Bayesian Inference in Deep Learning

Final Project for Math 153 - Bayesian Statistics

Zihao Xu

May 6, 2018

Contents

1 Introduction

In recent years, there has been a huge revival in using deep learning techniques, represented by neural networks, to solve complicated problems. We have witnessed unprecedented success in the applications of neural networks in a variety of tasks, including image recognition, natural language understanding and development of autonomous cars.

However, one critique that the standard feed-forward natural networks often receives is they are prone to over-fitting (Blundell et al., 2015). Furthermore, when applied to supervised learning tasks, the standard natural networks is unable to produce useful measures of uncertainty that inform confidence in predictions or results (Blundell et al., 2015). In light of such deficiencies, several scholars (Neal, 1995)(Miller et al., 2015)(Blundell et al., 2015)(Tran et al., 2016) have attempted to incorporate methods from the Bayesian paradigm that could place distributions over the weights of the network. These type of neural networks with uncertainty over the weights are commonly known as Bayesian Neural Networks (BNN).

There are multiple advantages to BNN that the standard neural networks do not have: regularization over the weights to prevent over-fitting, posterior inference on the weights for prediction, and useful measures of uncertainty attributed to unseen predictor space or heteroskedasticity. However, there are several challenges associated with training BNN, including selecting a reasonable prior, approximating the loss function, performing gradient descent over the loss function and etc.

In Chapter 2, I will briefly overview the Bayesian paradigm, reflecting on the merits of Bayesian methods and briefly introducing a class of techniques called variational methods. Then in Chapter 3, I will introduce the theoretical foundations of neural networks and its limitations. After that, I will discuss the Bayesian Neural Network (BNN) in Chapter 4 and the crucial concepts that make the training of BNN possible. In Chapter 5, I will first apply BNN to a toy example to graphically illustrate its ability to capture uncertainties in the data. Then I will apply the standard neural networks as well as BNN to a classification problem and compare their performances. Finally, I will conclude with some further thoughts in Chapter 6.

2 The Bayesian Paradigm

2.1 Bayesian Statistics

Bayesian Statistics and its methods are largely characterized by the use of prior knowledge in conjunction with new data in producing a posterior distribution of the parameters of interest, which then could be used in inference or prediction. The calculation of posterior distribution depends on the classic Bayes' Rule:

$$p(\theta|x) = \frac{f(x|\theta)p(\theta)}{\int f(x|\theta)p(\theta)d\theta} \tag{1}$$

Since the denominator of equation 1 is the marginal probability of $f(x)$, a constant, the posterior distribution, $p(\theta|x)$, could also be expressed as:

$$p(\theta|x) \propto f(x|\theta)p(\theta) \tag{2}$$

This is to say that our posterior depends on two quantities: $p(\theta)$, the prior distribution of θ that encodes our prior beliefs about θ, and $f(x|\theta)$, the likelihood of the data given θ. The combined result of such formulation is that the posterior distribution is biased towards our prior beliefs, in contrast to the method of Maximum Likelihood Estimation (MLE) employed by Frequentists, which only uses the likelihood of data.

In simpler situations, the prior and posterior distributions over θ satisfy conjugacy, i.e. the kernel of the posterior distribution is directly recognizable and the same as that of the prior distribution. However, when more variables are involved or more complicated distributions are required in modelling, we have to apply methods such as the Metropolis-Hastings Algorithm or the Gibbs Sampler to simulate random draws from the posterior to approximate the posterior distribution, and then perform inferences on it.

There are several advantages to the Bayesian framework. First, it provides statisticians a principled way of incorporating prior knowledge with data. Depending on the confidence we have on our prior intuition (if at all), Bayesians could choose parameters that encodes their prior beliefs on expected value or certain conditions on the CDF of the parameters of interest. Second, parameters of the distribution on θ directly indicates level of confidence. As an example, when using Beta Distribution as a prior to model p, the parameter of Bernoulli trials, one could choose small/large values of α and β to express low/high level of confidence, or $\alpha = \beta = 1$ to indicate no prior information is available. Third, credible intervals obtained from posterior distribution provide interpretable answers, such as over a lifetime of experiments, the true parameter has a probability of 0.95 of falling in a 95% credible interval, compared to its counterpart, Confidence Interval in a Frequentist setting, which is hard to explain. Fourth, depending on the loss function we use, the Bayesian will give a clear answer that could minimize the Bayes risk: posterior mean is used to minimize squared error loss, while posterior median is used to minimize absolute error loss.

However, as a counter-argument, one may point out that specifying a prior is often a hard task, as it is not easy to find a suitable distribution, and our prior beliefs might also seem baseless. To address this problem, Bayesians proposed the Hierarchical Model in which hyper-priors of prior could be calculated and specified to inform the choice of priors. Such methods could make the Bayesian models more structured.

2.2 Variational Bayesian Methods

Within the Bayesian paradigm, variational Bayesian methods is a class of techniques for approximating intractable integrals arising in Bayesian inference and machine learning.

There are two general genres of variational Bayesian methods. The first set of variational methods are used to approximate the posterior distribution of unobserved variables (parameters or latent variables). These group of methods could serve as an alternative to the Markov Chain Monte Carlo sampling methods like Gibbs Sampler. The second set of variational methods are purposed to derive a lower bound for the marginal likelihood (sometimes called the "evidence") of the observed data[1]. Higher marginal likelihood for a given model indicates a better fit of the data by that model and hence a greater probability that the model in question was the one that generated the data. The second use of Bayesian variational learning is relevant to our construction of Bayesian Neural Networks, which will be further discussed in Chapter 4.

3 Deep Learning

Deep Learning is a subclass of machine learning models based on learning data representations. It is represented by deep neural networks (Haykin, 1998), convolutional neural networks (CNN), recurrent neural networks (RNN) and etc. Recently, the academia has witnessed several breakthroughs achieved by applications of deep learning in computer vision, medical image analysis, natural language processing and bioinformatics. For example, in the famous ImageNet image classification challenge, CNN models built by researchers obtained error rate lower than that of a human being:

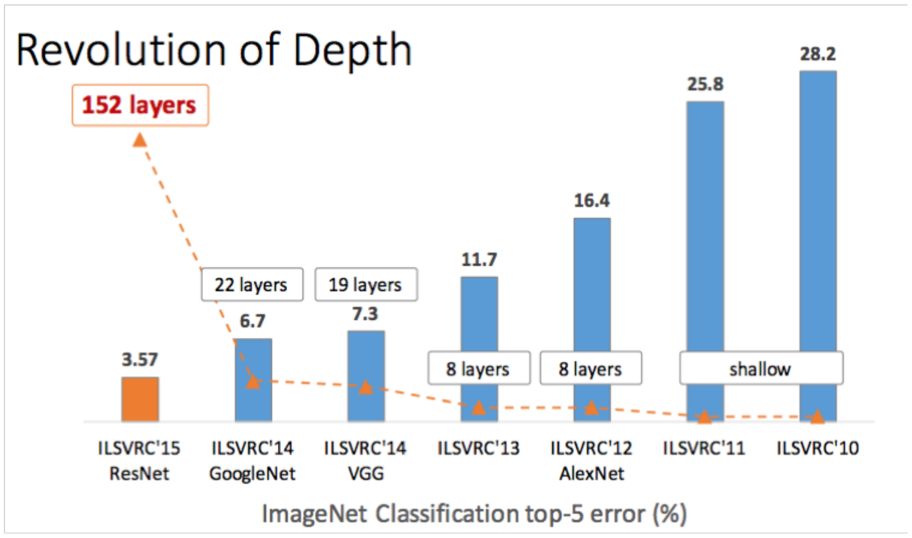

Figure 1: Error Rate by Year for the Large Scale Visual Recognition Challenge

In the following sections, I will briefly overview the fundamentals of neural networks as well as its Frequentist nature. Then, I will discuss its limitations, which will pave the way for our introduction of Bayesian Neural Networks.

3.1 Neural Networks

For the purpose of our discussion, we view neural networks as a probabilistic model, $P(y|X, w)$, where y is the response variable (categorical or continuous), $X \in \mathbb{R}^d$ are the predictors and w refers to the weights (and biases) within a Neural Network. For classification (y is categorical), $P(y|X, w)$ is a discrete probability mass function and is often related with cross-entropy loss function. Given

[1]Definition obtained from Wikipedia: `https://en.wikipedia.org/wiki/Variational_Bayesian_methods`

a vector $p \in \mathbb{R}^k$ representing the predicted probability for each class, where k is the number of classes, and $y \in \mathbb{R}^k$ is a vector of zeros except the index of true label, which equals one:

$$cross\_entropy(p, y) = -\sum_{i=1}^{k} y_i log(p_i) \tag{3}$$

For regression, $P(y|X, w)$ is assumed to be Gaussian (Blundell et al., 2015), which is associated with the loss function of Mean Squared Error:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (\hat{y_i} - y_i)^2 \tag{4}$$

where $\hat{y_i}$ and y_i represent predicted and true value of y for observation i.

With the notation in place, we will briefly go over the structure of a typical feed-forward, multi-layer perceptron (MLP) neural network.

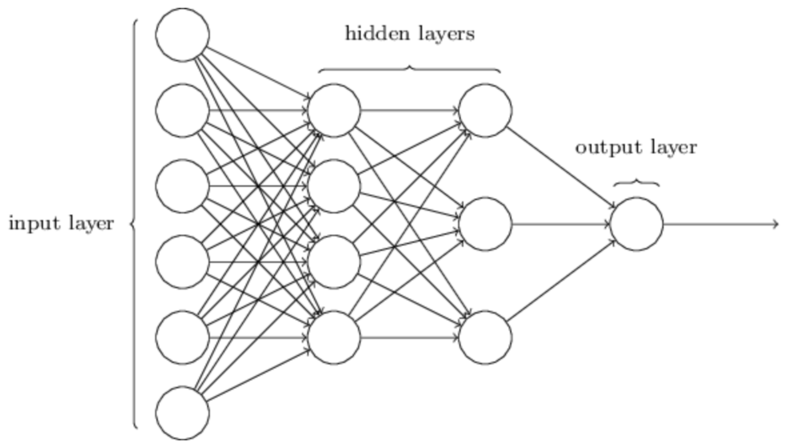

Figure 2: Multi-Layer Perceptron network

As shown in Figure 2, a neural network typically consisted of three types of layers: input layer, hidden layer(s) and output layer. As the name suggests, input layer takes in the input (often a numeric value) and passes on the value to the hidden layer. Each hidden layer consists of a fixed number of perceptrons (aka neurons or hidden units), each having a weight term, w, and a bias term, b. The output for a single perceptrons is calculated as $output = x^T \cdot w + b$. The number of perceptrons per layer and the number of layers are decided by the designer of the neural network and can in fact be treated as hyper-parameters for the neural network that directly affects its performance. In addition to the calculation of perceptrons, we typically apply some form of non-linear transformations to the output of each hidden unit so that the neural network as a whole is able to model complex and non-linear functions that encode the true relationships between the input and output. Some popular choices include rectified linear unit ($ReLU$, see Equation 5) and $tanh$ (see Equation 6).

$$ReLU(a) = max\{(a, 0)\} \tag{5}$$

$$tanh(a) = \frac{sinh(a)}{cosh(a)} = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{6}$$

Finally, the prediction, or output of the network, is calculated in the output layer. The number of units in the output layer depends on the specific task, i.e. regression (one unit representing the predicted value) or classification (the same of units and classes, representing predicted probability for each class).

3.2 Point Estimation for Neural Networks

In typical training of neural networks, we randomly initialize the weights and biases of the neurons within the network (typically standard normal is used) and use a method called back-propagation to optimize the weights and biases so that the chosen loss function is minimized. The exact mathematical derivation of back-propagation is beyond the scope of this project, but in Equation 7, we give the general formulation of the way to update weights and biases using their partial gradients with respect to the loss function:

$$w_k \to w'_k = w_k - \alpha \frac{\partial C}{\partial w_k} \qquad b_l \to b'_l = b_l - \alpha \frac{\partial C}{\partial b_l} \tag{7}$$

In this equation, C denotes the loss function, w and b denote the weight and bias for a particular neuron, and α refers to the learning rate (i.e. the rate at which the model descends towards a minimum of the loss function). In practice, instead of using all the data at once to calculate the gradient, the training is done using randomly selected mini-batches of training data with a constant size wherein the gradient is approximated using the average of the gradients of the data within a mini-batch. This method is know as Stochastic Gradient Descent, which yields an unbiased estimate of the gradients.

The weights and biases learned using such framework is equivalent to the Frequentist MLE estimates (Blundell et al., 2015), given by:

$$\begin{aligned} w^{MLE} &= \underset{w}{\text{argmax}} \;\; log(P(\mathcal{D}|w)) \\ &= \underset{w}{\text{argmax}} \; \sum_i \; log(P(y_i|X_i, w)) \end{aligned}$$

where $\mathcal{D} = (X, y)$ represents the training data.

The above setup forms the vanilla version of neural networks, which is in itself useful for a variety of regression and classification tasks. For more specified and complicated tasks like image classification, more advanced models and transformations like CNN and RNN are employed.

3.3 Limitations

Albeit the amazing results achieved by neural networks, this framework is not without its limitations.

First, the success of neural networks largely depends upon the collection of enormous data sets. In the ImageNet case, the data used for training includes 150,000 labeled photographs from 1000 classes. Had there only been few images for training, the trained neural networks would not be able to capture the deep features within the images and would then generalize poorly to unforeseen images due to over-fitting on the sparse training images. Besides, black-box like neural networks do not provide us with a useful measure of uncertainty in that the predictions are given in the form of point mass (a single number in regression or a vector of probabilities over classes in classification). In light of such deficiencies, I will transition to talk about the Bayesian Neural Networks (Neal, 1995), a form of neural network that makes use of Bayesian variational methods for training and allows for posterior inferences.

4 Bayesian Neural Network (BNN)

4.1 Overview

Bayesian Neural Network typically uses standard normal as prior distributions of weights and biases and calculates the posterior distributions of weights and biases (together represented as w for simplicity) using the training data ($\mathcal{D}$). We denote the posterior distribution as $P(w|\mathcal{D})$. The learned distribution can then be employed to make predictions on unforeseen observation: for an observation with unknown label y and predictors X, the prediction $\hat{y}$ is produced by a forward pass through the trained BNN network, using weights (w^*) corresponding to the specified loss function (posterior mean for MSE loss and posterior median for MAE loss):

$$\hat{y} = BNN(X, w^*) \tag{8}$$

Alternatively, we could leverage the entire posterior distribution of weights and biases to achieve something quite phenomenal: predicting by taking expectations (Blundell et al., 2015) over the weights, given by:

$$\hat{y} = \mathbb{E}_{P(w|\mathcal{D})}\hat{y}|\hat{X}, w = \int BNN(\hat{X}, w)P(w|\mathcal{D})dw \tag{9}$$

which is equivalent to taking an ensemble of uncountably infinite networks (Blundell et al., 2015). Despite the theoretical elegance, BNN are hard to train in practice because the calculation of posterior distribution over every single weight and bias is computational prohibitive. Thus, we would like apply Bayesian variational methods to approximate a lower bound for what is known as the Kullback-Leibler (KL) Divergence, which will be further discussed below.

4.2 Bayesian Backpropagation in Practice

As mentioned above, exact Bayesian inference on the weights of a neural network is intractable as the number of parameters is very large and the functional form of a neural network does not allow easy integration for weights update (Miller et al., 2015). This is why we need to find a variational approximation to the posterior distribution on the weights. In particular, the usage of variational approximation attempts to find the distribution over w, $Q(w|\theta)$, that minimizes the Kullback-Leibler (KL) Divergence.

4.2.1 Kullback–Leibler (KL) Divergence

The Kullback-Leibler (KL) Divergence is a numeric measure of the difference between two distributions. For two probability distributions Q and P, the KL divergence from Q to P in a discrete case is defined to be:

$$D_{KL}(Q||P) = \sum_i Q(i) log \frac{Q(i)}{P(i)} = \mathbb{E}_Q[logQ(i) - logP(i)] \tag{10}$$

while in a continuous case,

$$D_{KL}(Q||P) = \int_{-\infty}^{\infty} q(x) log \frac{q(x)}{p(x)} dx = \mathbb{E}_{q(x)}[log\ q(x) - log\ p(x))] \tag{11}$$

As we can see from Equations 10 and 11, KL divergence calculates the expected log differences in between two distributions with respect to distribution Q.

This measure of difference in two distributions is useful for the training of BNN, since it could serve as a loss function to be minimized. More specifically, we would like to find a variational approximation ($Q(w|\theta)$) to the posterior distribution ($P(w|\mathcal{D})$) on the weights of BNN. Theoretically, we want to find the best parameters, θ^*, of $Q(w|\theta)$ such that the KL divergence is minimized:

$$\begin{aligned}\theta^* &= \underset{\theta}{\text{argmin}}\, KL[Q(w|\theta)||P(w|\mathcal{D})] \\ &= \underset{\theta}{\text{argmin}}\, \mathbb{E}_{Q(w|\theta)}[logQ(w|\theta) - logP(w|\mathcal{D})]\end{aligned}$$

However, this optimization is intractable as it directly depends on the posterior distribution, $P(w|\mathcal{D})$. This is why we will introduce another quantity called the Evidence Lower Bound (ELBO) with which we could deduce a lower bound for KL divergence.

4.2.2 The Evidence Lower Bound (ELBO)

First, we consider the quantity $logP(\mathcal{D})$:

$$\begin{aligned}logP(\mathcal{D}) &= log \int P(\mathcal{D}, w) dw \\ &= log \int P(\mathcal{D}) \int Q(w|\theta) d\theta dw \\ &= \int \int logP(\mathcal{D}) Q(w|\theta) d\theta dw \\ &= \int \int log \frac{P(\mathcal{D})P(w|\mathcal{D})}{P(w|\mathcal{D})} Q(w|\theta) d\theta dw \\ &= \int \int log \frac{P(w, \mathcal{D})}{P(w|\mathcal{D})} Q(w|\theta) d\theta dw\end{aligned}$$

$$
\begin{aligned}
&= \int\int [log\frac{P(w,\mathcal{D})}{Q(w|\theta)} + log\frac{Q(w|\theta)}{P(w|\mathcal{D})}]Q(w|\theta)d\theta dw \\
&= \int\int log\frac{P(w,\mathcal{D})}{Q(w|\theta)}Q(w|\theta)d\theta dw + \int\int log\frac{Q(w|\theta)}{P(w|\mathcal{D})}Q(w|\theta)d\theta dw \\
&= E_{Q(w|\theta)}[logP(w,\mathcal{D}) - logQ(w|\theta)] + D_{KL}[Q(w|\theta)||P(w|\mathcal{D})]
\end{aligned}
$$

Since $logP(\mathcal{D})$ is the logarithm of the marginal likelihood of the data, it is a fixed constant (this quantity is also known as the model evidence). In other words, we could minimize $D_{KL}(Q||P)$ by maximizing the Evidence Lower Bound:

$$
ELBO(\theta) = E_{Q(w|\theta)}[logP(w,\mathcal{D}) - logQ(w|\theta)] \tag{12}
$$

It is important to notice that,

$$
ELBO(\theta) = E_{Q(w|\theta)}[logP(w,\mathcal{D})] - E_{Q(w|\theta)}[logQ(w|\theta)]
$$

and thus maximizing ELBO is equivalent to maximizing the first term, $E_{Q(w|\theta)}[logP(w,\mathcal{D})]$, which encourages the weights to fit the data well, and minimizing $E_{Q(w|\theta)}[logQ(w|\theta)]$, which discourages the distribution Q become too concentrated, i.e. overfitting.

In order to maximize ELBO, we need to calculate the gradient of ELBO with respect to θ, denoted as Δ_θ. According to Mill et al.(2015), the gradient inherits the ELBO's form as an expectation, which is in general an intractable quantity to compute. Therefore, Mill et al. introduced *reparameterization gradient estimators* (RGEs) computed using the reparameterization trick to tackle this problem.

4.2.3 Reparameterization

So far, we have been using $Q(w|\theta)$ to represent the variational distribution of the weights of BNN. This form of representation can be transformed so that the calculation of gradients becomes tractable. For example, for $w \sim \mathcal{N}(\mu, diag(\sigma^2))$, we could rewrite $\sigma = log(1 + exp(\rho))$ so that σ is non-negative (this is called the $softplus$ transformation, which we would use in the experiments below). With this, $w = \mu + log(1 + exp(\rho)) \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$ and $\odot$ represents an element-wise product. As such, we have:

$$
\begin{gathered}
\theta = [\mu, \rho] \\
\epsilon \sim \mathcal{N}(0, I) \\
w \sim \mathcal{T}(\epsilon, \theta)
\end{gathered}
$$

In this way we have separated the original distribution of w into two parts, $\epsilon \sim \mathcal{N}(0, I)$, which is independent of the variational parameters θ, and $\mathcal{T}(\epsilon, \theta)$ which is dependent on $\theta = [\mu, \rho]$. Under this setup, each step of optimization (we will minimize -ELBO, which is equivalent to maximizing ELBO) on a particular data point, x, will proceed as follows (Blundell et al., 2015):

1. Draw $\epsilon \sim \mathcal{N}(0, I) = q(\epsilon)$ (used to denote the distribution of ϵ)

2. Let $w = \mu + log(1 + exp(\rho)) \odot \epsilon$
3. Let $\theta = [\mu, \rho]$
4. Loss function

$$\begin{aligned} f_x(w, \theta) &= \mathbb{E}_{q(\epsilon)}(logP(w, x) - logQ(w|\theta)) \\ &= logQ(w|\theta) - logP(x, w) \\ &= logQ(w|\theta) - logP(w)P(x|w) \end{aligned}$$

5. Compute the gradient w.r.t μ,

$$\nabla_\mu(x) = \frac{\partial f(w, \theta)}{\partial w} + \frac{\partial f(w, \theta)}{\partial \mu}$$

6. Compute the gradient w.r.t ρ,

$$\nabla_\rho(x) = \frac{\partial f(w, \theta)}{\partial w} \frac{\epsilon}{1 + exp(-\rho)} + \frac{\partial f(w, \theta)}{\partial \rho}$$

7. Update the variational parameters using the user-defined learning rate α:

$$\mu' = \mu - \alpha \nabla_\mu$$
$$\rho' = \rho - \alpha \nabla_\rho$$

The same kind of calculation and gradient updates can be applied to a mini-batch of training data (called S), and the gradient in this case would be the average of the gradients computed using the sample:

$$\hat{\nabla}_\mu(S) \approx \frac{1}{||S||} \sum_{x \in S} \nabla_\mu(x)$$
$$\hat{\nabla}_\rho(S) \approx \frac{1}{||S||} \sum_{x \in S} \nabla_\rho(x)$$

Importantly, this reparameterization gradient is unbiased (Miller et al., 2015), i.e. $\mathbb{E}(\hat{\nabla}_\theta) = \nabla_\theta ELBO(\theta)$, which lays the theoretical foundation for the use of stochastic gradient decent (using randomized mini-batches of training data) to perform gradient updates for BNN.

4.3 Posterior Inference and Prediction

With the above training process, we have approximated the actual posterior distribution on weights given the training data, $P(w|\mathcal{D})$, with the variational distribution, $Q(w|\theta^*)$. Now, we could proceed to perform posterior inference and predictions using the trained BNN. Depending on the loss function, we will use posterior mean of the weights for the BNN to minimize MSE, while we will use posterior median of the weights to minimize MAE. What is more, we could perform model ensembles by constructing different models with randomly drawn weights from the posterior distributions in a very computationally efficient manner, which could be used to access uncertainty is predictions and potentially improve model generalization to unseen data.

5 Experimental Results

To illustrate the power of BNN in practice, we will apply and evaluate the network in two different scenarios: a two-dimensional toy example and a classification problem using the Wine dataset. We will use a Python package called Edward (Tran et al., 2016)(Tran et al., 2017) to model distributions on the weights and biases of the network, perform the gradient updates using -ELBO as the loss function, and access model performance using unseen test data. We would also compare the performances between the standard neural networks and BNN for the latter experiment.

5.1 A Toy Experiment

For visualization purposes, we will first apply BNN to a two-dimensional toy dataset, in which the predictor, x, and the response variable, y, are generated as follows:

$$N = 20$$

$$x \sim Unif([-3, 3])$$

$$y = cos(x) + \mathcal{N}(0, 0.1 * abs(x))$$

Note that N refers to the sample size, and the addition of $\mathcal{N}(0, 0.1 * abs(x))$ is to add some heteroskedasticity to the data. We wish to access if BNN is capable of capturing such varying levels of uncertainty in the data.

For model architecture, we build a BNN model with one input layer, one hidden layer that contains two neurons, each has a weight and a bias term, followed directly by a non-linear transformation (aka activation function), and one output layer that has one neuron with a weight and a bias term. Given the curvature of the data points (which we can directly observe from the data), we decide to use the $tanh$ activation function.

Before training begins, we will first randomly initialize all the weights and biases as random draws from the standard normal distribution, then we place normal priors on each of the latent variables, using $softplus$ transformation for the standard deviation terms:

$$w_i \sim \mathcal{N}(\mu_i, log(1 + exp(\rho_i))\ I)$$

We first visualize the raw data and 10 predictions of all the x values within the shown x range, using random draws from the prior distributions of the weights and biases, shown in Figure 3:

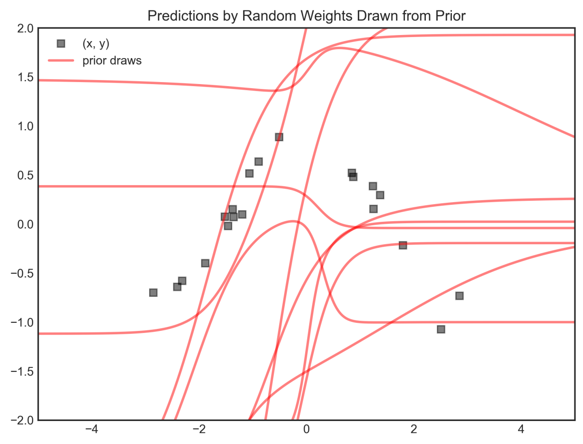

Figure 3: Raw Data and 10 Pred. using Random Draws from Prior Distribution

After that, we perform two independent runs of gradient descent using the ELBO approximation trick, for 50 and 300[2] iterations respectively. For both cases, we plot the predictions made by 10 random draws from the posterior distributions of the weights and biases (Figure 4 and 5), and the 95% credible (confidence?) intervals for predictions at all x values. This is obtained by taking the $[2.5\%, 97.5\%]$ interval for predictions at each x value for 1000 posterior draws. (Figure 6 and 7).

[2]The values of 50 and 300 are chosen based on trial and error: 50 gives poor but stabilized results with large margin of error, while 300 gives a pretty good fit to the data without being overconfident about the predictions.

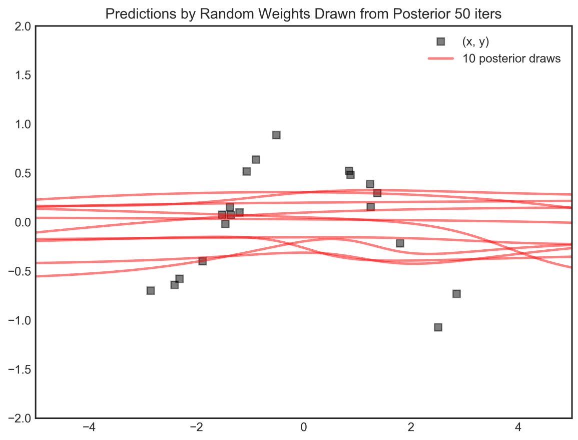

Figure 4: Posterior Predictions with 50 Iter.

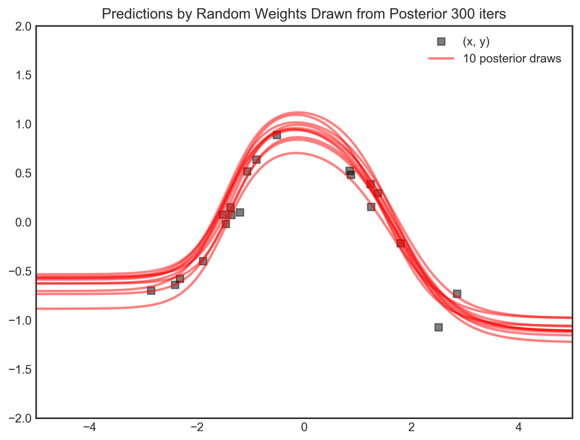

Figure 5: Posterior Predictions with 300 Iter.

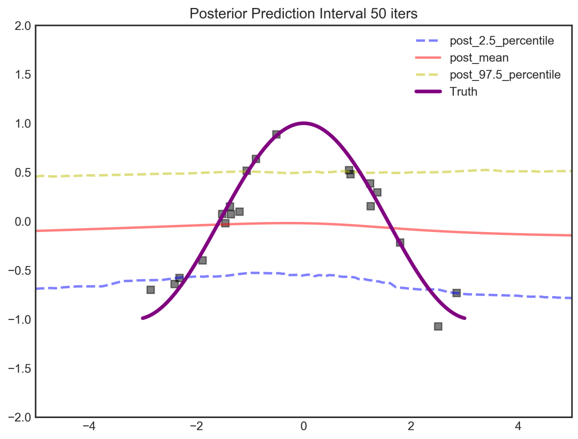

Figure 6: Posterior CI with 50 Iter.

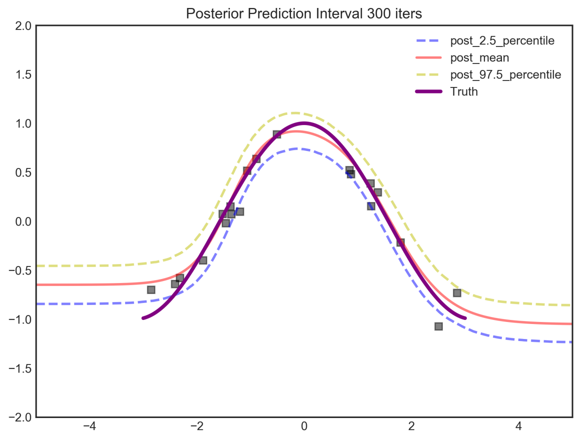

Figure 7: Posterior CI with 300 Iter.

We observe that, with 50 iterations, we are able to stabilize the weights and biases to capture the general shape and structure of the data, but the fit is not too good. However, with 300 iterations, we are able to capture the cosine relationship (the $Truth$ plotted in purple) between x and y quite well. More interestingly, if we observe the width of the credible interval at varying values of x, the CI is wider when there happens to be few data point (e.g. x close to 0), and as x moves away from 0 (the heteroskedasticity we introduced when creating y). Such finding suggests that BNN is able to capture the uncertainties in the predictor space where data is spare or where heteroskedasticity is present. Such ability to capture uncertainty is of great value to researchers and scientists, because it allows them to locate and understand areas where the model is uncertain about (as opposed to confidently give a prediction like the standard MLE neural networks, which may generalize poorly to unseen data), and where more data points could be collected to fill in the gaps that may improve model performance.

Also, to better illustrate the updates on distribution of weights during the training process of the BNN, we plot random draws from the distribution of the weight of the first neuron within the hidden layer, before training (prior), after 50 iterations (posterior 50) and after 300 iterations (posterior 300), shown in Figure 8:

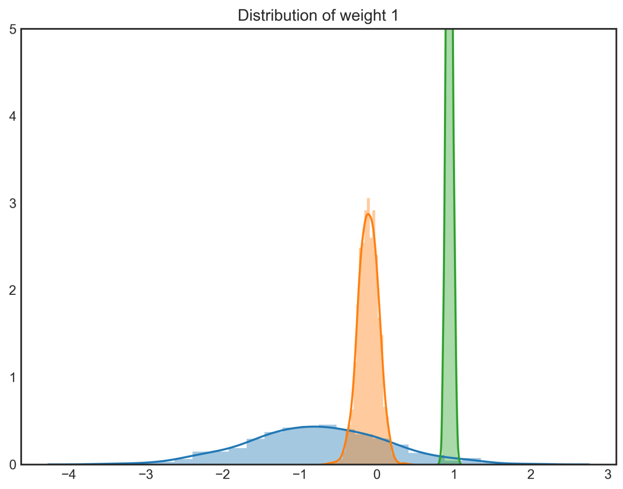

Figure 8: Prior and Posterior Distributions of Wight 1

Observe that, with 50 iterations, the weights moved away from -1, its initial value, and is centered at around 0. This corresponds to the fact that posterior predictions shown in Figure 4 after 50 iterations are almost flat. The variance of the distribution is also significantly smaller, which is reflected by the fact the posterior draws are generally in accordance with the shape of the data, as opposed to the quite random predictions in the prior draws. With 300 iterations, the distribution of weight 1 is closely centered at 1, with even lower variance, indicating a high confidence that the actual value of weight 1 should be 1 for the given the data and structure of the BNN. Even though the variance is very low, making the distribution on weight 1 almost a point mass, we have shown in Figure 7 that the model still preserves great ability in capturing and describing the uncertainty in its predictions. Thus, to some degree, we should treat the number of iterations as a hyper-parameter that controls the trade-off between model fit to the training data and degree of variability. Too few iterations will lead to poor fit of the data, while too many iterations,similar to its MLE counterpart, will make the model over-fit the training data and become over-confident in its predictions (which could lead to poor generalizability to new data).

5.2 Predicting Wine Types

To demonstrate the predictive power of BNN and to facilitate comparison of performances between BNN and the standard feed-forward neural networks, we apply the two frameworks to the Wine dataset[3] obtained from UCI Machine Learning Repository. Below, we will go through the data pre-processing, training process of both networks and compare their performance.

[3]Link: `https://archive.ics.uci.edu/ml/datasets/wine`

5.2.1 Data Pre-processing

For simplicity, we only include two types of wine from the given dataset, red and white, and frame the problem as a binary classification: classifying the type of wine, *type*, as either red or white. Our predictors include: 'fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality'. In Figure 9 below, we plot the correlation matrix of the variables within the dataset. We see that a lot of variables are highly correlated with the target variable, *type*. For a complete description of each of the variable, visit `https://archive.ics.uci.edu/ml/datasets/wine`.

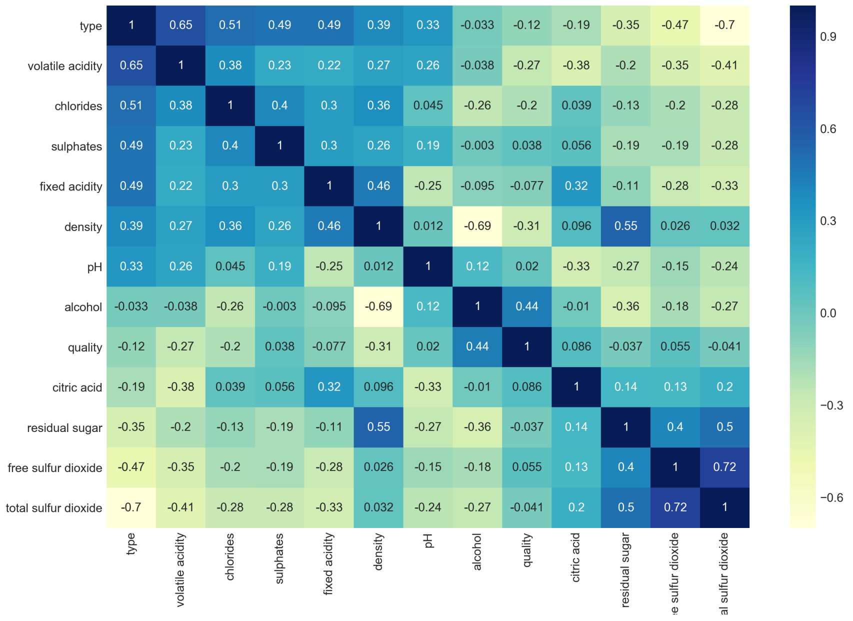

Figure 9: Correlation Matrix of the Variables within Wine Dataset

The entire dataset includes 6497 observations. Before training, we first divide the data into train and test sets, each with 3248 observations (the reason behind this 50-50 split with equal sizes is that the Edward package requires inputs to BNN to be of the same shape). Then, we standardize each of the predictors to have mean 0 and standard deviation 1. This is a common pre-processing step for training neural networks, because predictors with different scale and variance might be hard to work with. Furthermore, for BNN in particular, because the prior on *type* is a binary variable representing wine class, we have to place what is known as a *OneHotCategorical*[4] prior on it. So for BNN only, we have to apply one-hot-encoding to *type*, i.e. $0 \rightarrow [1, 0]$ and $1 \rightarrow [0, 1]$. With the standardized data, we are ready to train the two types of neural networks.

[4]The *OneHotCategorical* prior is commonly used as a prior on categorical variables. It could be initialized with multinomial probabilities or logits. For more info, visit: `http://edwardlib.org/api/ed/models/OneHotCategorical`

5.2.2 MLE Neural Networks

For the standard feed-forward MLE neural networks, we build a architecture with one input layer, three hidden layers using *ReLU* activation, each with 12, 8 and 8 hidden units (neurons) and one output layer with *sigmoid* transformation.

For training, we use the *EarlyStopping* monitor provided by the Keras package under Python so that the training process stops after the validation loss stops decreasing for more than two consecutive epochs. This kind of early stopping technique has been shown to prevent over-fitting (Blundell et al., 2015). We use the Adam optimizer to minimize the $binary_crossentropy$ loss and a learning rate of $5e-3$ to fit the model for 20 epochs. During the training process, we use 20% of the training data as validation set for the *EarlyStopping* monitor to use. The training process stopped after 12 epochs of training, with the lowest validation loss of 0.0256 achieved at epoch 10. The updates on loss values are reported below in Figure 10.

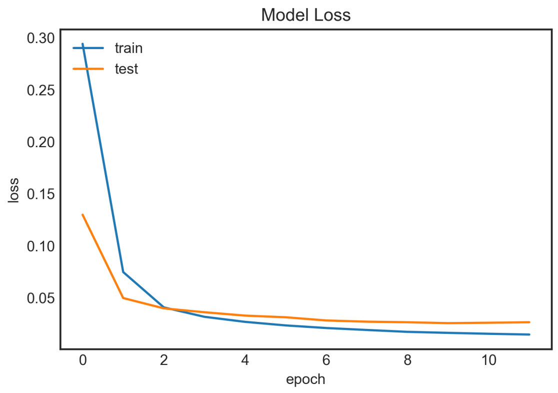

Figure 10: Loss Updates for Neural Networks

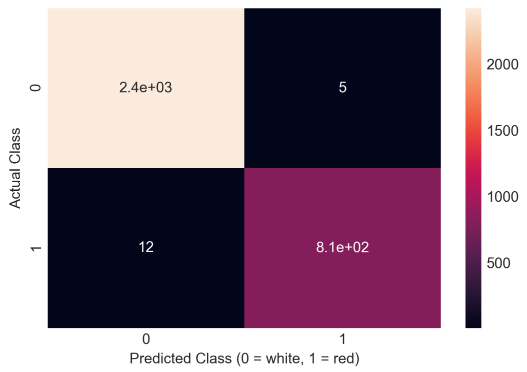

Figure 11: Confusion Matrix for Neural Networks

To access the model performance, we feed the features of the reserved test set to the trained neural networks to obtain the predictions. The confusion matrix is shown in Figure 11, and the overall test accuracy is 99.47%, or 3231 correct predictions out of 3248 observations.

5.2.3 Bayesian Neural Networks

Besides the standard neural networks, we also trained a Bayesian Neural Network model with the same architecture (1 input layer, 3 hidden layers with 12, 8 and 8 neurons using *ReLU* activation and 1 output layer using *sigmoid* activation). As in the toy example, the weights and biases of each neuron within the network is initialized as random draws from the standard normal distribution. Then, we place normal priors on top of each of the weights and biases, and a *OneHotCategorical* prior on the final neuron, i.e. the prediction of BNN. We run the ELBO approximation as optimization for 1000 iterations and the loss plateaued at around 700 iterations. We again plot the loss (-ELBO) against iterations, given in Figure 12 below:

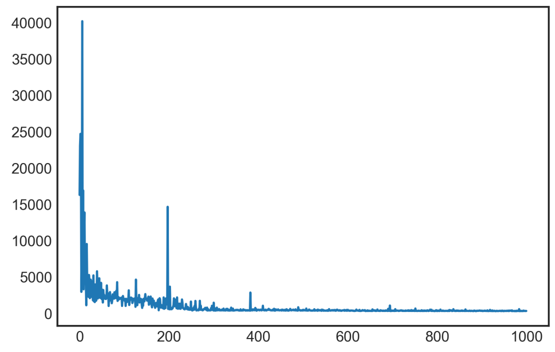

Figure 12: Loss Updates for BNN

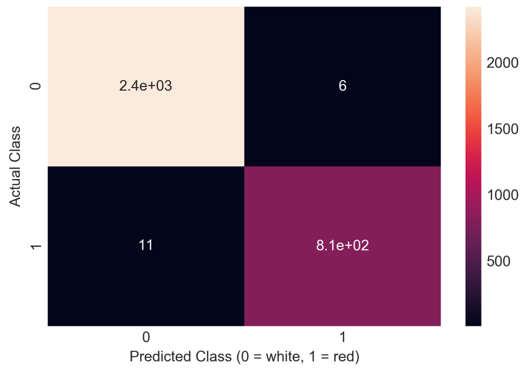

Figure 13: Confusion Matrix for BNN

After model training, we calculate the posterior mean (the posterior mean, median and mode are actually the same for a normal posterior) for the distributions of each of the weights and biases, and make predictions using these values. For the same set of test data, we actually obtained the same test accuracy of 99.47%, correctly classifying 3231 out of 3248 observations. We also report the confusion matrix of predictions from BNN above in Figure 13 (+1 for red and -1 for white).

5.3 Model Comparison

In the analysis above, we have applied the standard neural networks and BNN to the same classification problem. The two models happens to have obtained the same accuracy, though notice that the two models use completely different ways of training with different loss functions. Such comparable accuracy may not be reproducible on other predictive problems, but this serves as evidence that BNN could obtain similar levels of accuracy as that of the standard neural networks. Such phenomenon echos with the fact that performances or estimates made by Bayesian methods eventually converges to those of their Frequentist counterparts.

We could also imagine using the posterior distributions on weights and biases to obtain measure of uncertainties on the model predictions. But such calculation requires more careful discussion and design to be generally employed.

6 Conclusion and Further Discussion

In this project, we have recapped the realm of Bayesian methods and, in particular, introduced the model of Bayesian Neural Networks. We discussed in detail how BNN could be trained and then applied BNN to two different predictive tasks. In the first toy example, we demonstrate the power of BNN to capture uncertainties when data is sparse in the predictor space and when predictors suffer from heteroskedasticity; in the second analysis, we demonstrate the predictive power of BNN, which proved to be comparable with that of the standard neural networks.

For future explorations, we could try placing more complicated (mixed) prior on weights and biases of BNN, a practice that could lead to significant improvement in model performance (Blundell et al., 2015). We could also delve into the types of uncertainties that BNN is able to capture, and further discuss how these measure of uncertainties could used for practical purposes.

References

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *the 32nd International Conference on Machine Learning (ICML 2015)*. eprint arXiv:1505.05424.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

Miller, A. C., Foti, N. J., D'Amour, A., and Adams, R. P. (2015). Reducing reparameterization gradient variance. *Machine Learning (stat.ML); Learning (cs.LG)*.

Neal, R. M. (1995). *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, University of Toronto.

Tran, D., Hoffman, M. D., Saurous, R. A., Brevdo, E., Murphy, K., and Blei, D. M. (2017). Deep probabilistic programming. In *International Conference on Learning Representations*.

Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., and Blei, D. M. (2016). Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*.