q(w^i|\phi) - \log p(w^i) - \log p(D|w^i)$ where w^i are the Monte-carlo samples drawn from the variational posterior $q(w|\phi)$.

If the variational posterior is assumed to be a diagonal Gaussian distribution, then the weights can be sampled from a unit Gaussian and transformed using the deterministic function $t(\phi, \epsilon)$ where ϕ will consist of the mean μ and the standard deviation σ. The standard deviation is further parameterized to $\sigma = \log(1 + \exp(\rho))$. Thus the variational posterior parameters are μ and ρ and the transformation function $t(\phi, \epsilon) = \mu + \log(1 + \exp(\rho))$. The resulting algorithm is simple. First sample $\epsilon \sim N(0, I)$, then compute $w = t(\phi, \epsilon)$. Next, compute the objective function $f(\phi)$ and the expected gradient with respect to μ and ρ. Finally update the parameters using these gradients. Alternately, a combination of a diagonal Gaussian posterior and a mixture of Gaussians prior can be used i.e. $p(w) = \Pi_j \pi N(w_j|0, \sigma_1^2) + (1-\pi)N(w_j|0, \sigma_2^2)$ where w_j is the j^{th} weight of the neural network.

3.1.2 Local Reparameterization Trick

Consider the basic objective function, $L(\phi) = D_{KL}[q(w|\phi)||p(w)] - \mathbb{E}_{q(w|\phi)}[\log p(D|w)]$. Let $L_D(\phi) = \mathbb{E}_{q(w|\phi)}[\log p(D|w)]$. Now, $L(\phi)$ can be approximated by stochastic gradient variational Bayes (SGVB), $L^{SGVB} = \frac{M}{N}\sum_i N\log p(y^{(i)}|x^{(i)}, w)$ where $(x^{(i)}, y^{(i)}) \sim D$ are the datapoints in the minibatch of size N, M is the number of minibatches.

The variance of $L^{SGVB}(\phi)$ is given by,

$$Var[L^{SGVB}(\phi)] = \frac{M^2}{N^2}[\sum_{i=1}^{N} Var[L_i] + 2\sum_{i=1}^{N}\sum_{j=i+1}^{N} Cov[L_i, L_j]] \tag{2}$$

where, $L_i = \log p(y^{(i)}|x^{(i)}, w)$.

An attempt can be made to make the $Cov[L_i, L_j] = 0$ to reduce the variance in the objective. This can be done by sampling different weights for each sample in the minibatch. By doing so, the global uncertainty in the weights is translated into a form of local uncertainty that is independent across examples and easier to sample. But, sampling separate weights for each sample can be computationally expensive, so we can sample activations directly. This is the algorithm discussed by [Kingma et al., 2015].

For example, lets consider a hidden layer in a standard neural network $Z = XW$ where Z is the pre-activation outputs, X is the input (or the output of the previous layer) and W is the weight matrix. Each $w_{i,j}$ in W is specified by the variational posterior $q(w_{i,j}|\phi) = N(\mu_{i,j}, \sigma_{i,j}^2)$. This means that the weights are sampled from a unit normal and transformed i.e. $w_{i,j} = \mu_{i,j} + \epsilon_{i,j}\sigma_{i,j}^2$ where $\epsilon_{i,j} \sim N(0,1)$. If a separate W is sampled for every sample in the minibatch, we can ensure that $Cov[L_i, L_j] = 0$. But, this is computationally expensive as sampling N weight matrices for every minibatch is tedious. The pre-activations, Z, are also normally distributed with mean $\hat{\mu}_{i,j} = \sum_k \mu_{i,k}x_{k,j}$ and variance $\hat{\sigma}^2 = \sum_k \sigma_{i,k}^2 x_{k,j}^2$ and size of Z are much smaller compared to W. So, instead of sampling W, Z is sampled from $N(\hat{\mu}, \hat{\sigma}^2)$.

3.1.3 Monte Carlo Dropout

Authors in [Gal and Ghahramani, 2016] show a theoretically principled way to make sense of dropout based neural-networks. Dropout based networks when viewed from a bayesian viewpoint are approximately equivalent to a Deep Gaussian Process, the kernel of which depends on the non-linearities of the given network.

More formally, let f be the output of a Neural-Network with L layers, and $l(.,.)$ be a loss function. Denote by W_i the weight matrix of the i^{th} layer of dimension $K_i x K_{i-1}$ and b_i be the bias parameter.

Let X, Y be our given dataset with inputs X and outputs Y. Dropout based neural network training works by sampling bernoulli random variables at each layer except the final layer, multipliying these samples with the layer values and training the resulting network on the following objective :

$$L_{dropout} = \frac{1}{N}\Sigma_{i=1}^{N} l(y_i, f_i) + \lambda \Sigma_{i=1}^{L}(||W_i||_2^2 + ||b_i||_2^2)$$

The second part of the above objective is equivalent to L_2-regularization. Such an interpretation puts randomness over the layer neurons and looks at weights from a non-random perspective. However, the same can be interpreted in terms of the weights being random.

Now consider a deep Gaussian Process with L layers, where the weights W_i are random. Denote $\omega = \{W_i\}_{i=1}^{L}$. Let each row of W_i distribute according according to prior $p(w)$. The predictive probability of such a network is given by :

$$p(y|x, X, Y) = \int p(y|x, \omega)p(\omega|X, Y)d\omega$$
$$p(y|x, \omega) = \mathbb{N}(y|f(x, \omega), \beta^{-1}I_D)$$

where β is a precision parameter. The output f is composed of multiple layers, where each layer is of the form :

$$o_i = \sigma(W_i in_i + b_i)$$

with inputs in_i and outputs o_i and $\sigma(.)$ is a non-linearity. The posterior $p(\omega|X, Y)$ is intractable, hence a variational approximation $q(\omega)$ is used for it, defined as :

$$W_i = M_i \cdot diag([z_{i,j}]_{j=1}^{K_i})$$
$$z_{i,j} \sim Bernoulli(p_i); \;\; i = 1, \cdots L, \;\; j = 1 \cdots K_{i-1}$$

Here the variational parameters are the probabilities p_i and matrices M_i. It can be shown that instead of masking out layer neurons in dropout, one can mask out the immediately successive weight matrix M_i and result in the same computation. Thus this transfers randomness from the layers to the weights and views each forward pass as as sampling weights from the variational distribution. It can be shown that [Gal] [Gal and Ghahramani, 2016] minimizing the KL-Divergence between $q(\omega)$ and $p(\omega|X, Y)$, in certain hyperparameter settings, leads to optimizing the same objective as $L_dropout$. This gives an interpretation of dropout networks as doing an approximate inference to model the posterior.

Obtaining Model Uncertainty : The posterior predictive, approximated by the variational distribution is given by :

$$q(y^*|x^*) = \int p(y^*|x^*, \omega)q(\omega)d\omega$$

where $\omega = W_{i}{}_{i=1}^{L}$. This integral can be approximated empirically by sampling T realisations of the weights using the Bernoulli distributions, leading to a set of weights : $\{W_1^t, \cdots, W_L^t\}_{t=1}^{T}$. The mean and variance of this predictive can then be given by :

$$\mathbb{E}_{q(y^*|x^*)}(y^*) = \frac{1}{T}\Sigma_{t=1}^{T} f^*(x^*, W_1^t, \cdots, W_L^t)$$

$$V_{q(y^*|x^*)(y^*)} = \frac{1}{T}\Sigma_{t=1}^{T} f^*(x^*, W_1^t, \cdots, W_L^t)^T f^*(x^*, W_1^t, \cdots, W_L^t) \\ - \mathbb{E}_{q(y^*|x^*)}(y^*)^T \mathbb{E}_{q(y^*|x^*)}(y^*) + \beta^{-1}I_D$$

which is the sample mean of the stochastic forward passes through the network and sample variance plus a precision term. Thus uncertainty estimates can simply be obtained by using dropout and doing multiple forward passes through the network, and looking at the sample mean and variance.

3.2 Non-VI based approaches

The class of approaches we discuss here do not involve any variational distribution. Instead, they introduce randomness into the optimization procedure of the network weights, which can then be shown to converge to samples from the true posterior. Two such approaches have been presented.

3.2.1 Stochastic Gradient Langevin Dynamics (SGLD)

Stochastic Gradient Langevin Dynamics[Welling and Teh, 2011a] is a popular technique where Bayesian learning can be done on large datasets by learning from small mini-batches. In this technique, a small amount of noise is added to the standard SGD algorithm, and by annealing appropriately, the samples will converge to true posterior. Through this transition from optimization to sampling from posterior prevents overfitting. By injecting appropriate Gaussian noise in parameter update rule, SGD which normally converges to maximum a posteriori mode, would instead cause updates in such a way that the trajectory of parameters converges to posterior distribution.

MLP and MAP estimates have several issues - they cannot show uncertainty and high chances of overfitting.

Combining ideas from SGD and Langevin dynamics gives several advantages from both of the methods - the minibatch update rule allows for training on large-scale datasets while the addition of Gaussian noise causes trajectory to follow posterior which helps in capturing uncertainty and preventing overfitting.

Assume have a dataset $\{X_i\}_{i=1}^N$. Our model parameters are θ apriori distributed according to $p(\theta)$ and having likelihood of $\Pi_{i=1}^N p(X_i|\theta)$. The posterior is given as $p(\theta|X) \propto p(\theta)\Pi_{i=1}^N p(X_i|\theta)$. We are interested in finding the MAP estimate of this posterior. Consider that during optimization, the parameters change at each step by a value $\Delta\boldsymbol{\phi}_t$.

SGD updates using the following rule :

$$\Delta\boldsymbol{\phi}_t = \gamma_t\Big(\nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{\phi}_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{d}_{t_i}|\boldsymbol{\phi}_t)\Big) \tag{3}$$

where $\{\gamma_t\}$ is a sequence of step sizes, and $\mathcal{D}^t$ is a subset of n points sampled from D at current iteration.

Gradient vectors during DNN training are obtained using the backpropagation algorithm. At test time, the Bayesian predictive estimate for input x, is given by $p(y|x, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\phi}|\mathcal{D})}[p(y|x, \boldsymbol{\phi})]$. The MAP estimate simply approximates this expectation as $p(y|x, \mathcal{D}) \approx p(y|x, \boldsymbol{\phi}_{\text{MAP}})$, ignoring parameter uncertainty.

Instead of directly using the gradient vector to update the parameters, SGLD constructs a Gaussian centered on the gradient vector and adding i.i.d. noise to each vector component. It thus samples $\boldsymbol{\phi}$ from the posterior distributions via a Markov Chain with steps:

$$\Delta\boldsymbol{\phi}_t \sim \mathcal{N}\left(\frac{\gamma_t}{2}\Big(\nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{\phi}_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\phi}} \log p(\boldsymbol{d}_{t_i}|\boldsymbol{\phi}_t)\Big), \gamma_t \mathbf{I}\right) \tag{4}$$

with $\mathbf{I}$ denoting the identity matrix. The step sizes $\{\gamma_t\}$ are decreasing, *i.e.*, $0 < \gamma_{t+1} < \gamma_t$, with 1) $\sum_{t=1}^{\infty} \gamma_t = \infty$; and 2) $\sum_{t=1}^{\infty} \gamma_t^2 < \infty$

Given a set of samples from the update rule (4), posterior distributions can be approximated via Monte Carlo approximations as $p(y|x, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y|x, \boldsymbol{\phi}_t)$, where T is the number of samples. [Welling and Teh, 2011a]

When the SGLD algorithm is run, initial steps are dominated by the stochastic gradient noise because of which the algorithm behaves somewhat similar to a usual SGD algorithm. As the algorithm progresses, the injected noise starts to dominate, and the algorithm behaves like Langevian dynamics Metropolis Hastings algorithm. Therefore, the algorithm transitions smoothly between them. Therefore it is important that collecting of the posterior samples is done only after the algorithm has transitioned to Langevin dynamics phase.

Once some samples are obtained, [Welling and Teh, 2011a] show that rather than taking simple averaging, a weighted average with weights as the step sizes should be taken which results in the estimator having lower variance.

3.2.2 Pre-conditioned SGLD

DNNs have become increasingly common, however training them effectively is a challenge as they have pathological curvatures and saddle points. Because of these SGLD becomes inefficient. In normal optimization, there are several ways to overcome this problem for SGD algorithm, one of which is using a preconditioning matrix. Pre-conditioned SGLD combines the two approaches together i.e. uses a pre-conditioning matrix with SGLD.

In standard SGLD, the step size is same for all the parameters. However, it can become inefficient when the curvatures are different for different parameters. A solution is to use a preconditioning matrix $G(\boldsymbol{\phi})$ in SGLD. [Li et al., 2016] proposed to use the same preconditioner as in RMSProp. The update rule becomes :

$$\Delta\boldsymbol{\phi}_t \sim \frac{\epsilon_t}{2}\Big[G(\boldsymbol{\phi}_t)\Big(\nabla_{\boldsymbol{\phi}}\log p(\boldsymbol{\phi}_t) + \frac{N}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\phi}}\log p(\boldsymbol{d}_{t_i}|\boldsymbol{\phi}_t)\Big) + \Gamma(\boldsymbol{\phi}_t)\Big] + G^{\frac{1}{2}}(\boldsymbol{\phi}_t)\mathcal{N}(0,\epsilon_t\mathbf{I})$$

$$G(\boldsymbol{\phi}_{t+1}) = \text{diag}\left(\mathbf{1}\oslash\left(\lambda\mathbf{1}+\sqrt{V(\boldsymbol{\phi}_{t+1})}\right)\right) \tag{5}$$

$$V(\boldsymbol{\phi}_{t+1}) = \alpha V(\boldsymbol{\phi}_t) + (1-\alpha)\bar{g}(\boldsymbol{\phi}_t;\mathcal{D}^t)\odot\bar{g}(\boldsymbol{\phi}_t;\mathcal{D}^t)\ , \tag{6}$$

where $\bar{g}(\boldsymbol{\phi}_t;\mathcal{D}^t) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\phi}}\log p(\boldsymbol{d}_{t_i}|\boldsymbol{\phi}_t)$ is the sample mean of the gradient for the mini-batch $\mathcal{D}^t$. $\alpha\in[0,1]$. $\odot$ denotes element-wise matrix multiplication and $\oslash$ denotes element-wise matrix division.

RMSprop is an extension of gradient descent that uses a decaying average of partial gradients for adapting step size for each parameter. Gradient information is locally consistent. When the landscape is curved, the gradients are larger than when the landscape is flat.

4 Experiments

We test the different bayesian inference techniques for both classification and regression. For classification, we use MNIST dataset [mnist] which consists of handwritten digits. It consists of 60,000 training images and 10,000 testing images. For regression, we used data generated from a Gaussian Process, $GP(0,K)$ where K is an RBF kernel added with a diagonal gaussian noise.

4.1 Classification Results

Table 1 contains the classification errors on MNIST test set. The plots for the training and testing errors with respect to the number of training epochs are shown in the Figure 1

Method	Classification error	Time per iter.
Baseline	0.02500	-
BBP (Gaussian Prior)	0.023500	38.437286s
BBP (GMM Prior)	0.023400	72.007019s
Local Reparam	0.025100	30.978277s
MC Dropout	**0.019000**	16.925309s
SGLD	0.035800	4.913495s
p-SGLD	0.033100	4.808156s

Table 1: Test set erorr rate on MNIST and Iteration times for each approach. One iteration time involves training on a batch of data and evaluation on complete development data.

4.2 Regression Results

On learning regression models, we can directly plot the two types of uncertainties in the model. The regression plots are shown in Figure 2. The blue region shows the epistemic uncertainty while the yellow region shows the aleatoric uncertainty. It is evident from the plots that as one goes further away from the training regime, the uncertainty in the predictive distribution increases, implying that the model is not sure of what it predicts. This is the expected behavior as well.

4.3 Analysis

The following experiment was performed for analysis : A single image of a digit 3 was chosen and it was rotated from 0-180 degrees successively by 20 degrees. At each step, the image was passed

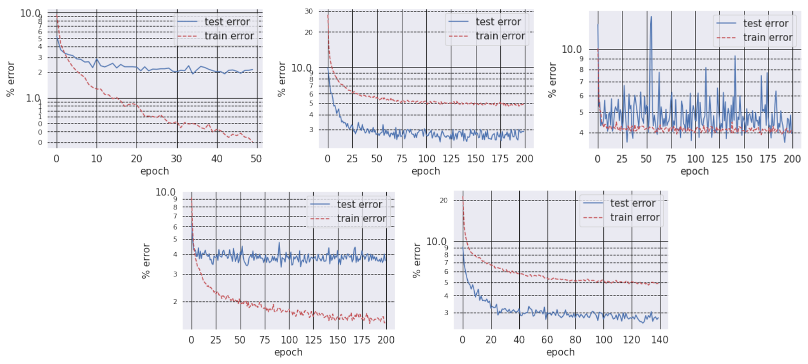

Figure 1: Results for classification - error curves. Top Row (From left) : MC-Dropout, BBP (GMM Prior), SGLD. Bottom Row (From left) : p-SGLD, Local Reparam

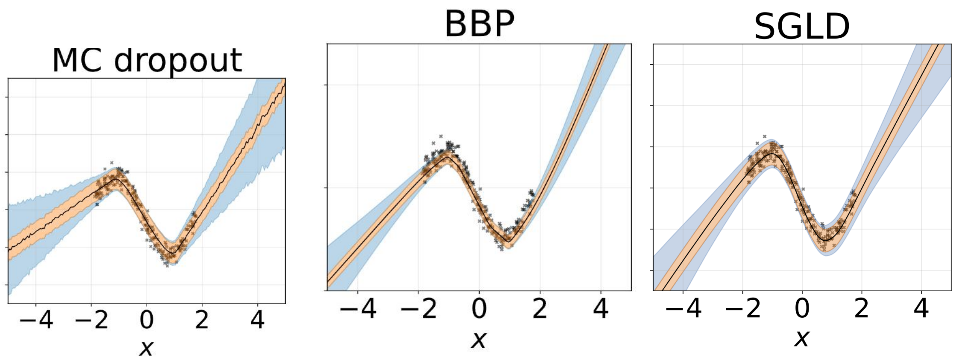

Figure 2: Regression Results. Yellow represents predictive uncertainty. Blue represents model uncertainty. It is clear as we go away from training regime, uncertainty increases by thicker blue fillings.

through the networks and the predictive probabilities were obtained including the mean and the variance. Predictive entropies of the predictive distribution were also obtained. These have been plotted in Figure 3. From the figures, it is clear that as the angle increases the uncertainty in the correct class's prediction also increases shown by the thickening of the green region. The same can also be observed in the red region. These regions are the thinnest when the rotation angle is zero, indicating that the network is most certain of its prediction for this image compared to the other images. The predicted class probabilities for all the images is given in Figure 4.

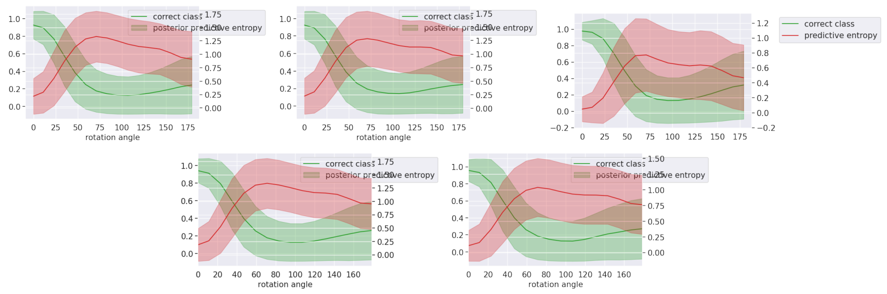

Figure 3: Analysis - Correct class probability and Predictive Uncertainty along with their variances. Top Row (From Left) : BBP (Gaussian Prior), Local Reparameterizatiion, MC-Dropout. Bottom Row : SGLD, p-SGLD

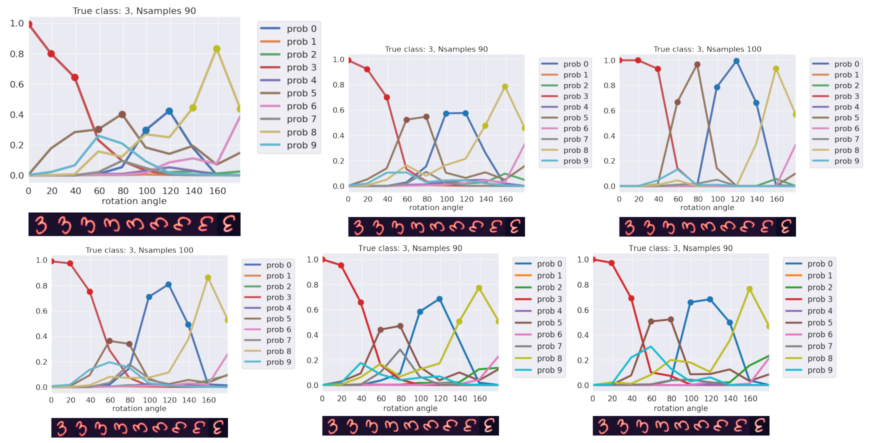

Figure 4: Analysis : Variation of predictive distributions for each class by rotating a single image by 20 degrees successively. Top-Row (From Left) : BBP (Gaussian Prior), BBP (GMM Prior), MC-Dropout. Bottom Row (From Left) : Local Reparam, SGLD, p-SGLD.

5 Applications of Uncertainty Estimates of Neural Networks

Epistemic uncertainty or uncertainty in the model has been widely exploited be it in reinforcement learning, control theory or computer vision [Loquercio et al., 2020]Deisenroth and Rasmussen [2011]. Over the years, researchers have widely used several ways to estimate the uncertainty in the model. In this section, we will discuss a few applications where model uncertainty is extensively used.

5.1 Applications in Reinforcement Learning

A common use of model uncertainty is solve the exploration-exploitation trade-off. Many algorithms still rely on ϵ-greedy exploration method which although is guaranteed converge,it does not use model uncertainty. There are several algorithms that try to exploit epistemic uncertainty by estimating a posterior over the expected value, for example using Bayesian linear regression [Osband et al., 2016], [Azizzadenesheli et al., 2018b] or Bayesian neural networks [Lipton et al., 2016]. Upper Confidence Bound or UCB is an algorithm that uses uncertainty information to deal with the exploration-exploitation tradeoff [Auer, 2003]. It takes the action that maximizes the sum of reward's mean and variance.

[Blau et al., 2019] uses Bayesian Linear Regression and latent state embeddings to generate intrinsic rewards to encourage the agent to visit new states. The work introduces a curiosity based exploration technique that can be applied to any arbitrary RL algorithm. Using BLR, the uncertainty of the model for arbitrary states can be computed. This uncertainty will show that the new states are dissimilar to the already observed states.

[Azizzadenesheli et al., 2018a] introduces Bayesian Deep Q-Learning that uses Thompson's Sampling instead of the commonly used ϵ-greedy. Thompson's sampling enables better exploration in high dimensions through posterior sampling thereby greatly improving the performance. To reduce computation costs, the authors only apply uncertainty to the final layer of the neural network through Bayesian Linear Regression. The improvements over DQN outweigh any other improvement techniques such as Prioritized Replay or Dueling. This shows how important better exploration techniques are for performance of RL agents.

5.2 Uncertainty estimation for Steering Angle Prediction

Authors in [Loquercio et al., 2020] propose an uncertainty estimation procedure for steering angle prediction in autonomous driving, given an image of the input frame. Monte Carlo Dropout is used for obtaining the estimates. Moreover they also extend their approach to cases of already trained networks. This has a lot of practical value since most computer vision networks are highly sophisticated and pre-trained networks available are generally not trained keeping in mind uncertainty calibration. The authors observe that in images that are well illumniated, the uncertainties are pretty low. But in low illumination images/noisy images, the estimates go up significantly, indicating that the network is not confident of its predictions. Such a model is of utmost importance in a fully self driving setup if one is to deploy a neural network.

6 Conclusion

Estimating epistemic uncertainty in a neural network has a wide variety of applications. As a result, researchers have tried to exploit various ways to compute it. One way to compute the model uncertainty is to use bayesian inference and estimate the entire posterior distribution. But in neural networks bayesian inference is intractable. Hence, several approximate methods have been derived that give an estimate of the uncertainty. A common path followed by several methods is to use variational inference to learn a variational posterior which is close to the actual posterior.

In this work, we have shown several algorithms that perform an approximate bayesian inference on neural networks. We have analysed the uncertainty estimates for unseen samples both in classification and regression settings. Finally, we briefly discussed some recent works that rely on the uncertainty estimates provided by bayesian inference.

References

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3(null): 397–422, mar 2003. ISSN 1532-4435.

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *CoRR*, abs/1802.04412, 2018a. URL http://arxiv.org/abs/1802.04412.

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9, 2018b. doi: 10.1109/ITA.2018.8503252.

Tom Blau, Lionel Ott, and Fabio Ramos. Bayesian curiosity for efficient exploration in reinforcement learning. *CoRR*, abs/1911.08701, 2019. URL http://arxiv.org/abs/1911.08701.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015a.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015b.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

Marc Deisenroth and Carl Rasmussen. Pilco: A model-based and data-efficient approach to policy search. pages 465–472, 01 2011.

Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D Skeel, and Hartmut Neven. Bayesian sampling using stochastic gradient thermostats. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/21fe5b8ba755eeaece7a450849876228-Paper.pdf.

Yarin Gal. Uncertainty in deep learning.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1050–1059. JMLR.org, 2016.

Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*, pages 1861–1869. PMLR, 2015.

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*, COLT '93, page 5–13, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897916115. doi: 10.1145/168304.168306. URL https://doi.org/10.1145/168304.168306.

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

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick, 2015.

Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Zachary C. Lipton, Jianfeng Gao, Lihong Li, Xiujun Li, Faisal Ahmed, and Li Deng. Efficient exploration for dialog policy learning with deep BBQ networks \& replay buffer spiking. *CoRR*, abs/1608.05081, 2016. URL http://arxiv.org/abs/1608.05081.

Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, PP:1–1, 02 2020. doi: 10.1109/LRA.2020.2974682.

David J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Comput.*, 4(3): 448–472, may 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448. URL https://doi.org/10.1162/neco.1992.4.3.448.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/wan13.html.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011a.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011b. Omnipress. ISBN 9781450306195.