CSC 2516: Partial Bayesian Neural Networks

Nathan Friedman & Cyrus Maz
Department of Statistical Sciences
University of Toronto

Abstract

We introduce a new neural network architecture called the *Partial Bayesian Neural Network*, which, in essence, is a hybrid of Bayesian and non-Bayesian neural networks. A Bayesian neural network applies the Bayesian paradigm to all the weights of a neural network, whereas the partial Bayesian neural network randomly designates a proportion of the network's weights to be Bayesian, and the rest to be non-Bayesian; this proportion is treated as a hyper-parameter of the model. A partial Bayesian neural network reduces the dimensionality of the posterior in comparison to a Bayesian neural network, and we found that ensembles of partial Bayesian neural networks can approximate Bayesian neural networks quite accurately.

1 Introduction

1.1 The BNN

Bayesian neural networks (BNNs) have been widely studied, and proven to be powerful tools. The main advantage of a BNN over a non-Bayesian neural network (NN) is that it accounts for uncertainty of the weights and hence the predictions it produces. It does so by treating the weights of the network as distributions, rather than single points. It then imposes a prior distribution on the weights, conditions on data, and arrives at a posterior distribution for the weights.

However, BNNs come at a cost: optimizing a BNN and deriving the posterior distribution of the weights is computationally expensive, and often even an intractable problem depending on the size of the network, choice of priors, and distributional assumptions on the data. Even in the simplest regression settings, approximating the posterior is an expensive endeavour.

1.2 The PBNN

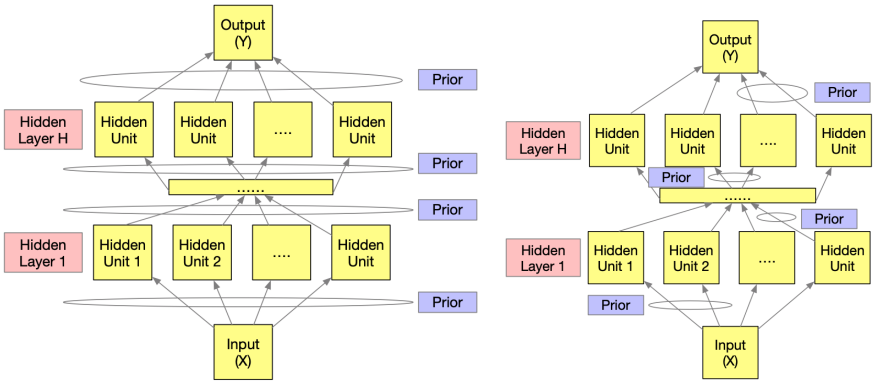
The Partial-Bayesian neural network (PBNN) strives to give performance comparable to the BNN but at a lower computational cost. The PBNN designates only some of the network's weights to

be distributional and the rest to be single points. In other words, only some of the weights are Bayesian. The proportion of the weights that are Bayesian is a hyper-parameter, allowing the network's "Bayesian-ness" to be adjustable.

1.3 The PBNN Ensemble

Given that the Bayesian/non-Bayesian weight designation process of the PBNN is entirely random, high model variance is a valid concern. To mitigate the potentially high model variance, we use an ensemble of PBNNs to make predictions. In other words, we train multiple PBNNs individually on the same data set, and average their predictions.

2 Figures



(a) Conventional Architecture (b) Random BNN

Figure 1: BNN vs. PBNN technique.

Figure 1 shows a conventional BNN next to a PBNN. The architecture shown here is arbitrary, and the main take away is that the weights (implied by the connections) have priors (implied by the circles) on them. In figure 1(a) all the weights have priors on them. This is typical for a BNN, although in some architectures all the weights of only certain layers have priors on them. What we are proposing is the architecture for figure 1(b). Instead of all the weights having priors on them, we will randomly select a subset of a fixed number of weights to have priors on them, while the remainder are single-point parameters and then generate predictions using that model. This will be repeated, and then the predictions will be averaged.

3 Formal Description

We explore the PBNN in the context of regression as regression motivates much of the statistical theory and machine-learning advances to date. Before delving into the details of PBNNs, a quick review of NNs and BNNs is in order. But first, we must establish some notation.

3.1 Notation

Training Data: $D = (X_i, Y_i), i = 1, ..., n$. The X_i's are the vectors explanatory variables. Y_i's are the response variables. $X_i \in \mathcal{X}, Y_i \in \mathcal{Y}$, where $\mathcal{X}$ is the space of the explanatory variable(s), and $\mathcal{Y}$ is the response space.

Network Weights: $\mathbf{W} = \mathbf{w} \cup \mathbf{w}^B$. Where $\mathbf{w}$ denotes the set of non-Bayesian weights, and $\mathbf{w}^B$ denotes the set of Bayesian weights.

Optimal non-Bayesian weights: $\mathbf{w}^*$. The set of non-Bayesian weights that optimize some cost function, J

Neural Network Function: $n : \mathcal{X} \to \mathcal{Y}$ n represents a neural network as a function that maps the predictor space to the response space. n is parameterized by a set of weights, $\mathbf{W}$. Depending on context we may or may not specify the parameterization when referring to n. But the output of n is always a prediction based on the input. $n(X) = n(X, \mathbf{W}) = \hat{Y}$.

Prior Distribution of the Weights: $P(\mathbf{W})$

Conditional Distribution of the Data Given Weights: $P(D|\mathbf{W})$

Posterior Distribution of the Weights Given the Data: $P(\mathbf{W}|D)$

3.2 NN

A typical neural network is a composed of layers and each layer is further composed of neurons. The first layer has the same dimension as X, and the last layer has the same dimension as Y. Any layer in between the first and last layer is known as a hidden layer, and the neurons in the hidden layers are known as hidden units. A neural network can have any number of hidden layers, and each hidden layer may have any number of hidden units.

Note that since the NN has no Bayesian weights, we have that $\mathbf{W} = \mathbf{w}$ (because $\mathbf{w}^B = \emptyset$).

The typical neural network for regression assumes a Gaussian distribution for the response, i.e. $P(Y|X, \mathbf{w}) \sim N$. Note that the parameterization of $P(Y|X, \mathbf{w})$ is non-trivial as it is a composite function of $\mathbf{w}$, and all the X's. In this setting, $\mathbf{w}^*$ can be calculated by Maximum Likelihood. We may set

$$\mathbf{w}^* = \mathbf{w}^{MLE} = \underset{\mathbf{w}}{\operatorname{argmax}} \log P(D|\mathbf{w})$$

Then to make a prediction, given some $\hat{X}$, we let $\hat{Y} = n(\hat{X}, \mathbf{w}^*)$

3.3 BNN

If we impose priors on the weights, $P(\mathbf{W})$, assume some distribution for the data conditional on the weights, $P(D|\mathbf{W})$, derive the posterior distribution of $P(\mathbf{W}|D)$ then our network can be (fully) Bayesian.

Note that since the BNN has no non-Bayesian weights, we have that $\mathbf{W} = \mathbf{w}^B$ (because $\mathbf{w} = \emptyset$).

In this scenario, given some $\hat{X}$, we let the prediction be the expectation of the neural network over the probability measure of the posterior distribution of the weights, i.e. $P(\mathbf{w}^B | D)$:

$$\hat{Y} = \mathsf{E}_{\mathbf{w}^B | D}\left[n(\hat{X}, \mathbf{w}^B)\right] \tag{1}$$

Note that $\mathbf{w}^B$ can have very high dimensionality, and thus $P(\mathbf{w}^B | D)$ may be too expensive if not downright impossible to calculate.

3.4 PBNN

The PBNN sets some of the weights to be Bayesian and the rest to be non-Bayesian. In other words, $\mathbf{W} = \mathbf{w} \cup \mathbf{w}^B$, and both $\mathbf{w}$ and $\mathbf{w}^B$ are non-empty. Just as in the BNN, the predictions are still the expectation of the neural network over the probability measure of the posterior distribution of the Bayesian weights; however, in the PBNN, the posterior is conditioned on D as well as $\mathbf{w}$ (i.e. the optimal non-Bayesian weights).

$$\hat{Y} = \mathsf{E}_{\mathbf{w}^B | D, \mathbf{w}}\left[n(\hat{X}, \mathbf{w}^B, \mathbf{w})\right] \tag{2}$$

The PBNN combines the BNN and NN into a hybrid network whose "Bayesian-ness" (i.e. $\frac{|\mathbf{w}^B|}{|\mathbf{W}|}$ is adjustable.)

4 Related Work and Bibliography

4.1 Weight Uncertaintly in Neural Networks

Our work builds on the work done by Blundell, et al. (2015) from Google DeepMind in their paper 'Weight Uncertainty in Neural Networks'. This paper focused on training neural networks where the weights had probability distributions. Blundell, et al. propose using a combination of variational inference, the reparameterization technique, and Monte-Carlo sampling to find an optimal approximation to the posterior. They call their methodology *Bayes by Backprop*. We use some of the same ideas as Blundell, et al., but we modified them to work in situations where only a portion of the parameters are Bayesian.

Specifically, Blundell, et al. makes predictions to approximate (1), whereas our predictions approximate (2). In the former, $\mathbf{w}$ is empty whereas in our model, both $\mathbf{w}^B$ and $\mathbf{w}$ are non-empty.

Blundell, et al. approximate the posterior of $\mathbf{w}^B | D$ through variational inference as follows:

$$= \text{argmin}\mathrm{KL}\left[q(\mathbf{w}^B | D, \quad) || P(\mathbf{w}^B | D)\right] \tag{3}$$

where $q(\mathbf{w}^B | D, \quad)$ is the approximate posterior, and parameterizes q. On the other hand, in addition to approximating the posterior of $\mathbf{w}^B | D$ through variational inference, we find the optimal non-Bayesian weights $\mathbf{w}$. That is, we find $\mathbf{w}$ and such that:

$$= \text{argmin}\mathrm{KL}\left[q(\mathbf{w}^B | D, \mathbf{w} \ , \quad) || P(\mathbf{w}^B | D, \mathbf{w})\right] \tag{4}$$

$$\mathbf{w} \quad = \underset{\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{\mathbf{w}^B | D, \mathbf{w}}\left[J(\mathbf{w} \cup \mathbf{w}^B)\right] \tag{5}$$

for some cost function J .

Lastly, after finding , Blundell, et al. makes predictions by approximating

$$\hat{Y} = \mathsf{E}_{q(\mathbf{w}|D,\ )}\left[n(\hat{X}, \mathbf{w})\right] \tag{6}$$

using Monte-Carlo. On the other hand, we make predictions using an ensemble of M PBNNs. Specifically, our predictions are of the following form:

$$\hat{Y} = \frac{1}{M}\sum_{m=1}^{M} \mathsf{E}_{q(\mathbf{w}^B_{(m)}|D,\ _{(m)},\mathbf{w}_{(m)})}\left[n(\hat{X}, \mathbf{w}^B_{(m)}\ [\ \mathbf{w}_{(m)})\right] \tag{7}$$

where $\mathbf{w}^B_{(m)}$ is the set of randomly designated Bayesian weights in the m-th PBNN in the ensemble. For each ensemble, the proportion of $\mathbf{w}^B$ to $\mathbf{w}$ is predetermined as a hyperparameter.

4.2 Other related works

Blei et al. (2017) give a detailed review of variational inference. We found Blei et al's paper illuminating in approximating the posterior of the weights in both BNNs using equation (3) and adapting the methodology to PBNNs.

Kingma et al. (2013) outlines the reparameterization trick and more in their paper, 'Auto-Encoding Variational Bayes'. This paper goes into detail about applying the reparameterization trick to approximate the gradient using Monte-Carlo sampling in order to find .

5 Comparison and Demonstrations

5.1 Fitting a PBNN: Partial-Bayes by Backprop

Fitting a PBNN has the added challenge of optimizing $\mathbf{w}$ (i.e. the non-Bayesian weights). This may seem like a daunting task at first, but under certain assumptions this is in fact not much different than *Bayes by Backprop* as outlined by Blundell, et al.

Let $q(\mathbf{w}^B|D, \mathbf{w},\ )$ be the *approximate posterior* distribution of $\mathbf{w}^B|D, \mathbf{w}$ as parameterized by . We will assume a distributional family for $q(\mathbf{w}^B|D,\ )$ and choose a that minimizes the Kullback-Leibler divergence between the approximate posterior and the true posterior. Then

$$= \text{argmin}\mathrm{KL}\left[q(\mathbf{w}^B|D, \mathbf{w},\ )||P(\mathbf{w}^B|D, \mathbf{w})\right] \tag{8}$$

$$= \text{argmin}\int q(\mathbf{w}^B|D, \mathbf{w},\ ) \log\frac{q(\mathbf{w}^B|D, \mathbf{w},\ )}{P(\mathbf{w}^B|D, \mathbf{w})}dw \tag{9}$$

$$= \text{argmin}\ \mathsf{E}_q\Big[\log q(\mathbf{w}^B|D, \mathbf{w},\ ) \quad \log\left[P(\mathbf{w}^B)P(D|\mathbf{w}\ \ [\ \mathbf{w}^B)\right]\Big] \tag{10}$$

Typically, for variational inference we would use the reparameterization trick and find an optimal that allows us to approximate the posterior. Then, to make predictions for a new $\hat{X}$, we would sample from $q(\mathbf{w}|\ )$ to obtain a Monte-Carlo estimate of $\hat{Y}$ as in (6). On the other hand, we have the

added complication of having non-Bayesian weights, $\mathbf{w}$ which also need to be optimized. Using the properites of the logarithm, equation (10) can be re-written as:

$$= \text{argmin } \mathsf{E}_q\Big[\log q(\mathbf{w}^B | D, \mathbf{w}, \quad) \quad \log P(\mathbf{w}^B) \quad \log P(D|\mathbf{w} \quad [\ \mathbf{w}^B)]\Big] \tag{11}$$

Note that since we approximate the posterior with q:

$$\mathbf{w} \quad = \underset{\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{\mathbf{w}^B|D,\mathbf{w}}\big[J\ (\mathbf{w}|D, \mathbf{w}^B)] \tag{12}$$

$$\underset{\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{q(\mathbf{w}^B|D,\mathbf{w},\quad)}\big[J\ (\mathbf{w}|D, \mathbf{w}^B)] \tag{13}$$

for some cost function J . We assume a distributional family for q such that the reparameterization trick (see appendix for details) is feasible. Additionally, we specify the cost function to be the negative log-likelihood. Lastly, we assume that the approximate posterior q is such that $\mathbf{w}^B$ is independent of $\mathbf{w}$ (i.e.
$q(\mathbf{w}^B|D, \mathbf{w}, \quad) = q(\mathbf{w}^B|D, \quad)$). Then our problem boils down to:

$$\mathbf{w} \quad = \underset{\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{q(\mathbf{w}^B|D,\mathbf{w},\quad)}\big[J\ (\mathbf{w}|D, \mathbf{w}^B)] \tag{14}$$

$$= \underset{\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{q(\mathbf{w}^B|D,\quad)}\big[\quad \log P(D|\mathbf{w} \ [\ \mathbf{w}^B)\big] \tag{15}$$

$$= \underset{\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{q(\mathbf{w}^B|D,\quad)}\big[\quad \log P(D|\mathbf{w} \ [\ \mathbf{w}^B) \quad \log P(\mathbf{w}^B) + \log q(\mathbf{w}^B|D, \quad ^*)] \tag{16}$$

Under the assumptions on our cost function and approximate posterior, we have that the objective in (16) is identical to the objective in equation (11). This implies that:

$$, \mathbf{w} \quad = \underset{,\mathbf{w}}{\text{argmin}}\ \mathsf{E}_{q(\mathbf{w}^B|D,\ )}\big[\quad \log P(D|\mathbf{w} \ [\ \mathbf{w}^B) \quad \log P(\mathbf{w}^B) + \log q(\mathbf{w}^B|D, \quad)] \tag{17}$$

This implies that we can optimize our non-Bayesian weights alongside the variational inference parameterization . Thus, in order to optimize , we can still use the reparameterization trick.

Letting $f(\mathbf{w}^B, \mathbf{w}, \quad) = \quad \log P(D|\mathbf{w} \ [\ \mathbf{w}^B) \quad \log P(\mathbf{w}^B) + \log q(\mathbf{w}^B|D, \quad)$, and $\mathbf{w}^B = \mathbf{t}(\ ,\ )$, where $\kappa(\ )$ we will have all the ingredients we need to apply to reparameterization technique (see appendix). We can estimate the gradient using Monte-Carlo sampling as follows:

$$r_{\mathbf{w},\ } \mathsf{E}_q\big[f(\mathbf{w}^B, \mathbf{w}, \quad] = \mathsf{E}_\kappa\big[\, r_{\mathbf{w},\ } f(t(\ ,\ ), \mathbf{w}, \quad)] \tag{18}$$

$$\frac{1}{M}\sum_{n=1}^{M} r_{\mathbf{w},\ } f(t(\ ,\ ^{(i)}), \mathbf{w}, \quad), \quad \text{where } ^{(i)} \quad \kappa(\ )(i.i.d). \tag{19}$$

where M is the total number of Monte Carlo samples, and $^{(i)}$ indicates the i-th Monte Carlo sample from $\kappa(\ )$. We then use gradient descent to optimize with respect to $\mathbf{w}$ and . That is:

$$\big[\mathbf{w}^{(p)}, \quad ^{(p)}] \quad [\mathbf{w}^{(p\ 1)}, \quad ^{(p\ 1)}] \quad \alpha\frac{1}{M}\sum_{n=1}^{M} r_{\mathbf{w},\ } f(t(\ ^{(p\ 1)}, \ ^{(i)}), \mathbf{w}^{(p\ 1)}, \quad ^{(p\ 1)}) \tag{20}$$

This is analogous to the fitting method from the (fully) Bayesian counterpart. The only difference is that we are optimizing for $\mathbf{w}$ in addition to , but remarkably we can use the same technique.

5.2 Experiment

5.2.1 Set up

In order to investigate the merit of our model, we regress on toy data. The model used has a single input, 2 hidden layers with 40 hidden units in each, and a single output. The architecture is shown in figure 2. The network is fully connected, with biases which leads to a total of 1761 weights.

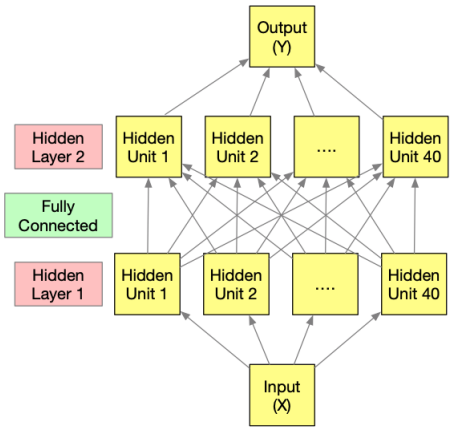


Figure 2: Architecture of models used for tbe experiments.

Six different generated data sets were used to train and test the model. The functions were:

$$f_1(x) = x + \gamma \tag{21}$$

$$f_2(x) = \frac{x^2}{10} + \gamma \tag{22}$$

$$f_3(x) = 3\sin(x) + x + \gamma \tag{23}$$

$$f_4(x) = 3\log(x^2) + \gamma \tag{24}$$

$$f_5(x) = 6\sin(\frac{x^2}{15}) + 6\cos(\frac{x^2}{15}) + \gamma \tag{25}$$

$$f_6(x) = \frac{x^2}{10}\sin(\frac{x^2}{20} + x) + \frac{x}{20} + \cos(x) + \gamma \tag{26}$$

where $\gamma \quad N(0, \sigma = 0.5)$ is a noise term. We sampled 40 data points on the x interval [-15,15] from the equations above. For each of the six generated data sets, we fitted one BNN, one NN, and ensembles of 10%, 25% and 50% PBNNs, with 10, 6, and 3 PBNNs in each ensemble, respectively. We used a learning rate of $\alpha = 0.005$ for all models, with $N = 600$ iterations of gradient descent with Adam for all functions except f_5, which we iterated 1000 times.

We also imposed a diagonal Gaussian prior on the Bayesian weights: $P(\mathbf{w}^B) = N(\mathbf{0}, \exp(\ \ 5)\mathbf{I})$ for mathematical convenience. Furthermore, we used a diagonal Gaussian as the approximate posterior, $q(\mathbf{w}^B j\ \ )$ in order to apply the reparameterization trick.

5.2.2 Results

The results from fitting all the models were quite positive. Figure 3 shows four of the 10 PBNNs in the 10% ensemble for function 3. From figure 3 it is clear that they all fit the data quite well, but in slightly different ways. The differences between the fits are due to i) the randomness involved in the Bayesian/non-Bayesian weight designation, and ii) the natural variation that comes from Monte Carlo sampling used for approximating the expectation in (7).

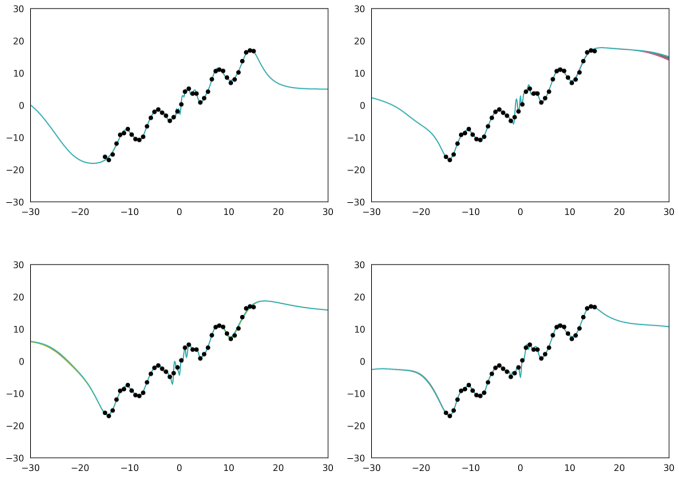


Figure 3: Members of the 10% PBNN ensemble, fitted on data generated from function 3.

Figure 4 shows the fitted models for the non-Bayesian NN, the BNN, as well as the 10%, 25%, and 50% ensembles for all six functions. Figure 4 illustrates how well the PBNN ensembles approximate the (full) BNN. With some exceptions, the fitted ensemble models appear to resemble the BNN more than the non-Bayesian NN. In general, it appears that the higher the proportion of Bayesian weights, the more the ensemble predictions resembles the BNN predictions. That being said, even the 10% PBNN ensemble does a good job of emulating the BNN, especially when interpolating.

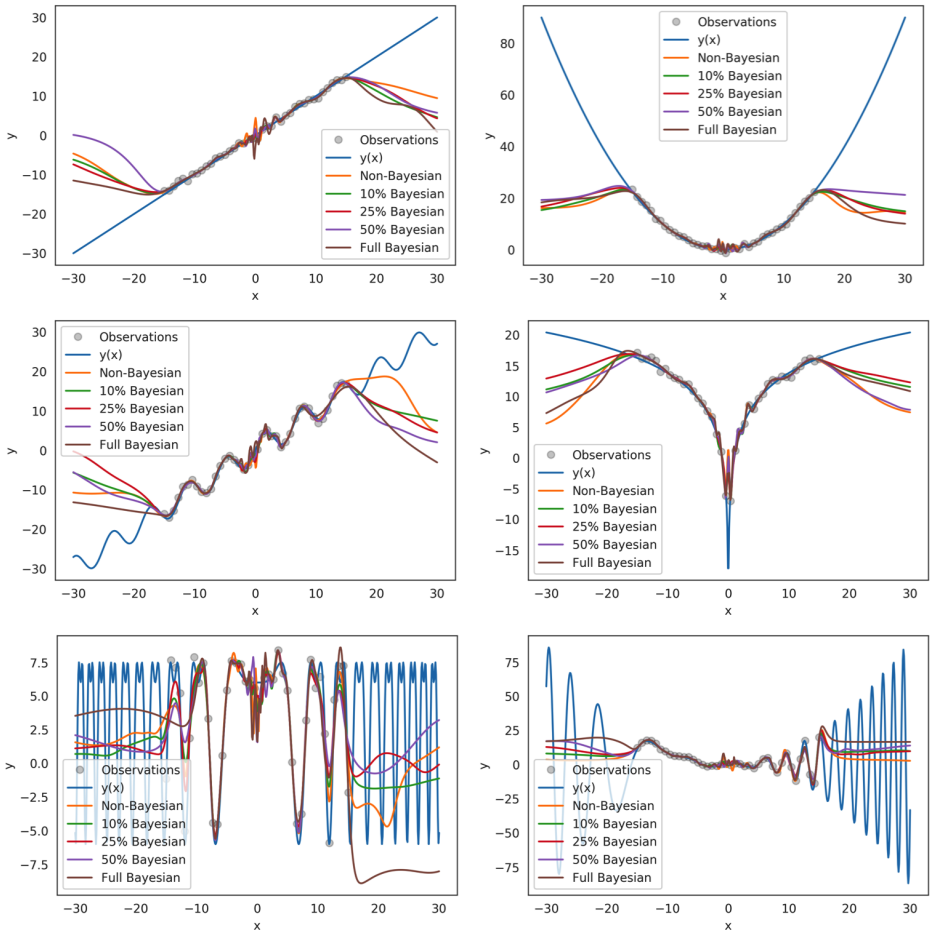


Figure 4: Fitted models vs. actual data and training observations. Top left: f_1, Top right: f_2, Middle left: f_3, Middle right: f_4, Bottom left: f_5, Bottom right: f_6

6 Limitations

There are three main limitations of our work: i) it was tested only on relatively simple models, ii) the prior on the Bayesian weights was diagonal Gaussian, and iii) the analysis done was largely qualitative.

Since it was only tested on simple models, there is no guarantee that it will scale well to more complicated problems. Having multidimensional inputs, sequential inputs, multiple outputs, or anything that increases the complexity of the model is not guaranteed to yield the same quality of results. The second limitation is the choice of prior. We used a diagonal Gaussian, which makes the Bayesian weights independent of each other as well as the non-Bayesian weights. Using a more complicated prior, or one that has the non-Bayesian weights as a parameter could lead to much more tedious calculations. Lastly, the analysis we did was qualitative, and a quantitative measure of how well the ensembles emulate the BNN may provide some key insights. Beyond those three limitations, the performance of the ensembles appear to be impressive, and estimate the BNN rather well.

7 Conclusion

When going about a regression problem, PBNN ensembles are a sound option. Specifying an appropriate cost function, and using a convenient approximate posterior make model training manageable. In fact, under these conditions training a neural-network with both Bayesian and non-Bayesian weights is nearly identical to training a fully Bayesian neural network, with the added advantage of having much lower posterior dimensionality. The performance of the PBNN ensembles were not exceptionally different from that of the BNN, which is suggestive of potential for the PBNN ensemble as a means to approximate BNNs.

8 References

[1] Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424.*

[2] Blei, D. M., Kucukelbir, A., McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association, 112*(518), 859-877.

[3] Kingma, D. P., Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114.*

9 Appendix

9.1 The Reparameterization Trick

For a random variable, w, if (i) $w = t(\theta, \epsilon)$ where ϵ is a random variable with density $\kappa(\epsilon)$ and $t(\theta, \epsilon)$ is a deterministic function, and (ii) w has density $q(w|\theta)$ such that $\kappa(\epsilon)d\epsilon = q(w|\theta)dw$ then for a function $f(w, \theta) = f(t(\theta, \epsilon), \theta)$ we have:

$$\frac{\partial}{\partial\theta}\mathsf{E}_q[f(w,\theta)] = \mathsf{E}_\kappa[\frac{\partial}{\partial\theta}f(t(\theta,\epsilon),\theta)] \tag{27}$$

$$= \mathsf{E}_\kappa\big[\frac{\partial f(t(\theta,\epsilon),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon)}{\partial\theta} + \frac{\partial f(t(\theta,\epsilon),\theta)}{\partial\theta}\big] \tag{28}$$