Regression with Bayesian Neural Networks

Emad Zadegan
emad.zadegan@mail.utoronto.ca
emadzadegan.github.io

2018-12-14

Abstract

While standard neural networks are capable of great feats in regression and classification, their preidctions are given as point estimates, and they lack the ability to give a measure of uncertainty on their predictions. Bayesian Neural Networks remedy this by introducing the Bayesian paradigm to neural networks. In this paper, we construct a Bayesian Neural Network as outlined in [1] from the ground up and use it for regression. Note that no new ideas are presented in this paper – the aim of this paper is detailed mathematical development for a deeper understanding of Bayesian Neural Networks by the reader (and writer).

Notation

Training Data: $\mathcal{D}$

$\mathcal{D} = (X_i, Y_i)$, $i = 1, ..., n$,

X_i's are the vectors explanatory variable(s). 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.

Neural Network Function: $\mathfrak{n} : \mathcal{X} \to \mathcal{Y}$

$\mathfrak{n}$ represents a neural network as a function that maps the predictor space to the response space. $\mathfrak{n}$ is parametrized by a set of weights, w. Depending on context we may or may not specify the parametrization when referring to $\mathfrak{n}$. But the output of $\mathfrak{n}$ is always a prediction based on the input. $\mathfrak{n}(X) = \mathfrak{n}(X, w) = \hat{Y}$. Furthermore $\mathfrak{n} \in \mathfrak{N}$ where $\mathfrak{N}$ represents the function space of neural networks.

Neuron: $\eta_{i,m} : \mathbb{R}^{[m-1]} \to \mathbb{R}$

$\eta_{i,m}$ is the i-th neuron in the m-th layer. It maps the output space of layer [m-1] to a real number. Let $X^{[m-1]}$ be the output from layer [m-1]. Then $\eta_{i,m}\big(X^{[m-1]}\big) = \gamma(X^{[m-1]} \cdot w_{i,m} + b_{i,m})$ where γ is a nonlinear *activation* function, and $b_{i,m}$ and $w_{i,m}$ are respectively the scalar *bias* vector and weight vector for $\eta_{i,m}$. Both $w_{i,m}$ and $b_{i,m}$ are contained in w.

Optimal Weights: w^*

w^* is the set of weights for $\mathfrak{n}$, which minimize some loss function, L.

Loss Function: $L : (\mathcal{X}, \mathcal{Y}, \mathfrak{N}) \to \mathbb{R}$

Neural Network For Regression: The Standard Approach

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.

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

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

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

Neural Network For Regression: The Semi-Bayesian Approach

The standard approach above becomes Bayesian if we impose a prior for the weights: $P(w)$. Then we have:

$$w^* = w^{Bayes} = \underset{w}{\operatorname{argmax}} \log \big[P(w|\mathcal{D})\big] = \underset{w}{\operatorname{argmax}} \log \big[P(\mathcal{D}|w)P(w)\big] = \underset{w}{\operatorname{argmax}} \big[\log P(\mathcal{D}|w) + \log P(w)\big]$$

And $\hat{Y} = \mathfrak{n}(\hat{X}, w^{Bayes})$

Note that the computation of w^{Bayes} and w^{MLE} differ only by the prior term, $\log P(w)$. Using a Gaussian prior for the weights makes this approach equivalent to the standard approach but with L2 regularization.

Neural Network For Regression: The Bayesian Approach

If we take advantage of the posterior *distribution* of $w|\mathcal{D}$ to make our predictions $\hat{Y}$, rather than just using the single point w^{Bayes}, then our approach becomes truly Bayesian.

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 $w|\mathcal{D}$:

$$\hat{Y} = \mathbb{E}_{w|\mathcal{D}}\big[\mathfrak{n}(\hat{X}, w)\big] \tag{1}$$

Calculating the above expectation is nontrivial and also intractable as w has *very* high dimensionality in any reasonably-sized neural network.

But the posterior can be approximated by variational inference.

Approximating the Posterior with Variational Inference

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

$$\theta^* = \underset{\theta}{\text{argmin}} \text{KL}\big[q(w|\mathcal{D},\theta)||P(w|\mathcal{D})\big] \tag{2}$$

$$= \underset{\theta}{\text{argmin}} \int q(w|\mathcal{D},\theta)\log\frac{q(w|\mathcal{D},\theta)}{P(w|\mathcal{D})}dw \tag{3}$$

$$= \underset{\theta}{\text{argmin}} \int q(w|\mathcal{D},\theta)\log\frac{q(w|\mathcal{D},\theta)}{P(w)P(\mathcal{D}|w)}dw \tag{4}$$

$$= \underset{\theta}{\text{argmin}} \int q(w|\mathcal{D},\theta)\log q(w|\mathcal{D},\theta)dw - \int q(w|\mathcal{D},\theta)\log\big[P(w)P(\mathcal{D}|w)\big]dw \tag{5}$$

$$= \underset{\theta}{\text{argmin}}\ \mathbb{E}_q\Big[\log q(w|\mathcal{D},\theta) - \log\big[P(w)P(\mathcal{D}|w)\big]\Big] \tag{6}$$

where $\mathbb{E}_q$ indicates expectation taken with respect to the approximate posterior distribution.

If we can differentiate the $\mathbb{E}_q\Big[\log q(w|\mathcal{D},\theta) - \log\big[P(w)P(\mathcal{D}|w)\big]\Big]$ with respect to θ, then we can use gradient descent or some variant of it to approximate θ^*.

There exists an assortment of theorems and techniques that permit differentiation of an integral/expectation with relative ease in a variety of situations. The one we will utilize is known as the *reparameterization trick* [1], [2].

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:

$$\frac{\partial}{\partial\theta}\mathbb{E}_q[f(w,\theta)] = \mathbb{E}_\kappa[\frac{\partial}{\partial\theta}f(t(\theta,\epsilon),\theta)] \tag{7}$$

$$= \mathbb{E}_\kappa[\frac{\partial f(t(\theta,\epsilon),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon)}{\partial\theta} + \frac{\partial f(t(\theta,\epsilon),\theta)}{\partial\theta}] \tag{8}$$

Let $f(w,\theta) = \log q(w|\mathcal{D},\theta) - \log\big[P(w)P(\mathcal{D}|w)\big]$. And let $q(w|\mathcal{D},\theta)$ be diagonal Gaussian; then we have $\theta = (\mu,\sigma)$, and $w = t(\theta,\epsilon) = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0,1)$; thus we have all the ingredients to apply the reparameterization technique.

Then we will estimate (8) by Monte Carlo sampling:

$$\frac{\partial}{\partial\theta}\mathbb{E}_q[f(w,\theta)] = \mathbb{E}_\kappa[\frac{\partial f(t(\theta,\epsilon),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon)}{\partial\theta} + \frac{\partial f(t(\theta,\epsilon),\theta)}{\partial\theta}] \tag{9}$$

$$= \mathbb{E}_\kappa\Big[\begin{pmatrix}\frac{\partial f(t(\theta,\epsilon),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon)}{\partial\mu} + \frac{\partial f(t(\theta,\epsilon),\theta)}{\partial\mu}\\ \frac{\partial f(t(\theta,\epsilon),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon)}{\partial\sigma} + \frac{\partial f(t(\theta,\epsilon),\theta)}{\partial\sigma}\end{pmatrix}\Big] \tag{10}$$

$$\approx \frac{1}{M}\sum_{i=1}^{M}\begin{pmatrix}\frac{\partial f(t(\theta,\epsilon^{(i)}),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon^{(i)})}{\partial\mu} + \frac{\partial f(t(\theta,\epsilon^{(i)}),\theta)}{\partial\mu}\\ \frac{\partial f(t(\theta,\epsilon^{(i)}),\theta)}{\partial t}\frac{\partial t(\theta,\epsilon^{(i)})}{\partial\sigma} + \frac{\partial f(t(\theta,\epsilon^{(i)}),\theta)}{\partial\sigma}\end{pmatrix} \tag{11}$$

where M is the total number of Monte Carlo samples, and $\epsilon^{(i)}$ indicates the i-th Monte Carlo sample from $\mathcal{N}(0,1)$.

We will also take $P(w)$ to be diagonal Gaussian. And we will assume $Y_i \sim P(Y_i|X_i, w)$ is Gaussian and iid for all $i = 1, ..., n$, which implies that $P(\mathcal{D}|w)\prod_{i=1}^{n} P(Y_i|X_i, w)$ is univariate Gaussian.

Finding θ^*

We will use gradient descent to find θ^*. Let α be the step size, and N be the number of iterations gradient descent we will perform. We will assume that for sufficiently large N, we have $\theta^* \approx \theta_N$. Start with some initial θ_0; then for $p = 0, 1, ..., N$, set

$$\theta_p = \theta_{p-1} - \alpha \frac{\partial}{\partial \theta} \mathbb{E}_q[f(w, \theta)]$$

Using the Monte Carlo gradient estimation from above, we have:

$$\theta_p = \begin{pmatrix} \mu_p \\ \sigma_p \end{pmatrix} = \begin{pmatrix} \mu_{p-1} \\ \sigma_{p-1} \end{pmatrix} - \alpha \frac{1}{M} \sum_{i=1}^{M} \begin{pmatrix} \frac{\partial f(t(\theta_{p-1}, \epsilon^{(i)}), \theta_{p-1})}{\partial t} \frac{\partial t(\theta_{p-1}, \epsilon^{(i)})}{\partial \mu} + \frac{\partial f(t(\theta_{p-1}, \epsilon^{(i)}), \theta_{p-1})}{\partial \mu} \\ \frac{\partial f(t(\theta_{p-1}, \epsilon^{(i)}), \theta_{p-1})}{\partial t} \frac{\partial t(\theta_{p-1}, \epsilon^{(i)})}{\partial \sigma} + \frac{\partial f(t(\theta_{p-1}, \epsilon^{(i)}), \theta_{p-1})}{\partial \sigma} \end{pmatrix} \tag{12}$$

Regression with a Bayesian Neural Network (BNN)

We train a BNN with two hidden layers, each with 40 neurons, to regress on a handful of different functions:

(1) $y(x) = x + \phi$,
(2) $y(x) = \frac{x^2}{10} + \phi$,
(3) $y(x) = 3\sin(x) + x + \phi$,
(4) $y(x) = 3\log(x^2) + \phi$,
(5) $y(x) = 10/x + phi$,
(6) $y(x) = \frac{x^2}{10}\sin(\frac{x^2}{20} + x) + \frac{x}{20} + \cos(x) + \phi$

Where $\phi \sim \mathcal{N}(0, \sigma = 0.25)$ is a noise term. The X's are 50 points that split [-10,10] into equal distances. $\alpha = 0.01$, $M = 10$, and $N = 500$.

We use a radial basis activation function ($rbf(x) = e^{-x^2}$), and we use Autograd [3] to calculate gradients and train the BNN.

In the plots below, the black points are the training samples. For each function, the plot on the left shows several predictions, each coloured line is the prediction on [-20,20] corresponding to one randomly drawn w from the approximate posterior. The plot on the right gives an aggregated prediction from all the predictions on the left plot, as well as a 95% prediction inteval.

Predictions that lie within the training domain are generally good, but outside the training domain they are not. In fact, the 95% confidence intervals outside the training domain do not include the true values for any of the functions

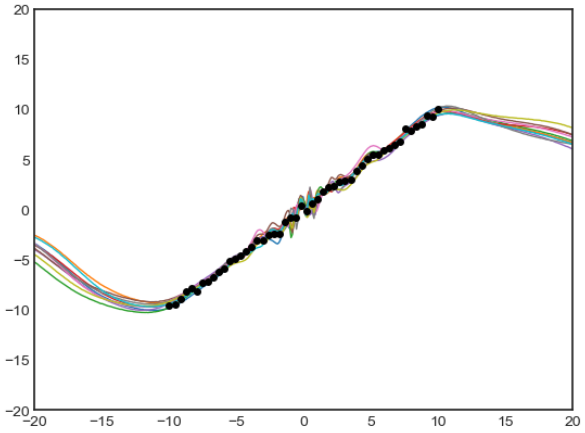
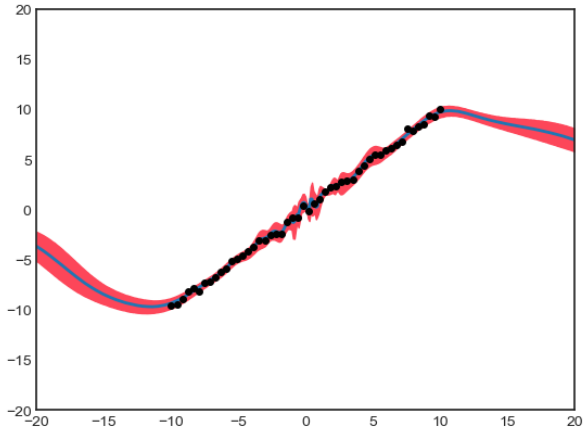
References

[1] Weight Uncertainty in Neural Networks: Blundell et al. https://arxiv.org/pdf/1505.05424.pdf

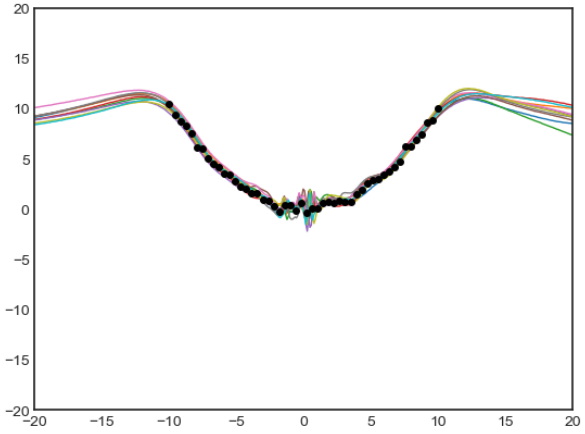
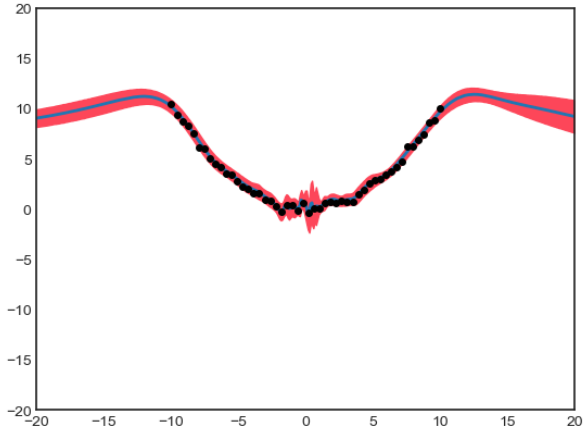
[2] Auto-Encoding Variational Bayes: Kingma and Welling https://arxiv.org/pdf/1312.6114v10.pdf

[3] Autograd: https://github.com/HIPS/autograd

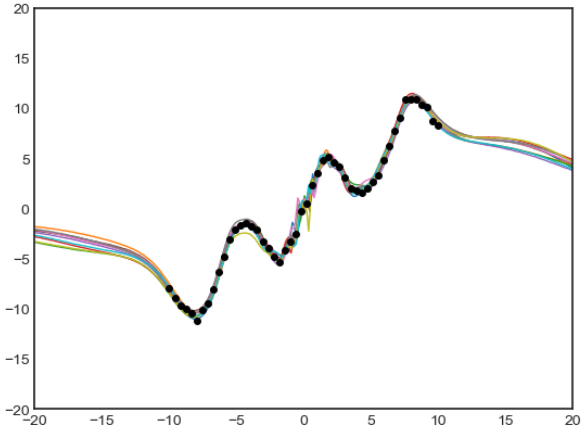
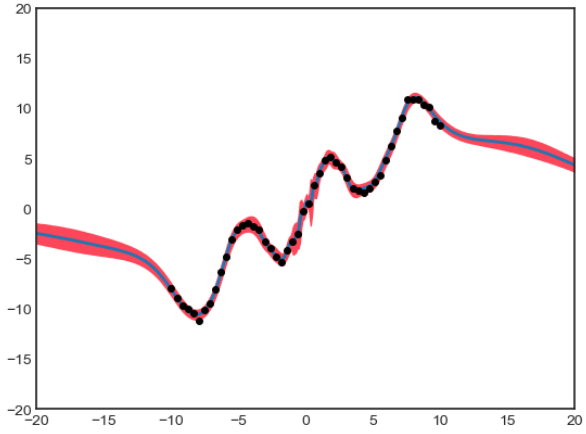
(1) $y = x + \phi$

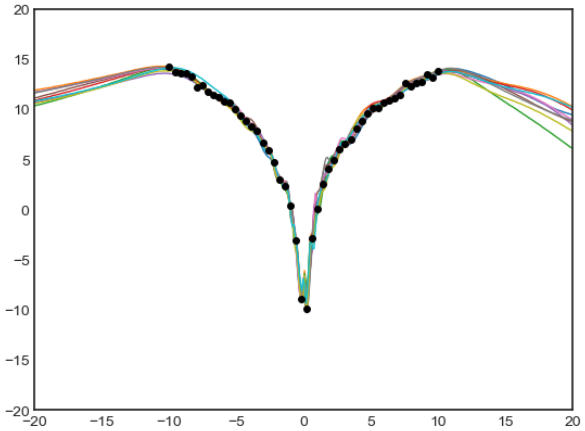
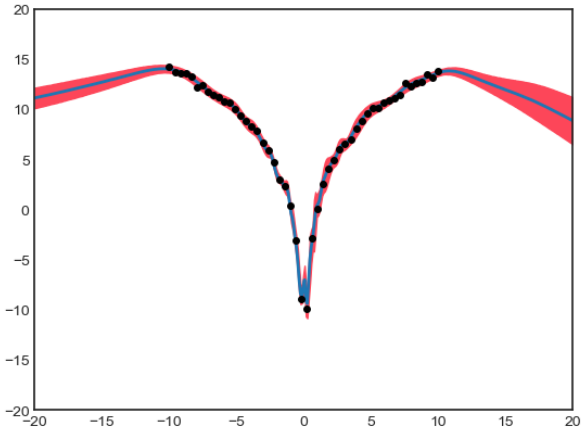
(2) $y = \frac{x^2}{10} + \phi$

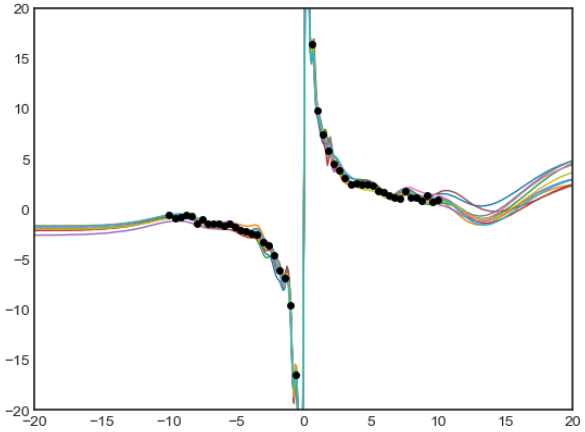
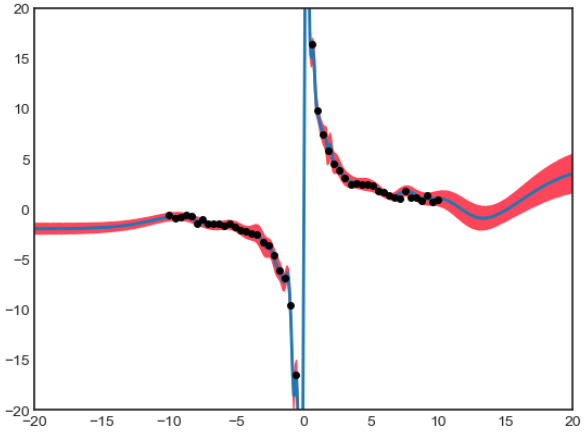
(3) $y = 3\sin(x) + x + \phi$

(4) $y = 3\log(x^2) + \phi$

(5) $y = \frac{10}{x} + \phi$

(6) $y(x) = \frac{x^2}{10}\sin(\frac{x^2}{20} + x) + \frac{x}{20} + \cos(x) + \phi$