Thomas Larsen

Bayesian Neural Networks

1 Overview

The classic formalization of a Neural Network that it is a function $NN_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta} = \{\theta_1, b_1 \dots \theta_n, b_n\}$ is a collection of all the trainable parameters, which are the weights associated with each later. Then

$$\begin{aligned} L_0 &= x \\ L_{i+1} &= a(\theta_{i+1} \cdot L_i + b_i) \\ NN_{\boldsymbol{\theta}}(x) &= L_n \end{aligned}$$

Where a is a nonlinear activation function. Given some training set $\mathcal{D}$, a common problem is how to infer the best set of weights, $\boldsymbol{\theta}$, so that $NN_{\boldsymbol{\theta}}(x)$ performs as well as possible. Using a loss function $L : \{\boldsymbol{\theta}\} \to \mathbb{R}_{\geq 0}$, one can find $\widehat{\theta} = argmin_\theta\, L(\theta)$ to minimize the loss. This is commonly done with Stochastic Gradient Descent (SGD). Unfortunately, these networks have problems.

> The point estimate approach is relatively easy (with modern algorithms and software packages), but tends to lack explainability and might generalize in unforeseen and overconfident ways on out-of-training-distribution data points [27, 63]. This property, and inability of ANNs to answer "I don't know" is problematic in fields where their predictions have critical implications, such as trading, autonomous driving or medical applications. Techniques exist to mitigate this risk [28] based either on a threshold for the softmax-predicted class logit or an additional module to classify out-of-distribution samples. Another method for out-of-distribution detection is the use of Deep Generative Models, a class of ANNs (e.g., Generative Adversarial Networks) meant to encode complex data distributions [79]. Concerns have, however, been raised about these different approaches, either because they are too simple, like a threshold on the softmax logits, or because the additional module or deep generative models used for out-of-distribution detection might themselves suffer from the same overconfidence problems they where supposed to fix in the first place [60]. The flaws of these different approaches is one of the main motivations for the introduction of stochastic neural networks. *From: Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users, page 4*

Definition 1.1. A Stochastic Neural Network is a neural network that has a probability distribution $p(\boldsymbol{\theta})$ over possible weights, instead of a single deterministic set of weights.

One can evaluate a Stochastic Neural Network as follows:

$$y = \int_{\boldsymbol{\theta}} NN_{\boldsymbol{\theta}}(x) p(\boldsymbol{\theta})$$

This is a very natural average over the estimates of each of the possible Neural Networks $NN_{\boldsymbol{\theta}}$, weighted by how probably they are. In practice, this is intractable (or just impossible, since there are infinitely many possible models?), so one instead can simply sample from the distribution to find a fixed set of weights, and predicting as a normal Neural Network.

$$\begin{aligned} \boldsymbol{\theta} &\sim p(\boldsymbol{\theta}) \\ y &= NN_{\boldsymbol{\theta}}(x) \end{aligned}$$

Of course, a better, but more computationally intensive approach is to average the prediction of N

samples:

$$\boldsymbol{\theta}_1 \dots \boldsymbol{\theta}_N \sim p(\boldsymbol{\theta})$$
$$y = \frac{1}{N}\sum_i^N NN_{\boldsymbol{\theta}_i}(x)$$

The variance of these N predictions allows one to get an estimate for the true variance encoded by a network. We denote Θ to be a collection of samples from p.

2 Bayesian Neural Network Basics

Definition 2.1. A Bayesian Neural Network (BNN) is a stochastic neural network that is trained using Bayesian Inference. In particular, one explicitly chooses a prior $p(\boldsymbol{\theta})$ over the possible model parameters, and a prior confidence $p(y|x, \boldsymbol{\theta})$.

The term $p(y|x, \boldsymbol{\theta})$ encodes the models predicted distribution given a specific input x and $\boldsymbol{\theta}$. This term captures the aleatoric uncertainty of the model. A normal NN, on a regression class outputs a number, a BNN outputs parameters that encode the distribution $p(y|x, \boldsymbol{\theta})$ over the possible numbers. When a normal NN asked to sort things into 10 classes outputs an element $\widehat{p} \in \delta^1 0$ (a 10-dimensional probability vector), we instead explicitly output a distribution over $\delta^1 0$.

Theorem 2.2. *The posterior of a BNN is*

$$p(\boldsymbol{\theta}|D) = \frac{p(D_y|D_x, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}'} p(D_y|D_x, \boldsymbol{\theta}')p(\boldsymbol{\theta}')}$$

The numerator encapsulates how well a given θ predicts the training data D along with the prior likelihood of that θ. The denominator is simply a scaling factor, representing how likely the observed data was given your prior. One cannot compute this directly: in particular the denominator involves an integral over a possibly infinite set of $\boldsymbol{\theta}$s. Instead, one can use either a Markov Chain Monte Carlo, or a Variational Inference approach. (Neither of which I understand right now, but that I will understand soon).

Definition 2.3. Given an input x, and a collection of samples the covariance matrix of x is:

$$\Sigma_x = \frac{\sum_i (\widehat{y_i} - \widehat{y})(\widehat{y_i} - \widehat{y})^T}{N - 1}$$

where y is the mean prediction, and y_i is the prediction of the i-th model in Θ. This is just the covariance matrix of the sampled predictions.

3 Inference

Recall that the posterior of a BNN is intractable to directly compute. Instead, we must approximate it.

There are two common approaches here: Variational Inference and Markov Chain Monte-Carlo (MCMC). The method applicable to deep learning is Variational Inference, whose key idea is that instead of directly sampling from the posterior, we learn a distribution, q, paramaterized by $\boldsymbol{\phi}$,

such that $q_{\boldsymbol{\phi}}$ is as close as possible to the posterior $p(\boldsymbol{\theta}|D)$. q often is either a multivariate normal, gamma, or Dirchlet distribution. Explicitly, we try to minimize the KL divergence:

$$KL(q_{\boldsymbol{\phi}}||P) = \int_{\boldsymbol{\theta}} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}')log(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{\theta}')}{p(\boldsymbol{\theta}'|D)})$$

This seems nice, until we realize that there is a posterior $p(\boldsymbol{\theta}|D)$ term in the KL-divergence, which is exactly what we were trying to avoid computing. So this doesn't seem to get us anywhere, until we have the clever idea of multiplying by a constant, P(D), to avoid computing the posterior $p(\boldsymbol{\theta}'|D)$. This allows us to instead compute the joint distribution $p(\boldsymbol{\theta}, D) = p(D|\boldsymbol{\theta})p(\boldsymbol{\theta})$, yielding:

$$ELBO = \int_{\boldsymbol{\theta}} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}')log(\frac{p(\boldsymbol{\theta}', D)}{q_{\boldsymbol{\phi}}(\boldsymbol{\theta}')}) = log(p(D)) - KL(q_{\boldsymbol{\phi}}||P)$$

Minimizing KL-divergence is equivalent to maximizing the evidential lower bound (ELBO) since $log(p(D))$ is constant over $\boldsymbol{\phi}$.

4 Uncertainty

One of the primary benefits of BNNs is that they allow us to quantify the uncertainty. We commonly talk about two key types of uncertainty.

Definition 4.1. The Epistemic Uncertainty of a BNN is the distribution $p(\boldsymbol{\theta}|D)$. This represents uncertainty due to lack of training data, and so as the training set D increases, this probability distribution has less variance.

Accounting for epistemic uncertainty happens when we have a probability distribution over weights, instead of point wise weights. Note that epistemic uncertainty involves computing the posterior, we so normally only have an estimate. Out of distribution datapoints will lead to high epistemic uncertainty. (TODO: why?) Shouldn't the epistemic uncertainty not change based on a given example, and instead the aleatoric uncertainty be high? I guess since aleatoric uncertainty is irreducible while epistemic is not?

Definition 4.2. The Aleatoric Uncertainty of a BNN is the distribution $p(y|x, \boldsymbol{\theta})$, which is uncertainty due to noise.

By having a distribution over model parameters, one captures the epistemic uncertainty, but this does not help to capture aleatoric uncertainty.

- In a basic SNN, we just sample the model parameters and then evaluate the posterior of each sampled model to sample the output distribution, and then one can compute sample variance and other statistics.
- In a BNN, we explicitly output parameters for a distribution instead of a point estimate. This requires new loss function that punishes how badly a model predicts this: use log-likelyhood.