Bayesian Neural Networks

At a high level, Bayesian machine learning methods are characterized by two features that set them apart from traditional, frequentist approaches: the ability to incorporate domain knowledge through the use of a prior and probabilistic predictions, i.e. posterior distributions instead of point estimates.

Unfortunately, both of these present issues when attempting to incorporate Bayesian inference into neural networks: most kinds of prior knowledge about a particular machine learning task are not easily translated into a prior over the parameters of a neural network and posterior inference in a neural network is generally intractable. These notes will focus on the latter issue of how to perform posterior inference; in practice, the issue of prior selection is usually swept under the rug and simple, uninformative priors are used such as independent, broad Gaussian or Laplace distributions.

Neural Networks

In this lecture, we will primarily work with feedforward neural networks (although many of the approximate inference techniques detailed below are applicable to other network structures).

A feedforward neural network is specified by its architecture, $\mathcal{A}$, which consists of the number of hidden layers in the network, the number of nodes in each layer and the nonlinearity or activation function used between layers. An example of a (fully-connected) feedforward neural network is shown in Figure 1.

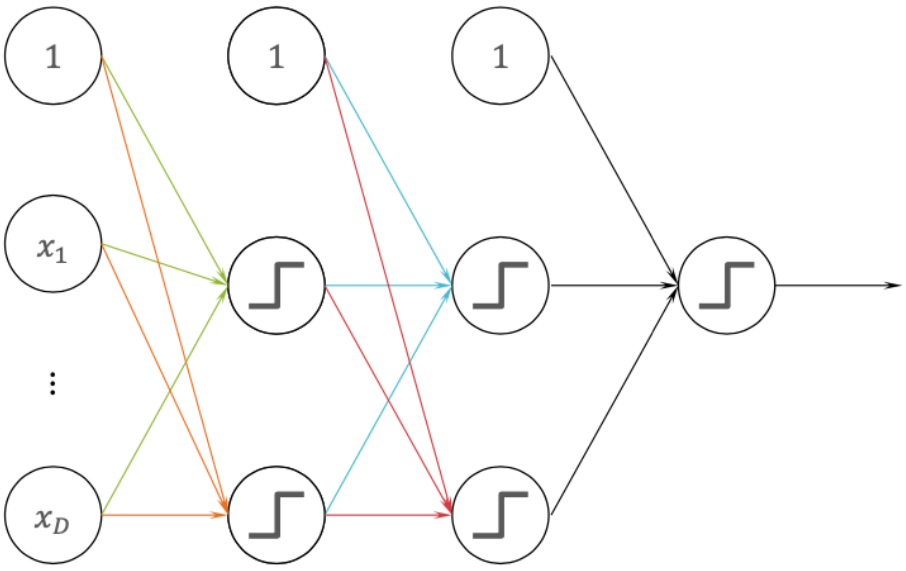

Figure 1: A fully-connected feedforward neural network over D-dimensional inputs with two hidden layers, two nodes in each hidden layer and an unspecified activation function represented by $\int$, typically something like tanh or ReLU.

We will treat $\mathcal{A}$ as fixed hyperparameters. Although it is possible to apply a fully Bayesian treatment to these values, this is typically not done in practice due to computational complexity. Instead, a more common approach is to treat different settings of $\mathcal{A}$ as models and apply Bayesian model selection (or averaging) when making predictions.

The parameters of interest in a feedforward neural network are the weights between nodes in different layers of the network, which we will represent as a vector $\mathbf{w}$. Then given some training data, $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, neural networks are typically trained by minimizing some error metric e.g., the squared error:

$$\mathcal{E}(\mathbf{w}) = \sum_{i=1}^{N} \big(f(\mathbf{x}_i; \mathbf{w}) - y_i\big)^2$$

where $f(\mathbf{x}_i; \mathbf{w})$ denotes the output of the neural network when the given $\mathbf{x}_i$ at the input layer and the weights are set to $\mathbf{w}$. While this is often a sufficient objective to learn with and achieve good results, many practitioners have observed that regularizing neural networks can improve their performance: there exist some strange forms of regularization for neural networks that are hard to express mathematically (e.g., dropout, early termination of optimization), but it turns out that simple L2 regularization can be applied to the weights of a neural network and have a similar effect as when applied to linear regression (i.e., preventing overfitting). We can express this regularized objective as

$$\mathcal{E}(\mathbf{w}; \alpha, \beta) = \frac{\alpha}{2} \sum_{i=1}^{N} \big(f(\mathbf{x}_i; \mathbf{w}) - y_i\big)^2 + \frac{\beta}{2} \sum_{w_j \in \mathbf{w}} w_j^2$$

where the parameters α and β are tuned to set the tradeoff between model fit and model complexity. The optimal weights,

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{E}(\mathbf{w}; \alpha, \beta)$$

can be (approximately) solved for using stochastic gradient descent (or other more complex methods such as ADAM) where the gradient of $\mathcal{E}$ w.r.t. $\mathbf{w}$ is computed efficiently using backpropagation.

A Bayesian Interpretation

Much like we saw with Bayesian linear regression, this regularized objective can be interpreted as a negative log likelihood for some implicit posterior that is being minimized to find the *maximum a posteriori* estimate for the parameters. The first term corresponds to a (negative log) likelihood for the data that assumes 1) the data are independent given the weights and 2) there is independent identically distributed additive Gaussian noise with variance $1/\alpha$. The second term can be interpreted as a prior over the weights, where each weight is an independent, identically distributed zero-mean Gaussian with variance $1/\beta$.

Formulated as such, we can treat the problem of setting the weights $\mathbf{w}$ as one of posterior inference and apply Bayes' rule:

$$p(\mathbf{w} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mathbf{w}) p(\mathbf{w})}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \mathbf{w}) p(\mathbf{w})}{\int p(\mathcal{D} \mid \mathbf{w}) p(\mathbf{w}) \, \mathrm{d}\mathbf{w}}.$$

Of course, this distribution is intractable so we will need to resort to approximate techniques.

Approximate Inference for Bayesian Neural Networks

A variety of approximate inference methods have been proposed for approximating the intractable posterior associated with the weights of a Bayesian neural network, many of which we have already seen in this course!

The Laplace Approximation for Bayesian Neural Networks

In one of the earliest works associated with Bayesian neural networks, Mackay (1992) proposed using a simple Laplace approximation for the intractable posterior:

$$p(\mathbf{w} \mid \mathcal{D}) \approx \mathcal{N}(\mathbf{w}; \mathbf{w}^*, \mathbf{H}^{-1})$$

where $\mathbf{w}^*$ is the MAP estimate of the weights and $\mathbf{H}$ is the negative Hessian matrix of $\mathcal{E}(\mathbf{w}; \alpha, \beta)$ evaluated at $\mathbf{w}^*$.

Theoretically, this matrix is easily computed using a variant of backpropagation as all the components of $\mathcal{E}$ are twice differentiable w.r.t. $\mathbf{w}$. In practice, computing the Hessian is much more computationally complex than just computing the gradient of $\mathcal{E}$ (which is typically all that is needed to optimize the weights): instead of just a single forwards and backwards pass through the network, computing the Hessian requires $\mathrm{O}(\nu)$ forwards and backwards passes where ν is the number of nodes in the network and each forwards and backwards pass requires $\mathrm{O}(W)$ operations where W is the total number of weights in the network. Furthermore, writing and storing this matrix requires $\mathrm{O}(W^2)$ operations and space which can be intractable for large enough networks.

There have been numerous approximations developed for the Hessian matrix associated with neural networks, both for Bayesian inference and for just generally optimizing the weights as many optimization methods can make use of second order partial derivatives. One common approximation is the (empirical) Fisher information matrix or the "squared" gradients:

$$\mathbf{H} \approx \nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}; \alpha, \beta) \, \nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}; \alpha, \beta)^\top$$

It can be shown that the Fisher information matrix is equal to the expected value of the negative Hessian w.r.t. the posterior distribution $p(\mathbf{w} \mid \mathcal{D})$. The empirical gradients can be computed in just a single forwards and backwards pass using the standard backpropagation algorithm but there is still the problem of inverting a full W-by-W matrix. A further simplification we can make to get around this is to just consider the diagonal elements of this matrix, functionally treating all the elements as independent:

$$\mathbf{H} \approx \mathrm{diag}\Big(\nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}; \alpha, \beta) \, \nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}; \alpha, \beta)^\top \Big)$$

where the diag operator just extracts the diagonal elements of the matrix; this approximation only requires $\mathrm{O}(W)$ time to invert.

Variational Inference for Bayesian Neural Networks

Perhaps the most common way to approximate $p(\mathbf{w} \mid \mathcal{D})$ is through variational inference: recall that the premise of variational inference is that we approximate an intractable posterior $p(\mathbf{w} \mid \mathcal{D})$ using some simple, parametrized distribution $q(\mathbf{w}; \theta)$. The parameters θ are tuned so as to minimize the KL divergence between the true and approximate posteriors:

$$\begin{aligned}
\theta^* &= \arg\min_\theta \big[q(\mathbf{w};\theta) \| p(\mathbf{w} \mid \mathcal{D})\big] \\
&= \arg\min_\theta \int q(\mathbf{w};\theta) \log \frac{q(\mathbf{w};\theta)}{p(\mathbf{w} \mid \mathcal{D})} \,\mathrm{d}\mathbf{w} \\
&= \arg\min_\theta \int q(\mathbf{w};\theta) \log \frac{q(\mathbf{w};\theta)\, p(\mathcal{D})}{p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})} \,\mathrm{d}\mathbf{w} \\
&= \arg\min_\theta \int q(\mathbf{w};\theta) \log \frac{q(\mathbf{w};\theta)}{p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})} \,\mathrm{d}\mathbf{w} \\
&= \arg\min_\theta \mathbb{E}_{q(\mathbf{w};\theta)}\big[\log q(\mathbf{w};\theta) - \log p(\mathbf{w}) - \log p(\mathcal{D} \mid \mathbf{w})\big].
\end{aligned}$$

The final expression is the negative ELBO for this setting. One method commonly applied to Bayesian neural networks is stochastic optimization, where we directly minimize the negative ELBO using gradient descent. The gradient of interest can be expressed as

$$\begin{aligned}
&\nabla_\theta \int q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big) \,\mathrm{d}\mathbf{w} \\
&= \int \nabla_\theta q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big) + q(\mathbf{w};\theta)\big(\nabla_\theta \log q(\mathbf{w};\theta)\big) \,\mathrm{d}\mathbf{w} \\
&= \int \nabla_\theta q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big) \,\mathrm{d}\mathbf{w} + \int \nabla_\theta q(\mathbf{w};\theta) \,\mathrm{d}\mathbf{w}
\end{aligned}$$

where we have used the chain rule:

$$\nabla_\theta \log q(\mathbf{w};\theta) = \frac{\nabla_\theta q(\mathbf{w};\theta)}{q(\mathbf{w};\theta)} \to \nabla_\theta q(\mathbf{w};\theta) = q(\mathbf{w};\theta)\, \nabla_\theta \log q(\mathbf{w};\theta).$$

We will use this second identity to replace the $\nabla_\theta q(\mathbf{w};\theta)$ in the first term above; continuing gives

$$\begin{aligned}
&\int \nabla_\theta q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big) \,\mathrm{d}\mathbf{w} + \int \nabla_\theta q(\mathbf{w};\theta) \,\mathrm{d}\mathbf{w} \\
&= \int q(\mathbf{w};\theta)\, \nabla_\theta \log q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big) \,\mathrm{d}\mathbf{w} + \nabla_\theta \int q(\mathbf{w};\theta) \,\mathrm{d}\mathbf{w} \\
&= \int q(\mathbf{w};\theta)\, \nabla_\theta \log q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big) \,\mathrm{d}\mathbf{w} + \nabla_\theta 1 \\
&= \mathbb{E}_{q(\mathbf{w};\theta)}\Big[\nabla_\theta \log q(\mathbf{w};\theta)\big(\log q(\mathbf{w};\theta) - \log p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})\big)\Big] + 0 \\
&\approx \sum_{s=1}^{S} \nabla_\theta \log q(\mathbf{w}_s;\theta)\big(\log q(\mathbf{w}_s;\theta) - \log p(\mathbf{w}_s)\, p(\mathcal{D} \mid \mathbf{w}_s)\big)
\end{aligned}$$

where in the last line we have made a Monte Carlo approximation to the expected value using samples $\mathbf{w}_1, \ldots, \mathbf{w}_S$ from the distribution $q(\mathbf{w}; \theta)$. We can use this approximate gradient for gradient descent and given certain conditions on the step size, it has been shown that gradient descent will converge on a local minimum for this objective. Unfortunately, the estimator above, while unbiased, tends to have a very large variance, meaning that we either need tons of samples in each iteration of gradient descent or tons of iterations of gradient descent in order to converge.

Various methods have been developed to decrease the variance of this estimator, the most famous of which makes use of the so-called "re-parameterization" trick giving rise to the "Bayes-by-Backprop" algorithm (Blundell et al., (2015)); this is effectively the same technique used to train variational autoencoders with a key difference being that in Bayes-by-Backprop, the latent variables are the parameters of the network as opposed to the latent codes for each data point.