Bayesian Neural Networks

John Hawkins [john.hawkins@sydney.edu.au]

June 18, 2018

1 Introduction

This document is a work in progress. Currently incomplete.

The goal of this document is to make explicit the derivation of the standard formulas and processes used in training a Bayesian Neural Network.

The Bayesian approach to machine learning involves a number of ideas and mathematical techniques that are not common knowledge among the broader machine learning community, nor to new students of the field. Ideally this document should help bridge that gap and make the entire process of developing a Bayesian model explicit and accessible to students and newcomers to the field.

There are a variety of approaches to designing and training Bayesian Neural Networks. For the sake of this document we have chosen the Bayesian Neural Network models developed by Chandra, Azizi and Cripps [1] as a reference and explain the derivation of all forumla in this particular paper.

2 Data

The data we are modelling is a single time series with discrete time steps for which we presume that each additional time step is a function of previous steps plus noise.

$$y_t = f(x_t) + \epsilon_t \tag{2.1}$$

Where x_t is a fixed window of previous time steps and $f(x_t)$ is an unknown non-linear function.

The prediction task, to which we will apply machine learning, is to approximate $f(x_t)$ based on a finite number of samples. This means simultaneously estimating the paramters θ of a function approximator and the properties of the noise ϵ_t. We generally assume that the noise is Gaussian and centred on zero, so we can define the distribution of ϵ_t as follows:

$$\epsilon_t \sim \mathcal{N}(0, \tau^2) \tag{2.2}$$

In other words each error term ϵ_t is drawn from a normal distribution with zero mean and τ^2 variance.

In order to use the complete time series data series data into the model we need to shape the data such that it can be used by a feed forward neural network. This involves determining a maximal time lag T, which in turn determines the embedding dimension D. In other words how many prior time steps are used as input into the model.

In the original paper this is demarcated $\mathbb{A}_{DT}$, i.e. the set of y_t values such that:

$$\mathbb{A}_{DT} = t; t > D, mod(t - (D+1), T) = 0 \tag{2.3}$$

3 Model

For this prediction task we will apply a Feed Forward Neural Network (FFNN) which consists of three layers of fully connected neurons. There are D input neurons (the symbol D capturing the embedding Dimension of the original time series data), there are H hidden neurons and 1 output neuron, indexed with symbol lower case o.

The hidden and output nodes of the network take as input a weighted sum of the activations of the network nodes that feed into them. This weighted sum is then processed by a non-linear activation function $g(x)$. There are a number of different choices for activation functions. a multiple options in the literature, but in this case the sigmoid function shown below:

$$g(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

So the output of any individual node N in the network can be expressed as the non-linear function applied to the sum of its weighted inputs I, as follows:

$$N(I) = g(\delta_n + \sum_{i=1}^{I} w_i I_i) \tag{3.2}$$

The output of the entire neural network can be expressed as a composition of each of the individual node activations leading up to the final output node, as follows:

$$f(x_t) = g(\delta_o + \sum_{h=1}^{H} v_h g(\delta_h + \sum_{d=1}^{D} w_{dh} y_{t-d})) \tag{3.3}$$

Where δ_o is the bias of the output neuron, and δ_h demarcates the biases of each of the H hidden nodes. The values y_{t-d} demarcate each of the d inputs (or features) used at time step t to predict y_t. In this model these features are time lagged copies of previous outputs, i.e. this is an autoregressive model. The weights of the network are shown as w and v, input to hidden and hidden to output respectively.

There is a convention to encapsulate the entire set of parameters that need to be estimated in a single term θ, defined as:

$$\theta = (W, V, \delta, \tau^2) \tag{3.4}$$

So we can alternatively express the output of the neural network as $E(y_t|x_t, \theta)$. In other words it is the expected value of y_t given the input features x_t at time t and the network parameters θ. To train the model these paramters need to be estimated based on a training data set.

4 Bayesian Primer

The goal of Bayesian machine learning is to estimate the posterior distribution of the model parameters θ given the observed data $\mathbb{D}$. Unlike most other machine learning approaches we will not a point estimate of the model parameters, but a distribution. This distribution allows to make point estimates if necessary, or to calculate confidence bands directly from the model.

The standard Bayesian notation for what we are doing is written:

$$p(\theta|\mathbb{D}) = \frac{p(\mathbb{D}|\theta)p(\theta)}{p(\mathbb{D})} \tag{4.1}$$

The term on the left $p(\theta|\mathbb{D})$ is called the ***Posterior***. This is the conditional distribution over the model parameters given the observed data. On the right hand side there are three terms: $p(\mathbb{D}|\theta)$ is the conditional probability of the data given a set of model parameters, this usually called the ***Likelihood*** function because the data is typically fixed and we investigate how its value changes as a function of the model paramters. The term $p(\theta)$ is the marginal probability of a given set of model paramters, this is typically called the ***Prior***: as it captures our a priori expecations about what the distribution over model parameters would look like. The final term in the denominator $p(\mathbb{D})$ is the marginal probability of the data, also known as the ***Evidence***.

Unfortunately, in most practical applications it is computationally expensive (if not thoroughly intractable), to directly calculate the posterior distribution. This is usually because calculating the denominator means marginalising over all possible models. In order to make Bayesian machine learning practical we need to estimate this posterior distribution over the model parameters.

How do we do this?

Techniques for estimating the posterior are an ongoing area of research, however they all generally involve some kind of sampling strategy to create an empirical estimate of the distribution. In this worked example we are going to take samples from the model parameters space. We want to design this sampling process such that those samples reflect the true posterior distribution. What this means is that we need to have some criteria for accepting or rejecting samples that will lead to a final set of accepted samples that reflects the true posterior distribution.

To perform this trick of making and selecting samples of the distribution we will use a Markov Chain Monte Carlo procedure with Metroplis Hastings step for accepting or rejecting the samples. The individual samples that will be put forward for acceptance or rejection are called ***poposals***.

5 MCMC: Markov Chain Monte Carlo

Markov Chain Monte Carlo processes are in essence a method for taking a series of samples, in which each subsequent sample is dependent on the preceding sample (the Markov Chain). The name Monte Carlo refers to group of numerical sampling methods for estimating unknown parameters. The classic example is to estimate the value of pi by dropping pins onto drawing of a circle inside a square. By counting the

The markov chain component means that each sample is dependent on the values of the last accepeted sample (proposal). In other words the sampling process has a state (the current model parameters), and method for generating new samples from that state (the markov chain). Finally we need a method for choosing whether or not to accept the proposed new model parameters that will garantee that the accepted set of samples reflects the true posterior (Metropolis Hastings).

In order to ensure that the result of the sampling process results in a set of parameters that reflect the true posterior of the parameters we need be careful about how the proposals are generated and then accepted. The proposals are generated by making small changes to current set of parameters.

When we then accept a proposal we do so with a probability that should reflect the relative proportion of the current parameters and the proposed parameters in the posterior distribution. The central trick of the Metropolis Hastings step is that to calculate this acceptance probability we only need a function $F(\theta)$ that is proportional to the true posterior probability, because it is the ratio of the respective posterior probabilities that matters when calculating the acceptance probability. Given this function $F(\theta)$ we calculate the acceptance probability as follows:

$$p(A) = min\big(1, \frac{F(\theta_p)}{F(\theta_c)}\big) \tag{5.1}$$

In other words we always accept proposals which are more probable given the data, and those that are less probable we accept with a probability proportional to their relative probability. For example if the current parameter set has a posterior probability of 0.8 and the proposed has probability 0.6, then the probability of acceptance is 0.6/0.8 or 0.75.

However, as noted before we do not have a function for directly calculating the true posterior probability (that is our goal). To use the Metropolis Hastings step we need find a function $F(\theta)$ that is proportional to the posterior probability.

How do we do this?

If you look at the formula for the posterior 4.1 you will note that that denominator $p(\mathbb{D})$ will be a constant for all candidate values of θ. This means we can ignore it and work with the numerator with certainty that it will be proportional to the posterior probability. To further simplify the process we will take the log of this function (so that products and quotients become the more computationally tractable sums and differences).

In other words our function $F(\theta)$ we will use in our Metropolis Hastings step is defined as:

$$F(\theta) = log(p(\mathbb{D}|\theta)p(\theta)) \tag{5.2}$$

In other words we need only calculate log likelihood and the log of the prior in order to evaluate each candidate value of θ.

In many instances the function above will be sufficient for estimating the posterior with an MCMC process. However, it does assume that the process for making proposals is symmetric. In other words the probability that θ_c is generated from θ_p has to be equal to the probability that θ_p is generated from θ_c. Note this is not the acceptance probabilities, which are clearly not symmetric. It is a condition on the process that precedes the calculation of the acceptance probability. In other words the Markov Chain procedure that generates each new candidate parameter set from the current parameter set: this needs to be symmetric.

Not all processes for exploring a paramter space are garanteed to have this property of symmetry. For example in the Langevin Bayesian Neural Network paper the gradient of the loss function was combined with a random walk. In these instances the Metropolis Hastings step requires an additional term that corrects for the fact that some parameter settings are less likely to be sampled than others. In this case the acceptance probability is calculated as

$$p(A) = min\Big(1, \frac{F(\theta_p)}{F(\theta_c)}\frac{p(\theta_c|\theta_p)}{p(\theta_p|\theta_c)}\Big) \tag{5.3}$$

Looking at the new term at the end of this new defintion of $p(A)$ we see that it a correction term that accounts for the fact that the probability of observing θ_c given θ_p is not equal to the probability of observing θ_p given θ_c. If these conditional probabilities are in fact equal, then this entire term become equal to one and we return to our original defintion of $p(A)$.

6 Applying MCMC

The MCMC algorithm will iteratively sample the model paramaters θ and determine whether to keep each sample. Accepted samples will be added to a vector E which contains an empirical estimate of the posterior distribution of parameters given the data. The MCMC algorithm is captured by the following procedure:

Algorithm 1: Overview of the Markov Chain Monte Carlo Algorithm

```
Initialize $\theta_c$ by drawing from the prior distributions;
Initialise $E$ to an empty vector;
for $s$ in 1 to $S$ do
    Generate a new proposal $\theta_p$ from the current parameters $\theta_c$;
    Calculate the log likelihood $log(p(Y|\theta_p))$ and the log prior $log(p(\theta_p))$;
    Calculate the acceptance probability $p(A)$;
    Draw a value $d$ from Uniform (0,1) distribution;
    if $d <= p(A)$ then
        Add $\theta_p$ values to $E$;
        Update $\theta_c$ to $\theta_p$;
    else
        Add $\theta_c$ to $E$;
    end
end
E now holds an empirical joint distribution of $\theta$ conditioned on the data;
```

7 Prior Distributions

To fit model parameters in a Bayesian fashion we need to define prior distributions over all of the model parameters. These distributions will be used in every step of the MCMC process to produce the log of the prior probability of a given set of parameters, which is then used to calculate the acceptance probability.

$$\begin{aligned}
\delta_o &\sim \mathcal{N}(0, \sigma^2) \\
v_h &\sim \mathcal{N}(0, \sigma^2) \text{ for h } = 1, ..., H \\
\delta_h &\sim \mathcal{N}(0, \sigma^2) \text{ for h } = 1, ..., H \\
w_{dh} &\sim \mathcal{N}(0, \sigma^2) \text{ for h } = 1, ..., H \text{ and d } = 1, ..., D \\
\tau^2 &\sim \mathcal{IG}(\nu_1, \nu_2)
\end{aligned}$$

8 Log Likelihood Function

The log likelihood function $log(p(\mathbb{D}|\theta))$ is the first part we need to calculate for our function $F(x)$.

To calculate the probability of the data given a set of model parameters we will assume that our model is capable of capturing the true underlying data generating process (shown in **??**). So any error in the model predictions should be captured by the error term in the data generating process (this is the only stochastic element). The error was assumed to be a Gaussian dstribution with zero mean and τ^2 variance. The density function for this is as follows:

$$\epsilon(x) = \frac{1}{\tau\sqrt{2\pi}} e^{\frac{-x^2}{2\tau^2}} \tag{8.1}$$

The error of the model at any timestep t can be found by taking the difference between the actual value y_t and the value given by the model. This is denoted:

$$Err(t) = y_t - E(y_t|x_t, \theta) \tag{8.2}$$

The probability of the data given the model parameters is calculated by taking a product over the error distribution for the observed errors on each of the $n = |\mathbb{A}_{DT}|$ data points, as shown below.

$$p(\mathbb{D}|\theta) =_{t=1}^{n} \frac{1}{\tau\sqrt{2\pi}} e^{\frac{-Err(t)^2}{2\tau^2}} \tag{8.3}$$

When we take a log of this function the product can be converted to a sum of logs, as follows:

$$log(p(\mathbb{D}|\theta)) = \sum_{t=1}^{n} log(\frac{1}{\tau\sqrt{2\pi}} e^{\frac{-Err(t)^2}{2\tau^2}}) \tag{8.4}$$

The product inside the internal logarithm can also be converted in a sum of logs, and separated, as follows:

$$log(p(\mathbb{D}|\theta)) = \sum_{t=1}^{n} log(\frac{1}{\tau\sqrt{2\pi}}) + \sum_{t=1}^{n} log(e^{\frac{-Err(t)^2}{2\tau^2}}) \tag{8.5}$$

This can be further simplified by converted the fraction in the first sum into a difference of logs, in which the first term, $log(1)$ is simply zero, which the log and exponentiated term in the second sum cancel. Leaving:

$$log(p(\mathbb{D}|\theta)) = \sum_{t=1}^{n} -log(\tau\sqrt{2\pi}) + \sum_{t=1}^{n} \frac{-Err(t)^2}{2\tau^2} \tag{8.6}$$

In the first sum there is no dependence on the content of the D data points, so this sum becomes simply n multiplied by the content of the sum. In the second sum we can pull the term $\frac{1}{2\tau^2}$ out because it is a constant, leaving:

$$log(p(\mathbb{D}|\theta)) = -nlog(\tau\sqrt{2\pi}) + \frac{1}{2\tau^2}\sum_{t=1}^{n} -Err(t)^2 \tag{8.7}$$

If we substitute in the defintion of the error function and move the negative sign out the front of the final sun we get the following defintion for the log likelihood:

$$log(p(\mathbb{D}|\theta)) = -nlog(\tau\sqrt{2\pi}) - \frac{1}{2\tau^2}\sum_{t=1}^{n}(y_t - E(y_t|x_t,\theta))^2 \tag{8.8}$$

The first term can be modified by taking the square root out of the entire term that is being given to logarithm function:

$$log(p(\mathbb{D}|\theta)) = -nlog((\tau^2 2\pi)^{frac12}) - \frac{1}{2\tau^2}\sum_{t=1}^{n}(y_t - E(y_t|x_t,\theta))^2 \tag{8.9}$$

The nature of the logrithm allows us to then extract that power and put it as a coefficient at the front:

$$log(p(\mathbb{D}|\theta)) = -\frac{n}{2}log(\tau^2 2\pi) - \frac{1}{2\tau^2}\sum_{t=1}^{n}(y_t - E(y_t|x_t,\theta))^2 \tag{8.10}$$

We can further exploit the nature of the logarithm and split the first term into 2 parts:

$$log(p(\mathbb{D}|\theta)) = -\frac{n}{2}log(\tau^2) + \frac{n}{2}log(2\pi) - \frac{1}{2\tau^2}\sum_{t=1}^{n}(y_t - E(y_t|x_t,\theta))^2 \tag{8.11}$$

The second of the new terms is a constant (not dependent on any of the parameters). So for the sake of simplicity we can elimiate it (we only need a function that is proportional to the true probability distribution). Leaving us with the final form of the log likelihood we will use in the MCMC process:

$$log(p(\mathbb{D}|\theta)) = -\frac{n}{2}log(\tau^2) - \frac{1}{2\tau^2}\sum_{t=1}^{n}(y_t - E(y_t|x_t,\theta))^2 \tag{8.12}$$

9 Log Prior Function

The prior probability of a given set of parameters θ is denoted:

$$p(\theta) = p(\tilde{w}, v, \delta, \tau^2) \tag{9.1}$$

Note that δ contains the biases of both hidden and output nodes, and w is a matrix of weights, while v is a vector containing hidden to output weights.

There are a number of additional parameters used to determine the distributions of the model parameters. So if we add these, to make the entire prior explicit, then the full joint probability is:

$$p(\theta) = p(\tilde{w}, v, \delta, \tau^2, \sigma^2, \nu_1, \nu_2) \tag{9.2}$$

Which we can reshape as follows:

$$p(\theta) = p(\tilde{w}, v, \delta, \tau^2 | \sigma^2, \nu_1, \nu_2) p(\sigma^2, \nu_1, \nu_2) \tag{9.3}$$

Now, because we assume that the distributions of all model parameters are independent we can change this again to:

$$p(\theta) = p(\tilde{w}|\sigma^2, \nu_1, \nu_2) p(v|\sigma^2, \nu_1, \nu_2) p(\delta|\sigma^2, \nu_1, \nu_2) p(\tau^2|\sigma^2, \nu_1, \nu_2) p(\sigma^2, \nu_1, \nu_2) \tag{9.4}$$

If we then remove the parameters that are irrelevant in the conditioning of the probabilities we get:

$$p(\theta) = p(\tilde{w}|\sigma^2) p(v|\sigma^2) p(\delta|\sigma^2) p(\tau^2|\nu_1, \nu_2) p(\sigma^2, \nu_1, \nu_2) \tag{9.5}$$

In other words, the distributions of all model parameters depend on a single hyperparameter σ^2 and the noise depends on the two parameters for the Inverse Gamma distribution. The probability of the hyper-parameters is equal to one becuase they have been set, rather than drawn from a distribution.

$$p(\theta) = p(\tilde{w}|\sigma^2) p(v|\sigma^2) p(\delta|\sigma^2) p(\tau^2|\nu_1, \nu_2) \tag{9.6}$$

References

[1] Rohitash Chandra, Lamiae Azizi, Sally Cripps (2017) *Bayesian neural learning via Langevin dynamics for chaotic time series prediction*,