Bayesian Deep Learning

Niranjan Thakurdesai   Ayush Baid   Manan Mehta   Priyal Jhaveri

1 Project Summary

Deep Learning models are increasing in their use day-by-day: they are the common choice of modeling in domains like autonomous navigation systems, conversation systems, and medical diagnosis. With the increase in use, the scrutiny of these models has also increased. Researchers and engineers are trying to understand the reasons behind model predictions and want to know the uncertainties involved in the model outputs. We want to know where the model is not confident in its predictions. The quantification of uncertainty is extremely important as these models are becoming ubiquitous. We briefly discuss the theory of Bayesian learning and different algorithms which have been proposed to tackle the problem. We also discuss some experiments which help us connect some general phenomena observed in training deep networks with the uncertainties from Bayesian approaches.

2 Introduction

Deep neural networks are now a common occurrence in automated systems deployed in the real world [1]. Erroneous predictions made by such networks can have catastrophic consequences on humans. For example, an assisted driving system confused the white side of a trailer for bright sky, resulting in a fatality [2].

The underlying problem here is that deep networks only provide point estimates in their predictions. However, it is important to assign uncertainties to each prediction to make an automated system safe, i.e. a trained model should know when it does not know. For example, if the assisted driving system above were able to assign a high level of uncertainty to its erroneous prediction, it may have been able to avert the disaster by alerting the driver. Regression models do not provide uncertainty estimates at all. Classification models often use softmax as the last layer and its output is often erroneously interpreted as model confidence [1], [3].

Bayesian probabilistic modeling offers a practical framework to estimate predictive uncertainties in deep neural networks. Models learned under the Bayesian learning formulation are probabilistic and offer confidence bounds for data analysis and decision-making.

In this project, we first review the types of uncertainties (section 3), study some important methods in Bayesian modeling (section 5) and perform experiments on a toy regression task (section 6).

3 Types of Uncertainties

There are three types of predictive uncertainties: aleatoric, epistemic, and distributional.

Aleatoric or data uncertainty is due to noise inherent in the data. Hence, it cannot be reduced by increasing the size of the training set. It arises from sensor noise, incomplete data, class ambiguity, and label noise [4]. It can be modeled by making the outputs of a neural network probabilistic. We talk more about the types of aleatoric uncertainty and methods to compute it in appendix section A

Epistemic uncertainty is the uncertainty of the model. The deep networks have numerous local optima, one of which might be selected at the end of the training process. There might be a large number of models which have similar performance on the training set. Also, different network architecture choices (e.g. number of hidden layers, size of hidden layers, choice of activation functions, etc.) can lead to same performance on training set and hence the uncertainty [5].

Distributional uncertainty is due to uncertainty in determining whether an example is out-of-distribution, i.e. if the example does not belong to the distribution of the training samples [6]. The previous two uncertainties are an implicit measurement of the distribution uncertainty. However, an explicit measurement of the uncertainties arising from a change in distribution during the model deployment is essential as it can provide actionable insights to debug poor performance, e.g. distributional uncertainty indicates the need to collect more training data from the new distributions.

4 Problem Setup: Bayesian Probabilistic Modeling

In the traditional deep learning setup, we are given a dataset $\mathcal{D} \doteq \{\mathbf{X}, \mathbf{Y}\}$, where $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ are the input feature vectors and $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^N$ are the corresponding outputs, which can be either vectors (regression) or categorical labels (classification). The weights of the network are denoted by $\mathbf{W}$. The network evaluation for $\mathbf{x}_i$ is denoted by $f(\mathbf{x}_i; \mathbf{W})$.

We make use of two important probability distributions, the prior $P(\mathbf{W})$ and the likelihood $P(\mathbf{Y}|\mathbf{X}, \mathbf{W})$. The prior represents our belief in the parameters and the likelihood represents our belief in how well the given parameters explain the data. Softmax and Gaussian likelihoods are widely used for classification and regression respectively. Gaussian priors are typically used which translates to L2 regularization on weights.

The posterior distribution for the parameters can be derived using Bayes' rule:

$$P(\mathbf{W}|\mathcal{D}) = P(\mathbf{W}|\mathbf{X}, \mathbf{Y}) = \frac{P(\mathbf{Y}|\mathbf{X}, \mathbf{W})P(\mathbf{W})}{P(\mathbf{Y}|\mathbf{X})} \tag{1}$$

In the traditional learning setup, we use gradient-based methods to minimize a *loss function* based on equation 1 to obtain the MAP estimate $\hat{W}_{\text{MAP}}$. This is used to predict the output $\mathbf{y}^*$ for a new input $\mathbf{x}^*$.

$$\hat{\mathbf{W}}_{\text{MAP}} \doteq \arg\max_{\mathbf{W}} P(\mathbf{Y}|\mathbf{X}, \mathbf{W})P(\mathbf{W}) \tag{2}$$

$$\mathbf{y}^* \doteq \arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}^*, \hat{\mathbf{W}}_{\text{MAP}}) \tag{3}$$

One can see that the uncertainty arising from $\mathcal{D}$ and $\mathbf{W}$ are lost in this setup. This can lead to erroneous predictions with high confidence [3].

Bayesian Neural Networks (BNNs) use the posterior to marginalize over weights and get a probability distribution over the output. This is called inference. Given a new input $\mathbf{x}^*$, BNNs predict the output distribution as follows:

$$P(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \int P(\mathbf{y}^*|\mathbf{x}^*, \mathbf{W})P(\mathbf{W}|\mathcal{D})\, \mathrm{d}\mathbf{W} \tag{4}$$

To perform inference, we need to evaluate the posterior (equation 1). The denominator is known as model evidence or marginal likelihood, computed by marginalizing out $\mathbf{W}$:

$$P(\mathbf{Y}|\mathbf{X}) = \int P(\mathbf{Y}|\mathbf{X}, \mathbf{W})P(\mathbf{W})\, \mathrm{d}\mathbf{W} \tag{5}$$

Except some special cases [7], the marginalization is intractable, and needs to be approximated. This is particularly true with deep networks. The complexity of the two integrals (equation 4, 5) are the main blockers for Bayesian Learning. We will discuss some work towards solving this problem in the next section.

5 Methods

We cover 3 methods in this section and have covered an additional method in appendix E.

5.1 Variational Inference

Instead of evaluating the true posterior $P(\mathbf{W}|\mathcal{D})$ analytically, we can approximate it with a variational distribution $q_\theta(\mathbf{W})$, parameterized by θ, which is easier to evaluate. This process is called Variational Inference (VI) [8]. We would like the variational distribution to be as close to the true posterior distribution as possible. To measure the approximation quality, we need a measure of similarity between true posterior and the variational approximation. The Kullback-Leibler (KL) divergence is a common measure of similarity:

$$\mathrm{KL}(q_\theta(\mathbf{W})||P(\mathbf{W}|\mathcal{D})) = \int q_\theta(\mathbf{W}) \log \frac{q_\theta(\mathbf{W})}{P(\mathbf{W}|\mathcal{D})}\, \mathrm{d}\mathbf{W} \tag{6}$$

By expanding equation 6 and using properties of KL divergence (section B), we can get the following lower bound for model evidence, known as ELBO:

$$\mathcal{L}_{\text{VI}}(\theta) = \int q_\theta(\mathbf{W}) \log P(\mathbf{Y}|\mathbf{X}, \mathbf{W})\, \mathrm{d}\mathbf{W} - \mathrm{KL}(q_\theta(\mathbf{W}), P(\mathbf{W})) \tag{7}$$

Minimizing $\mathrm{KL}(q_\theta(\mathbf{W})||P(\mathbf{W}|\mathcal{D}))$ is equivalent to maximizing the ELBO. The negative ELBO can be interpreted from an information theoretic point of view as minimum description length (MDL) (appendix section F) loss function [9] or variational free energy [10]. In the ELBO loss $\mathcal{L}_{\mathrm{VI}}(\theta)$, there is tension between the two terms: the first term pulls the parameters towards explaining the data well and the second term pulls it towards the prior.

After estimating the $q_\theta(\mathbf{W})$ which maximizes the ELBO, we can compute the approximate predictive distribution by replacing the posterior with $q_\theta(\mathbf{W})$ in 4.

However, there are a few problems in maximizing the ELBO, specially the term $\int q_\theta(\mathbf{W}) \log P(\mathbf{Y}|\mathbf{X}, \mathbf{W})$: it is not tractable for BNNs with more than a single hidden layer; and it does not scale well for large datasets due to integral over the entire dataset. To solve some of these problems, stochastic approximate inference is used. This involves data subsampling [10] (refer Appendix C) and Monte Carlo (MC) integration [1]. Another example of approximate inference is Bayes by Backprop by Blundell et al. [11], where they use the reparametrisation trick and model the scalar weight as a Guassian random variable $\mathbf{W}_i = \mu_i + \sigma_i \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, 1)$.

5.2 Probabilistic Backpropagation

Lobato et al. [12] introduced Probabilistic Backpropagation (PBP) to make BNNs scalable. They assume that each target $\mathbf{y}_i$ can be written as an output of the neural network plus an additive noise, where the noise is a Gaussian random variable parameterized with γ. Each scalar in $\mathbf{W}$ is modelled as a zero mean Gaussian random variable with variance as λ^{-1}. Hence, the likelihood and the prior terms can be written as:

$$P(\mathbf{Y}|\mathbf{W}, \mathbf{X}, \gamma) = \prod_{i=1}^{N} \mathcal{N}\left(\mathbf{y}_i | f\left(\mathbf{x}_i; \mathbf{W}\right), \gamma^{-1}\right) \qquad P(\mathbf{W}|\lambda) = \prod_{i} \mathcal{N}\left(W_i | 0, \lambda^{-1}\right)$$

We can now apply the Bayes' rule to obtain the posterior distribution on $\mathbf{W}$ and hyperparameters γ and λ (which have Gamma distribution priors) as:

$$P(\mathbf{W}, \gamma, \lambda|\mathcal{D}) = \frac{P(\mathbf{Y}|\mathbf{W}, \mathbf{X}, \gamma) P(\mathbf{W}|\lambda) P(\lambda) P(\gamma)}{P(Y|X)} \tag{8}$$

Equation 4 can now be written as:

$$P(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \int \mathcal{N}\left(\mathbf{y}^* | f\left(\mathbf{x}^*\right), \gamma\right) P(\mathbf{W}, \gamma, \lambda|\mathcal{D}) \, \mathrm{d}\gamma \, \mathrm{d}\lambda \, \mathrm{d}\mathbf{W} \tag{9}$$

This computation is still not tractable, particularly due to equation 8, and we need to use an approximate inference method. Lobato et al. introduced an alternative to the traditional backpropagation algorithm called PBP: it uses normal distributions instead of point estimates for each weight parameter in the network.

Similar to backpropagation, PBP has two phases: a forward pass of the input data and a backward pass of the gradients. As the activations after each layer are now random variables, PBP sequentially approximates them with Gaussians to match the marginal means and variances. Performing this step layer-by-layer completes the forward pass. The loss is the marginal log-probability of the target y. The derivatives with respect to the mean and variances are backpropagated through reverse-mode automatic differentiation. To get the new distributions for $\mathbf{W}$, a new Gaussian belief is computed to approximate the complex distribution obtained by incorporating the gradients w.r.t to the mean and variance of the current belief.

5.3 Stochastic Gradient Langevian Dynamics

Welling et al. [13] introduced an optimization algorithm called Stochastic Gradient Langevian Dynamics (SGLD), which is simple to incorporate in traditional learning models optimized with Stochastic Gradient Descent (SGD). In a setup with annealing step sizes, an additional of a *right* amount of noise results in convergence to true posterior distribution (equation 1). SGLD belongs to the Markov Chain Monte Carlo (MCMC) class of algorithms, and its use of mini-batches make it more practical than other MCMC algorithms which work on the whole dataset at a time.

Let $\mathbf{W}_t$ denote the network weights at time-step t. Optimizing equation 1 with SGD requires we take

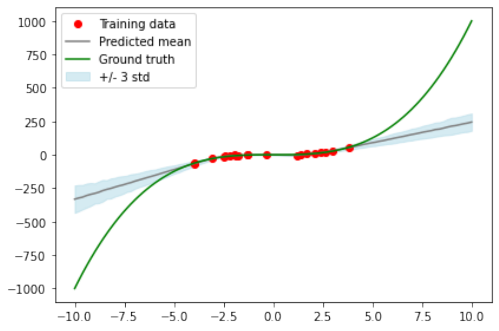


Figure 1: Monte Carlo dropout

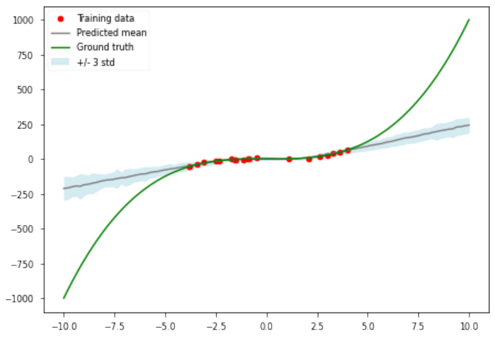


Figure 2: Bayes by Backprop

a step $\Delta \mathbf{W}_t$ in the direction of the gradient of $\mathbf{W}$. In SGLD, we take the step obtained from SGD and inject a Gaussian noise η_t. This technique is called Langevin dynamics [14], and it ensures that the learnt parameters do not just collapse to the MAP solution. With noise $\eta_t \doteq \mathcal{N}(0, \epsilon_t)$, the step for each gradient update becomes:

$$\Delta \mathbf{W}_t = 0.5\epsilon_t \left(\nabla \log P\left(\mathbf{W}_t\right) + \frac{N}{M} \sum_{n=1}^{M} \nabla \log P\left(\mathbf{Y}_{ti} | \mathbf{W}_t, \mathbf{X}_{ti}\right) \right) + \eta_t \tag{10}$$

The variance of the noise is balanced with the gradient step size. The authors derive results which show two phases of operation for the optimization: the initial phase where stochastic gradient noise dominates and the algorithm effectively imitates an SGD algorithm, and the latter phase where the algorithm will imitate Langevin dynamics due to the dominance of the injected noise. There are computations in [13] which check for the transition of the phases and the reader is encouraged to refer them. Under certain conditions (discussed in Appendix D), the algorithm converges to the true posterior.

Once the algorithm has entered its Langevin Dynamics phase, the samples of $\mathbf{W}$ along the convergence path can be considered to be samplings from the posterior. We can use the sampled weights to predict the target labels on the test set: the mean and variance of predictions can be obtained using network evaluations $\{f(\mathbf{W}_k)\}$ from the sampled network weights $\{\mathbf{W}_k\}$.

SGLD is easy to implement, but tuning the hyperparameters to get the the posterior samples is difficult. Also, the uncertainty prediction requires keeping the sampling of the weights in memory and thus wastes both time and memory at inference time.

6 Experiments

Dataset: We used samples drawn i.i.d. in the range [-4, 4] from a uniform distribution. The labels were generated using the function $y = x^3$ and i.i.d. Gaussian noise $\mathcal{N}(0, 3)$ was added to them.

6.1 Variational Inference

We tried out two different methods, Monte Carlo (MC) dropout [15] and Bayes by Backprop [11]. Gal [1] showed that MC dropout is a special case of VI with Bernoulli masks. Bayes by Backprop minimizes the ELBO using a Gaussian mixture prior and optimizing the mixture components.

Both methods predict low variance in regions where training data is present. As we move away from the training data, the variance increases. Hence, these methods predict more uncertainty when there is no nearby training data as opposed to a traditional neural network which can be overly confident. However, the true labels do not fall within three standard deviations of the predicted mean which suggests that more data is needed to get better uncertainty estimates.

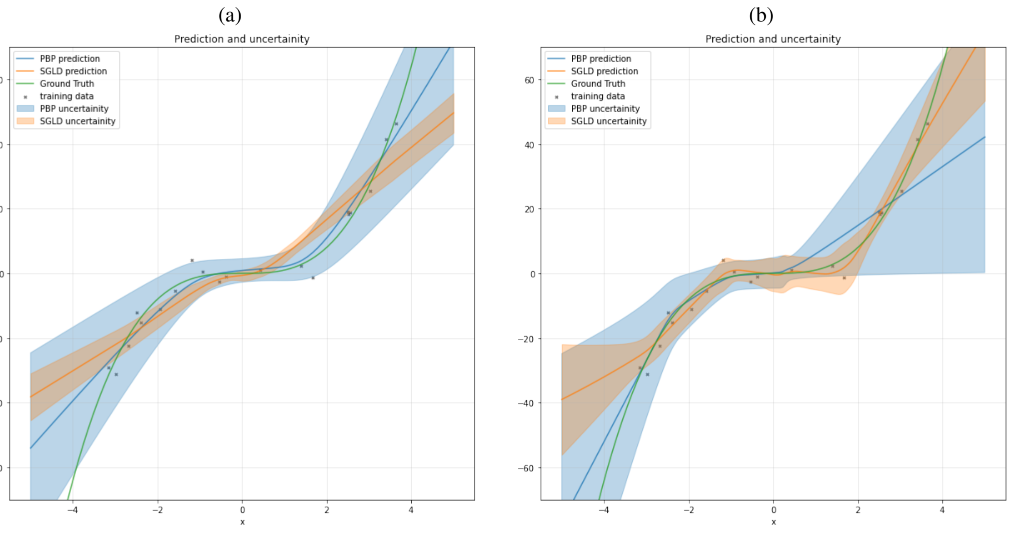


Figure 3: The mean prediction and uncertainty (3 std. deviations) of 2 network architectures trained on 20 training samples: (a) single hidden layer with 100 units; (b) two hidden layers with 50 units each.

6.2 Network Architecture Changes and Uncertainties

We use PBP and SGLD and vary the network architecture to understand the effect of underfitting and overfitting. The mean predictions and the captured uncertainties for two different network architectures are provided in figure 3. Figure 4 and 5 in the appendix contain more network architectures and a larger dataset.

The experiments confirm the uncertainty due to overfitting, and that depth of the network leads to more uncertainty than the width of the network. We can observe that PBP is less prone to underfitting and overfitting, while SGLD is prone to these phenomenon but reflects the behaviour in the uncertainty estimates.

7 Discussion

We reviewed Bayesian probabilistic modeling in the context of deep learning and looked at practical methods for inference. The methods can be plugged into existing deep neural networks directly to get predictive uncertainty estimates. However, all of them are sampling-based due to which training and inference times shoot up. These methods are thus suited for offline applications.

Through this project, we attempted to review important methods in Bayesian deep learning, from its early roots to the state-of-the-art methods in use currently. This will hopefully make future research in this area more accessible to readers.

8 Contributions

Each member of the project has contributed equally for the project.

9 Acknowledgment

The code for these experiments have been obtained from third party sources, listed here.

References

[1] Yarin Gal. “Uncertainty in deep learning”. In: *University of Cambridge* 1 (2016), p. 3.

[2] Alex Kendall and Yarin Gal. “What uncertainties do we need in bayesian deep learning for computer vision?” In: *Advances in neural information processing systems*. 2017, pp. 5574–5584.

[3] Dan Hendrycks and Kevin Gimpel. “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. In: *Proceedings of International Conference on Learning Representations* (2017).

[4] Gregory P. Meyer and Niranjan Thakurdesai. “Learning an Uncertainty-Aware Object Detector for Autonomous Driving”. In: *arXiv:1910.11375 [cs]* (2020). arXiv: `1910.11375 [cs]`.

[5] Ehsan Abbasnejad. “Uncertainty in Machine Learning”. University of Adelaide. 2018. URL: `https://cs.adelaide.edu.au/~javen/talk/ML05_Uncertainty_in_DL.pdf`.

[6] Andrey Malinin and Mark Gales. “Predictive Uncertainty Estimation via Prior Networks”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 7047–7058.

[7] Michael I Jordan. *The exponential family: Conjugate priors*. 2009.

[8] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. “Variational inference: A review for statisticians”. In: *Journal of the American statistical Association* 112.518 (2017), pp. 859–877.

[9] 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*. 1993, pp. 5–13.

[10] Alex Graves. “Practical variational inference for neural networks”. In: *Advances in neural information processing systems*. 2011, pp. 2348–2356.

[11] Charles Blundell et al. “Weight Uncertainty in Neural Networks”. In: *International Conference on Machine Learning*. 2015, pp. 1613–1622.

[12] José Miguel Hernández-Lobato and Ryan Adams. “Probabilistic backpropagation for scalable learning of bayesian neural networks”. In: *International Conference on Machine Learning*. 2015, pp. 1861–1869.

[13] 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)*. 2011, pp. 681–688.

[14] Radford M Neal et al. “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[15] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. 2016, pp. 1050–1059.

[16] Alex Kendall and Yarin Gal. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?” In: *arXiv:1703.04977*. Mar. 2017.

[17] Yongchan Kwon et al. “Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation”. In: (2018).

[18] Anoop Korattikara Balan et al. “Bayesian dark knowledge”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 3438–3446.

Appendices

A Aleatoric Uncertainty

Aleatoric uncertainty can be classified into two categories

- **Heteroscedastic Aleatoric Uncertainty:** A simple real life example of this uncertainty could be when we are measuring the amount of water consumed between morning 6am to 9pm when you sleep. If the measurement is done at 7am, 1pm, and 7pm, the amount of water consumed will generally be the same as people's routines remains stable. At 7am, it will be 1-1.5 glasses almost every day. At lunch, we might drink more water if we decide to go out and have a spicy lunch. As a lazy grad student, we might decide to work-out once each semester: the water intake that day will be abnormaly high at 7pm. As we see the variance in the number of glasses you drank varies with every input. This is exactly what Heteroscedastic Uncertainty is, which means that the noise is different for different inputs. Thus, it can be learned as a function of data given by:
$$\mathcal{L}_{\mathrm{NN}}(\theta)=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2\sigma\left(\mathbf{x}_i\right)^2}\left\|\mathbf{y}_i-\mathbf{f}\left(\mathbf{x}_i\right)\right\|^2+\frac{1}{2}\log\sigma\left(\mathbf{x}_i\right)^2 \tag{11}$$
We use MAP inference to find a single value of parameter theta and unlike epistemic uncertainty, do not use variational inference.As suggested by [16].
- **Homoscedastic Aleatoric Uncertainty** In this case the noise on every sample in the dataset is assumed to be the same.

A.1 Computing Aleatoric Uncertainty

$$\begin{aligned}\operatorname{Var}_q\left(p\left(y^*|x^*\right)\right)&=\mathbb{E}_q\left[y^*y^{*T}\right]-\mathbb{E}_q[y^*]\mathbb{E}_q[y^*]^T\\&=\int_\Omega\left[\operatorname{diag}\left(\mathbb{E}_{p(y^*|x^*,w)}\left[y^*\right]\right)-\mathbb{E}_{p(y^*|x^*,w)}\left[y^*\right]\mathbb{E}_{p(y^*|x^*,w)}\left[y^*\right]^T\right]q_\theta(w)dw\\&+\int_\Omega\left[\mathbb{E}_{p(y^*|x^*,w)}\left[y^*\right]-\mathbb{E}_{q_\theta(y^*|x^*)}\left(y^*\right)\right]\left[\mathbb{E}_{p(y^*|x^*,w)}\left[y^*\right]-\mathbb{E}_{q_\theta(y^*|x^*)}\left(y^*\right)\right]^Tq_\theta(w)dw\end{aligned}\tag{12}$$

Variance of equation (12) is the uncertainty that we are looking for. The proof of this can be viewed in [17]. The first term of the above equation is the aleatoric uncertainty. [17] proposes one method for estimating these values. Consider a Bayesian Neural Net for which we add a layer just before the activation to consist of the mean and variance of the logits. Once we learn the weights of this layer, the output of the layer would be a distribution with mean,variance = $\left(\hat{\mu}_t,\hat{\sigma}_t^2\right))$
and thus [16] proposes to estimate the uncertainty as

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T}\operatorname{diag}\left(\hat{\sigma}_t^2\right)}_{\text{aleatoric}}+\underbrace{\frac{1}{T}\sum_{t=1}^{T}\left(\hat{\mu}_t-\bar{\mu}\right)^{\otimes 2}}_{\text{epistemic}}\tag{13}$$

but [17] suggest certain drawbacks for the above method of estimation. And instead suggests a slight modification by fixing the last layer dimension. They suggest an estimation as follows.

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T}\operatorname{diag}\left(\hat{p}_t\right)-\hat{p}_t^2}_{\text{aleatoric}}+\underbrace{\frac{1}{T}\sum_{t=1}^{T}\left(\hat{p}_t-\bar{p}\right)^2}_{\text{epistemic}}\tag{14}$$

where we replace $\mathbb{E}(\mathbf{y}^*)$ in (12) as

$$\bar{p}=\sum_{t=1}^{1}\hat{p}_t/T\text{ and }\hat{p}_t=p\left(\hat{\omega}_t\right)=\operatorname{Softmax}\left\{f^{\omega_t}\left(x^*\right)\right\}\tag{15}$$

This helps us as we now no longer need to evaluate sigma. This is now nothing but the average variance thus, uncertainty arising from the noise of the dataset.

B ELBO derivation

The KL divergence between the variational distribution $q_\theta(\mathbf{W})$ and the true posterior $P(\mathbf{W}|\mathcal{D})$ can be expanded as follows:

$$\mathrm{KL}(q_\theta(\mathbf{W})||P(\mathbf{W}|\mathcal{D})) \tag{16}$$

$$= \int q_\theta(\mathbf{W}) \log \frac{q_\theta(\mathbf{W})}{P(\mathbf{W}|\mathcal{D})} \tag{17}$$

$$= \int q_\theta(\mathbf{W}) \log \frac{q_\theta(\mathbf{W})P(\mathbf{Y}|\mathbf{X})}{P(\mathbf{Y}|\mathbf{X},\mathbf{W})P(\mathbf{W})}\,\mathrm{d}\mathbf{W} \qquad \text{(using Bayes rule)} \tag{18}$$

$$= -\int q_\theta(\mathbf{W}) \log P(\mathbf{Y}|\mathbf{X},\mathbf{W})\,\mathrm{d}\mathbf{W} + \mathrm{KL}(q_\theta(\mathbf{W}), P(\mathbf{W})) + \log P(\mathbf{Y}|\mathbf{X}) \tag{19}$$

$$\implies \int q_\theta(\mathbf{W}) \log P(\mathbf{Y}|\mathbf{X},\mathbf{W})\,\mathrm{d}\mathbf{W} - \mathrm{KL}(q_\theta(\mathbf{W}), P(\mathbf{W})) + \mathrm{KL}(q_\theta(\mathbf{W})||P(\mathbf{W}|\mathcal{D})) = \log P(\mathbf{Y}|\mathbf{X}) \tag{20}$$

Since $\mathrm{KL}(q_\theta(\mathbf{W})||P(\mathbf{W}|\mathcal{D})) \geq 0$,

$$\int q_\theta(\mathbf{W}) \log P(\mathbf{Y}|\mathbf{X},\mathbf{W})\,\mathrm{d}\mathbf{W} - \mathrm{KL}(q_\theta(\mathbf{W}), P(\mathbf{W})) \leq \log P(\mathbf{Y}|\mathbf{X}) \tag{21}$$

C Stochastic Approximate Inference

Graves [10] proposed a method to scale ELBO maximization to large datasets. Instead of computing the expected log likelihood over the entire dataset, we can compute it over a random batch from the dataset:

$$\hat{\mathcal{L}}_{\mathrm{VI}}(\theta) = -\frac{N}{M}\sum_{i\in S}\int q_\theta(\mathbf{W}) \log P(\mathbf{y}_i|\mathbf{x}_i,\mathbf{W})\,\mathrm{d}\mathbf{W} - \mathrm{KL}(q_\theta(\mathbf{W}), P(\mathbf{W})) \tag{22}$$

where $|S| = M$. This is called data subsampling or mini-batch optimization.

D Stochastic Gradient Langevian Dynamics Convergence

The theoretical guarantee for convergence only exist when the step sizes $\{\eta_t\}$ are annealed such that:

$$\sum_t^\infty \eta_t = \infty \qquad\qquad \sum_t^\infty \eta_t^2 < \infty$$

A common sequence of step sizes to use is a polynomialy decaying sequence $a\,(b+t)^{-\gamma}, \gamma \in (0.5, 1]$. However, in practise one ones a constant step size once it has decreased below a critical level.

E Bayesian Dark Knowledge

In contrast to the variational Bayes, expectation propagation (e.g. probabilistic backpropagation), MCMC (e.g. SGLD), Korattikara et al. [18] introduced a novel approach to train a parametric model $\mathcal{S}\,(\mathbf{Y}|\mathbf{X},\mathbf{W})$ (*student network*) to approximate the Monte Carlo posterior predictive distribution $q\,(\mathbf{Y}|\mathbf{X})$ (*teacher network*).

SGLD [13] is used to compute posterior distribution of the weights $q(\mathbf{W}_t)$, and hence $q(\mathbf{Y}|\mathbf{X})$. Simultaneously, the student network is trained online to minimize the KL divergence between student and teacher network. If the student network has weights $\mathbf{W}_s$, the loss for a given input random variable $\mathbf{X}$ can be written as:

$$\begin{aligned}\mathcal{L}\left(\mathbf{W}_s|\mathbf{X}\right) &= KL\left(P\left(\mathbf{Y}|\mathbf{X},\mathcal{D}\right)||\mathcal{S}\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_s\right)\right)\\ &= -\mathbb{E}_{P(\mathbf{Y}|\mathbf{X},\mathcal{D})}\left[\log\mathcal{S}\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_s\right)\right] + \text{const}\\ &= -\int\left[\int P\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_t\right)P\left(\mathbf{W}_t|\mathcal{D}\right)\,\mathrm{d}\mathbf{W}_t\right]\log\mathcal{S}\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_s\right)\,\mathrm{d}\mathbf{Y}\\ &= -\int P\left(\mathbf{W}_t|\mathcal{D}\right)\left[\int P\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_t\right)\log\mathcal{S}\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_s\right)\,\mathrm{d}\mathbf{Y}\right]\,\mathrm{d}\mathbf{W}_t\\ &= -\int P\left(\mathbf{W}_t|\mathcal{D}\right)\mathbb{E}_{P(\mathbf{Y}|\mathbf{X},\mathbf{W}_t)}\left[\log\mathcal{S}\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_s\right)\right]d\mathbf{W}_t\end{aligned}$$

This integral is intractable, and SGLD (or other MCMC methods) can be used to approximate the integral:

$$\hat{\mathcal{L}}\left(\mathbf{W}_s|\mathbf{X}\right) = -\frac{1}{|\mathcal{W}|}\sum_{w\in\mathcal{W}}\mathbb{E}_{P(\mathbf{Y}|\mathbf{X},\mathbf{w})}\left[\log\mathcal{S}\left(\mathbf{Y}|\mathbf{X},\mathbf{W}_s\right)\right]$$

where $\mathcal{W}$ are the parameter samples from the teacher network. The loss is still a function of random variable $\mathbf{X}$. A dataset $\mathcal{D}'$ is created to train the student network on, the target probability distributions coming from the teacher network. The Monte Carlo approximation of the loss is:

$$\hat{\mathcal{L}}\left(\mathbf{W}_s\right) \approx \frac{1}{|\mathcal{W}|}\frac{1}{|\mathcal{D}'|}\sum_{\mathbf{w}\in\mathcal{W}}\sum_{\mathbf{x}'\in\mathcal{D}'}\mathbb{E}_{P(\mathbf{y}|\mathbf{x}',\mathbf{w})}\log\mathcal{S}\left(\mathbf{y}|\mathbf{x}',\mathbf{W}_s\right)$$

This distillation process captures the uncertainty in a very simple manner. The student networks are smaller than the original network and hence require less time and memory than other techniques.

F Practical VI Methods

Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be the parameters of the prior and the variational distribution respectively. The variational free energy, negative of the ELBO (equation 7), is given by:

$$\mathcal{F} = -\left\langle \ln\left[\frac{P(\mathcal{D}|\mathbf{W})P(\mathbf{w}|\boldsymbol{\alpha})}{Q(\mathbf{W}|\boldsymbol{\beta})}\right]\right\rangle_{\mathbf{W}\sim Q(\boldsymbol{\beta})} \tag{23}$$

F.1 Minimum Description Length

We can modify the above equation as:

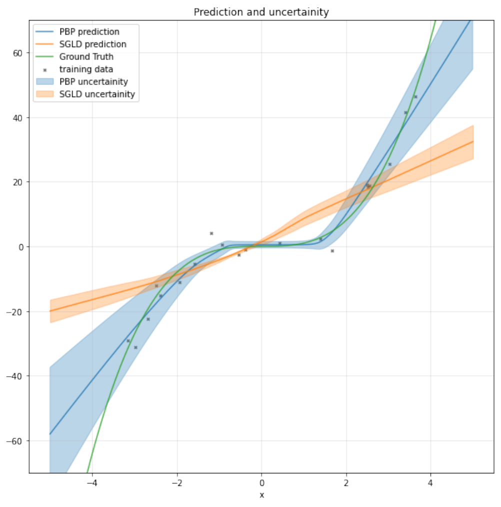
$$\mathcal{F} = \mathbb{E}_{\mathbf{W}\sim Q(\boldsymbol{\beta})}L^N\left(\mathbf{W},\mathcal{D}\right) + \mathrm{KL}\left(Q\left(\beta\right)||P(\alpha)\right) \tag{24}$$

where the first term is the expectation of the likelihood w.r.t. the variational approximation distribution and the second term measures the similarity between the approximated distribution and the prior. These two components can be restructured as L^E, the error loss, and L^C, the complexity loss.

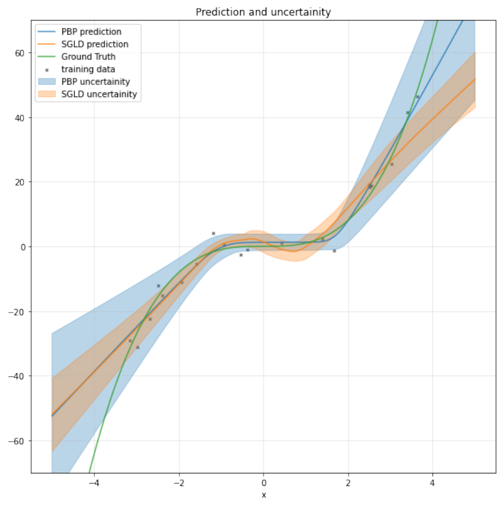
$$L(\boldsymbol{\alpha},\boldsymbol{\beta},\mathcal{D}) = L^E(\boldsymbol{\beta},\mathcal{D}) + L^C(\boldsymbol{\alpha},\boldsymbol{\beta}) \tag{25}$$

This loss can be viewed from an information theoretic standpoint. The error loss is the nats required to transmit the targets in $\mathcal{D}$ to a receiver who already has the inputs; the targets are computed from a network whose weights are sampled from the approximate distribution. The complexity loss is the nats required to transmit the network weights to a receiver who has the prior information. A combination of these two losses gives the total information content that needs to be transmitted to reach a prediction from just the input and the prior information on weights.

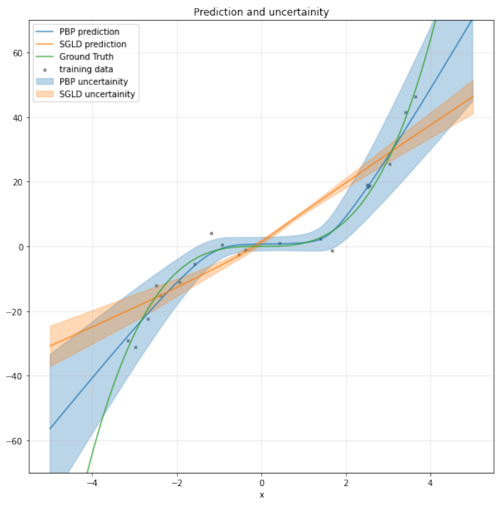
(a) [25]x1



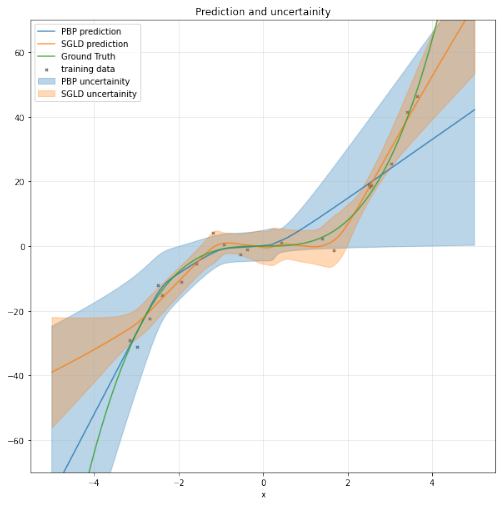
(b) [25]x2



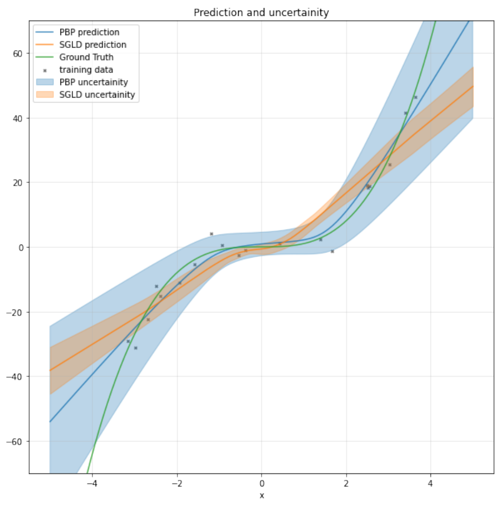
(c) [50]x1



(d) [50]x2



(e) [100]x1



(f) [100]x2

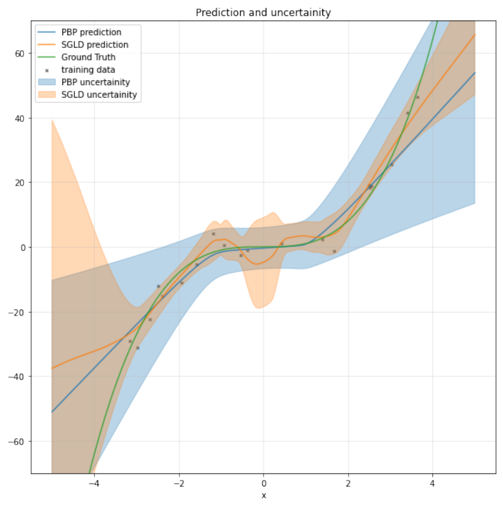
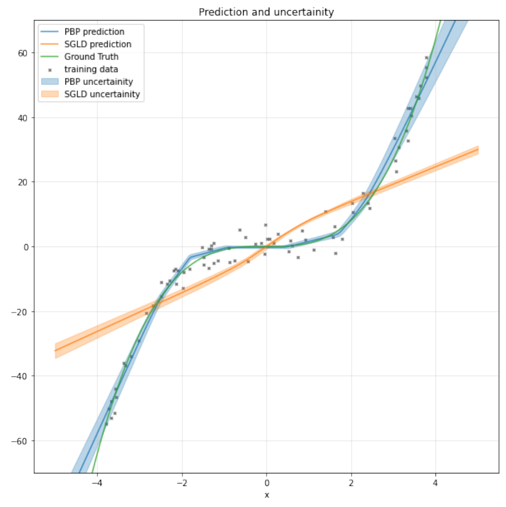
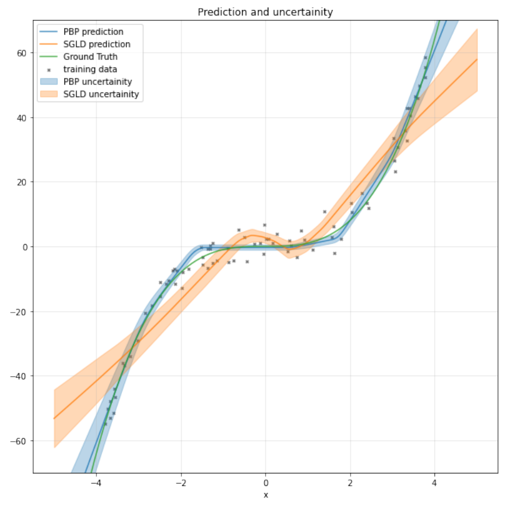


Figure 4: The mean prediction and uncertainty (3 standard deviations) of different network architectures trained on 20 training samples using uniform distribution in the data range. The plots (a)-(f) are captioned with the network architecture as *[units in one hidden layer]* x *number of hidden layers*.

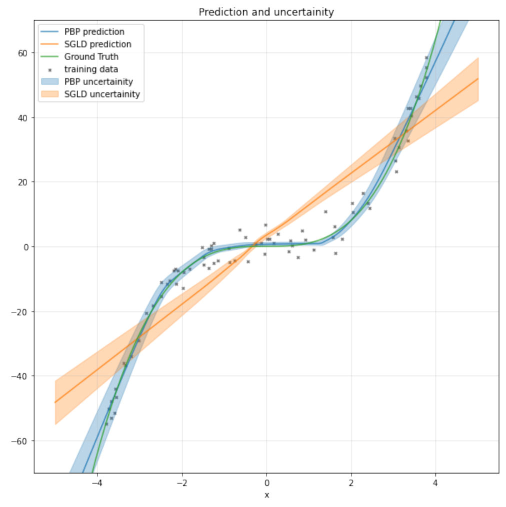
(a) [25]x1



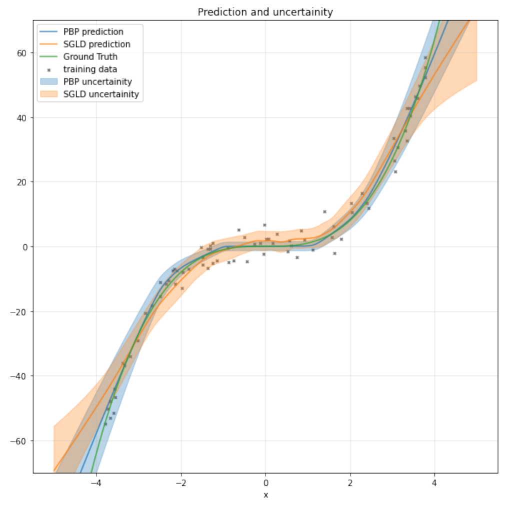
(b) [25]x2



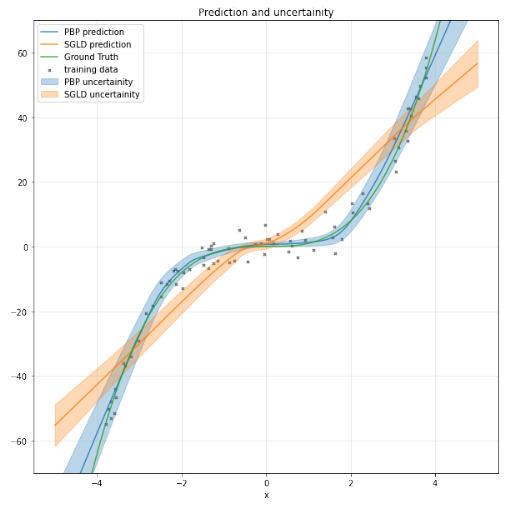
(c) [50]x1



(d) [50]x2



(e) [100]x1



(f) [100]x2

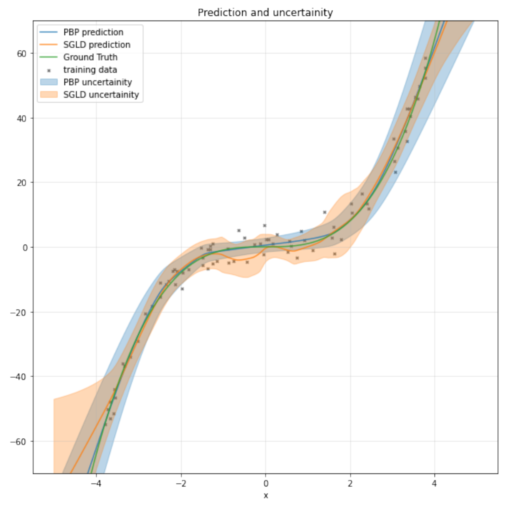
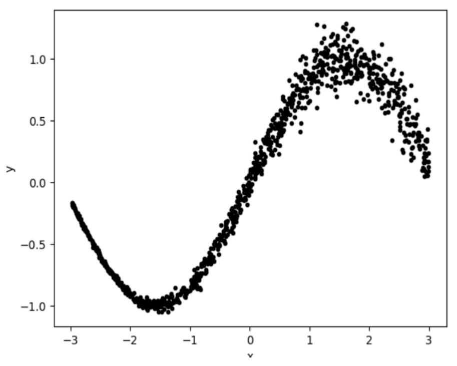
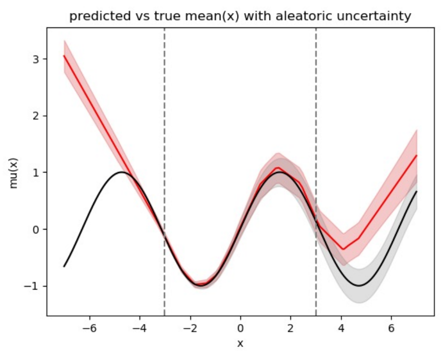


Figure 5: The mean prediction and uncertainty (3 standard deviations) of different network architectures trained on 80 training samples using uniform distribution in the data range. The plots (a)-(f) are captioned with the network architecture as *[units in one hidden layer]* x *number of hidden layers*.

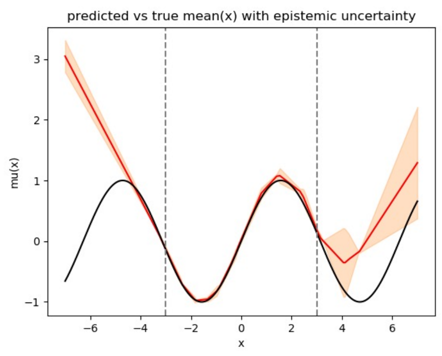
(a) Dataset for SGLD



(b) Predicted vs true mean(x) with aleatoric uncertainty



(c) Predicted vs true mean(x) with epistemic uncertainty



(d) Predicted vs true mean(x) with total uncertainty

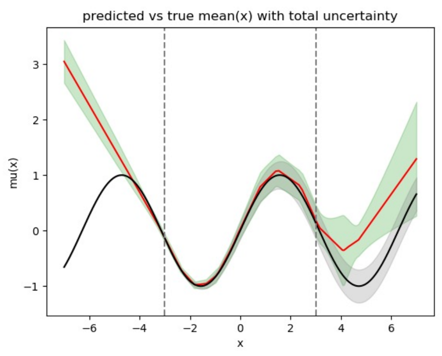


Figure 6: Different uncertainty estimates from the SGLD algorithm on a 3 hidden layer network, each layer containing 10 hidden units. The dataset has 1000 points.