Basics of Bayesian Neural Networks

Nicholas Orriols

Motivation

Suppose we have a (training) data set *D* which contains a series of inputs ***x*** (collectively referred to as D_x) and their corresponding response vectors ***y*** (collectively referred to as D_y), and we want to fit a feedforward artificial neural net (ANN) with *n* layers and activation σ. The output of the i^{th} layer is represented by

$$F_i(\boldsymbol{x}) = \sigma(W_i F_{i-1}(\boldsymbol{x}) + b_i) \tag{1}$$

for *i = 1, 2…n*. When fitting this model to *D*, we find the weight matrices and bias vectors (collectively, the parameterization θ) which minimizes the average of some loss function *L* over all predictions. That is, if ***F(x)*** is our net's output for input vector ***x*,** then we minimize *J*

$$J(\theta, \mathrm{D}) = \frac{1}{|D|}\sum_{i=1}^{|D|} L(F(x_i), y_i) \tag{2}$$

Notice that ANNs, once parameterized, provide a single estimate for the response of a given input. Oftentimes, one is interested in the potential distribution of predictions. Additionally, ANNs can overconfidently generalize to out-of-sample data points when making predictions, which can lead to poor performance, especially on out-of-sample points [1]. Lastly, ANNs provide only a single estimate (effectively maximum likelihood) for the parameterization. As a result, we have no idea as to the spread/distribution of possible parameterizations for *D*, or the probability of obtaining a given parameterization.

In the interest of risk mitigation, the distributions of parameters and the confidence in predictions for a given parameterization are desired. Stochastic neural nets assume the weights or activations of an ANN are stochastic and simulate many models under the assumed distribution(s). Bayesian Neural Nets (BNNs) are stochastic neural nets trained using Bayesian inference.

To begin estimating the variance associated with this model, we assume that θ is a random variable following some chosen prior distribution $p(\theta)$. We also choose a prior confidence in the model's predictions represented by $p(\boldsymbol{y}|\boldsymbol{x}, \theta)$. Making the common assumption that the inputs (and thus the predicted outputs for a given parameterization) are independent, our chosen prior confidence implies

$$\prod_{\mathrm{i}=1}^{\mathrm{N}} p(\mathbf{y_i}|\mathbf{x_i}, \theta) = \mathrm{p}\left(\mathrm{D_y}\middle|\mathrm{D_x}, \theta\right) \tag{3}$$

We can think of $p(\theta|D)$ as the distribution of parameterizations based on the dataset *D*. Using our choice of prior distribution for θ, by Bayes formula we have

$$\mathrm{p}(\theta|\mathrm{D}) = \frac{p(\mathrm{D}|\theta)p(\theta)}{p(\mathrm{D})} = \frac{p(\mathrm{D}|\theta)p(\theta)}{\int_\Theta p(\mathrm{D}|\theta_t)p(\theta_t)d\theta_t} = \frac{p\left(\mathrm{D_y}\middle|\mathrm{D_x}, \theta\right)p(\theta)}{\int_\Theta p\left(\mathrm{D_y}\middle|\mathrm{D_x}, \theta_t\right)p(\theta_t)d\theta_t} \tag{4}$$

where Θ represents the domain of possible parameterizations. This quantity represents the epistemic uncertainty in our model: generally, epistemic uncertainty is regarded as lack of knowledge about things which can be known (knowable unknowns) such as ideal model parameters. Large values for $p(\theta|D)$

indicate strong confidence in obtaining parameterization θ from dataset *D*, and thus a low amount of epistemic uncertainty. Epistemic uncertainty can be reduced with more data.

Conversely, $p(\boldsymbol{y}|\boldsymbol{x},\theta)$ represents aleatoric uncertainty for outcomes: aleatoric uncertainty is associated with the noisy relationship between inputs and their known response values. Low aleatoric uncertainty is represented by large values of $p(\boldsymbol{y}|\boldsymbol{x},\theta)$, which indicates strong confidence in obtaining prediction *y* from input *x* with parameterization θ.

BNNs' ability to discern between epistemic and aleatoric uncertainty and quantify both is one of their most practical benefits. ANNs can be overconfident when making predictions for out-of-sample points. Conversely, out-of-sample points always present with high epistemic uncertainty: consequently, BNNs will not be over-confident when it makes predictions for out-of-sample points. This advantage of BNNs is invaluable for applications where incorrect predictions must be avoided.

Furthermore, since we have assumed a distribution of parameterizations, we can sample *N* independent parameterizations $\theta_1, \theta_2, \ldots \theta_N$. Assuming the response vectors $D_{\boldsymbol{y}}$ are continuous, and letting $F_\theta(\boldsymbol{x})$ be the output of a neural net with parameterization θ for input $\boldsymbol{x}$, we can average the models for an ensemble learning estimate for the response of $\boldsymbol{x}$

$$\hat{\boldsymbol{y}} = \frac{1}{N}\sum_{i=1}^{N} F_{\theta_i}(\boldsymbol{x}) \tag{5}$$

Note here that $\hat{\boldsymbol{y}}$ is the sample mean for the predictions for our set of *N* models for some input $\boldsymbol{x}$. Although there is no theoretical argument as to why, it has been shown BNNs' estimates can outperform ANNs' estimates [1]. Using the above equation, we can calculate the sample covariance matrix of predictions for input $\boldsymbol{x}$ as

$$\hat{\Sigma}_{\boldsymbol{y}|\boldsymbol{x},D} = \frac{1}{N-1}\sum_{i=1}^{N}\left(F_{\theta_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right)\left(F_{\theta_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right)^T \tag{6}$$

If our predictions are single-entry vectors (i.e., scalars), then this formula reduces to the commonly known sample variance formula, and we obtain a scalar measure of variance. This matrix represents the uncertainty in our predictions for a given $\boldsymbol{x}$ and will be used to evaluate the model's performance with calibration curves.

To exactly quantify the model's uncertainty in prediction for any $\boldsymbol{x}$, we can use the posterior to write

$$p(\boldsymbol{y}|\boldsymbol{x},D) = \int_\Theta p(\boldsymbol{y}|\boldsymbol{x},\theta_t)p(\theta_t|D)\,d\theta_t \tag{7}$$

This is our confidence in response $\boldsymbol{y}$ for input $\boldsymbol{x}$. This can be computed directly [3], but it is more practical to sample it indirectly by sampling around (that is, with noise) $F_\theta(x)$, where θ is sampled from a uniform distribution or a variational distribution which approximates $p(\theta)$ [1].

Building Bayesian Neural Networks

Functional Model

To build a BNN, we first need to specify a functional model, i.e., the numbers of neurons in each layer, the number of layers, etc. Since we do not make any assumptions of the functional model, virtually any functional model can be used for Bayesian deep learning. In practice, it is a good idea to choose a functional model based on the dataset, and then use a Bayesian deep learning to estimate the variance of that model's parameterizations and predictions.

Stochastic Model

In Bayesian deep learning, one can assume the parameterization (biases, weights) of the net or the activation functions of each layer to be stochastic. Throughout this paper, we consider only Bayesian neural nets with stochastic weights. A common but arbitrary choice for the stochastic model is

$$\theta \sim \mathrm{p}(\theta) = \mathcal{N}(\mu, \Sigma) \tag{8}$$

$$y \sim \mathrm{p}(\boldsymbol{y}|\boldsymbol{x}, \theta) = \mathcal{N}(F_\theta(\boldsymbol{x}), \Sigma) \tag{9}$$

[1]. Here we are assuming the vectors of weights and biases for the functional model are normally distributed with mean μ and variance Σ. We are also assuming the response for a given input x is normally distributed with variance Σ about the prediction for input $\boldsymbol{x}$ by the functional model with parameterization θ. Choices for the distribution parameters μ, Σ are discussed shortly.

Note that the second assumption quantifies the aleatoric uncertainty in the model. Also, using both assumptions and equation (4), we can calculate the posterior up to a scaling constant. As we shall soon see, we can sample the posterior without knowing the evidence term (the denominator), so long as we know the numerator of equation (4).

Priors

The default and most common choice of parameters for $p(\theta)$ are $\mu = \vec{0}, \Sigma = \sigma I$ [1]. Although there is no theoretical justification for this choice over other potential priors, it is preferred because of some properties which are discussed below.

Despite its benefits, oftentimes the normal prior is not the ideal choice. Ultimately, choice of prior depends on the data and the way the BNN behaves, so the decision is made on a case-by-case basis, often by testing various alternatives [2].

Regularization

It can be shown that the above normal prior is equivalent to ℓ_2 regularization, which is similar to penalized regression since it biases the weight matrices (instead of regression coefficients) toward zero matrices. In this case, regress on θ over *n* layers to minimize

$$K(\theta) = J(\theta) + \frac{\lambda}{2|D|}\sum_{i=1}^{n}||W_i||_F^2 \quad (10)$$

λ is a regularization parameter and can be chosen with cross-validation; alternatively, each layer can have its own regularization parameter, but this often incurs a large increase in computation time. Regardless of the number of regularization parameters, a bias-variance tradeoff is presented: setting λ too large could make too many neurons have weights of 0, potentially biasing the BNN, while setting λ too small could lead to an overparameterized network with large variance.

The normal prior is also preferred because it reduces the probability of encountering scaling symmetry. Scaling symmetry is when two adjacent layers have weight matrices which are scalar multiples of each other; this can arise when using certain activations such as RELU. This is a type of statistical unidentifiability, that is, a state where inference can lead to different model parameterizations. For BNNs, unidentifiability means that the posterior is not easily approximated. The normal prior described reduces the scaling symmetry problem via regularization [1].

Bayesian Inference Algorithms

With BNNs, one of our main objectives is to find the posterior $p(\theta|D)$. However, the evidence term (denominator in equation (4)) is almost always too complex to justify computing it. Furthermore, even if we have the evidence term, sampling the posterior directly is oftentimes too difficult because the sampling space is very large, and we must apply a non-trivial transformation to uniform samples [1]. Consequently, in practice the posterior is approximated, most often by Markov Chain Monte Carlo or variation inference methods. The following is a brief summary of these two inference methods.

Markov Chain Monte Carlo Methods

In general, Markov Chain Monte Carlo (MCMC) methods construct sequences of samples S_{i+1} which depend on the previous sample S_i and eventually have a desired distribution. More technically, MCMC methods construct a Markov Chain of autocorrelated samples with a desired equilibrium distribution.

Since we do not know the distribution of the posterior, we consider the Metropolis-Hastings class of MCMC algorithms. These algorithms only require a distribution which is proportional to the desired distribution: since we can calculate the numerator in equation (4), these algorithms circumvent needing the evidence term.

Briefly, the algorithm samples candidate points from a proposal distribution "around" the previous point, and accepts the candidate point if it is more likely according to the objective distribution (in this case, the posterior); if the candidate point is less likely, it is accepted with a certain probability *p*. The algorithm is summarized in Figure 1. In this case, *f* would be the numerator in equation (4).

Algorithm 1 Metropolis-Hasting

Draw $\boldsymbol{x}_0 \sim Initial$
while $n = 0$ **to** N **do**
  Draw $\boldsymbol{x}' \sim Q(\boldsymbol{x}|\boldsymbol{x}_n)$
  $p = min\left(1, \frac{Q(\boldsymbol{x}'|\boldsymbol{x}_n)}{Q(\boldsymbol{x}_n|\boldsymbol{x}')}\frac{f(\boldsymbol{x}')}{f(\boldsymbol{x}_n)}\right)$
  Draw $k \sim Bernoulli(p)$
  if k **then**
    $\boldsymbol{x}_{n+1} = \boldsymbol{x}'$
    $n = n + 1$
  end if
end while

Figure 1 [1]

Note the proposal distribution *Q* (potentially) changes with each step. *Q* is ideally chosen to be centered around the previous point, so normal and uniform distributions centered about the previous point are common choices.

An important consideration is the variance/spread of the proposal distribution. If the variance is too small, the samples may be more autocorrelated than desired, which can lead to only drawing samples/candidate points from a small, non-representative area of the posterior. If the variance is too large, many candidate points will be rejected which increases computation time and can also affect the quality of samples.

Unfortunately, there is no general procedure to optimize the variance of the proposal distribution. However, there is a method to optimally choose *p* which utilizes Hamiltonian mechanics [1]. Additionally, there are several implementation details for MCMC we have not mentioned, such as burn time and caching to consider, but the overall MCMC procedure is the same.

Variational Inference

In variational inference for BNNs, we approximate the posterior $p(\theta|D)$ by a variational distribution $q_\phi(\theta)$. This distribution is specified/parameterized by ϕ which is chosen such that $q_\phi(\theta)$ is "as close" to $p(\theta|D)$ as possible. The most convenient measure of closeness is *Kullback-Leibler divergence* also known as *KL divergence* [1].

Roughly, if *Q* and *R* are two continuous probability distributions, the KL divergence of *R* from *Q* measures the information lost when using *R* to approximate *Q*. In our case, the KL divergence is given by

$$\mathrm{D}_{KL}(\mathrm{q}_\phi||P) = \int_\Theta q_\phi(\theta_t)\, log\left(\frac{q_\phi(\theta_t)}{P(\theta_t|D)}\right) d\theta_t \tag{11}$$

The above use of P instead of p is to emphasize that $P(\theta|D)$ gives the conditional probability of the data; properties of probability will be used in derivation of equation (12). Although the posterior (which we are trying to estimate) is in the integrand, we can derive the evidence lower bound, also known as the ELBO, as

$$\mathrm{ELBO} = \int_\Theta q_\phi(\theta_t)\, log\left(\frac{P(\theta_t, D)}{q_\phi(\theta_t)}\right) d\theta_t = log\big(P(D)\big) - D_{KL}\big(q_\phi||P\big) \tag{12}$$

A derivation of this formula is given at the end of this paper. Since $log\big(P(D)\big)$ is a constant given any choice of prior, minimizing the KL divergence is equivalent to maximizing the ELBO, so we do not need to compute the posterior to choose ϕ.

Given the variational distribution $q_\phi(\theta)$, many methods of classical machine learning can be used to maximize the ELBO. Most popular is stochastic variational inference [1].

Optimizations to Bayesian Learning

MCMC methods and variational inference are the foundation for most Bayesian deep learning methods. However, they must be modified in order to perform well for large functional models. These various improved methods are very interesting but covering them could be another paper on its own. However, we presented MCMC and variational inference because they are used in the more efficient algorithms for Bayesian deep learning.

Evaluating Bayesian Neural Networks

Whereas point-estimate neural net's output a point-estimate prediction for a given input, BNNs output a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x}, D)$ for a given input. However, we can extract an estimate for the response associated with a given input from this distribution: the most common estimate would be the expected value of the distribution. After computing the estimates from the BNN for a testing dataset, there are many metrics we can use to evaluate our model's performance (e.g., MSE, cross-entropy) by comparing our model's estimates versus the known response values for our test inputs.

It is also important to determine if the BNN is overconfident or underconfident in its predictions, which amounts to examining the behavior of the output conditional distribution. This is most readily accomplished with calibration curves.

Calibration Curves

Generally, a calibration curve is a function $p^*: [0,1] \rightarrow [0,1]$ where p^* is the observed frequency of observations expressed as a function of the probability $\hat{p}$ predicted by the BNN. If p^* is consistently greater than $\hat{p}$, then our model is underconfident; if p^* is consistently less than $\hat{p}$, then our model is overconfident. Accordingly, we desire $\hat{p} \cong p^*$, which would indicate our model is neither overconfident nor underconfident.

Since we have assumed continuous response vectors, our model outputs a (continuous) distribution of possible outputs. If we have a test set *T* of randomly selected inputs $T_{\boldsymbol{x}}$, we can assume that our model's predictions for $T_{\boldsymbol{x}}$ are independent. Letting $\widehat{\boldsymbol{y}}_{\boldsymbol{i}}$ be i[th] model output, we can then assume

$$(\widehat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{i}})^T \Sigma_{\boldsymbol{y}_{\boldsymbol{i}}}^{-1} (\widehat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{i}}) \sim \chi^2_{Dim(\boldsymbol{y}_{\boldsymbol{i}})} \tag{13}$$

This is the i[th] variance-normalized square error (VNSE) for model. Since we can assume its distribution, we can calculate our model's i[th] predicted probability as the probability of obtaining a VNSE (for the i[th] prediction) less than or equal to the observed VNSE. To estimate the covariance matrix, we use the formula in equation (6). Formally, the i[th] predicted probability is

$$\hat{p}_i = X^2_{Dim(\boldsymbol{y})} \left((E_{p(\boldsymbol{y}|\boldsymbol{x}_i, D)}[\boldsymbol{y}] - \boldsymbol{y}_i)^{\intercal} \Sigma^{-1}_{\boldsymbol{y}|\boldsymbol{x}_i, D} (E_{p(\boldsymbol{y}|\boldsymbol{x}_i, D)}[\boldsymbol{y}] - \boldsymbol{y}_i) \right) \tag{14}$$

where $X^2_{Dim(y)}$ is the cdf of $\chi^2_{Dim(\boldsymbol{y}_{\boldsymbol{i}})}$.

The i[th] observed probability (that is, the proportion of predicted probabilities greater than $\widehat{p}_i$) is

$$p_i^* = \frac{1}{|T|} \sum_{j=1}^{|T|} I(\widehat{p}_j \geq \widehat{p}_i) \tag{15}$$

Evaluating whether the model is over or underconfident is reduced to finding the predicted and observed probabilities for each input in *T* and comparing the two functions.

If the BNN is miscalibrated, one should consider changing the prior or investigating the inference algorithms; the prior may be unreasonable, or the inference algorithms may use a variational distribution which inadequately approximates the posterior.

References

[1] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. 2020. Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users. ACM Comput. Surv. 1, 1 (July 2020), 35 pages. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiEsazY4pDwAhUOUK0KHZmSCpQQFjAAegQIBRAD&url=https%3A%2F%2Farxiv.org%2Fpdf%2F2007.06823&usg=AOvVaw25GTTLFhXtQyZMNSIq5_2E

[2] Daniele Silvestro and Tobias Andermann. Prior choice affects ability of Bayesian neural networks to identify unknowns. CoRR, abs/2005.04987, 2020. URL http://arxiv.org/abs/2005.04987

[3] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. CoRR, abs/2002.08791, 2020. URL http://arxiv.org/abs/2002.08791

Derivation of Equation (12)

$$D_{KL}(q_\phi \| P) = \int_\Theta q_\phi(\theta') \log\left(\frac{q_\phi(\theta')}{P(\theta'|D)}\right) d\theta'$$

$$= -\int_\Theta q_\phi(\theta') \log\left(\frac{P(\theta'|D)}{q_\phi(\theta')}\right) d\theta'$$

$$= -\int_\Theta q_\phi(\theta') \log\left(\frac{P(\theta', D)}{q_\phi(\theta') P(D)}\right) d\theta'$$

$$= -\int_\Theta q_\phi(\theta') \left[\log\left(\frac{P(\theta', D)}{q_\phi(\theta')}\right) - \log(P(D))\right] d\theta'$$

$$= \log(P(D)) \int_\Theta q_\phi(\theta') d\theta' - \int_\Theta q_\phi(\theta') \log\left(\frac{P(\theta', D)}{q_\phi(\theta')}\right) d\theta'$$

$$= \log(P(D)) - ELBO$$